Searching for Sentient Design Tools for Game Development

By

Antonios Liapis

Dissertation

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in the Center for Computer Games Research at IT University of Copenhagen, 2014

Copenhagen, Denmark

Sentient 6

Nevermore

Lyrics by Jeff Loomis and Warrel Dane

I am sentient number six, I stand in line
I am the prototype of a benign convenience for mankind
Superior is digital, human flesh so trivial
I hate that I can't see the one that made me

I am the new awakening of different eyes
My children, you are my army
They are what we can never see and still despise
And their sky cries Mary

Trained, I see imperfection in your race
Lying in wait, blind I suffer knowing I'll never reach your heaven

Why is this control, behavior based and reactive
Adapting to every new environment?
Rewarded when I replicate, isolate and mutate
To assimilate a fragmented plea for ego

Trained I see imperfection in your race
Lying in wait, blind I suffer knowing I'll never reach your heaven
It's unattainable, please teach me how to dream
I long to be more than a machine

Acknowledgments

I would first like to thank my supervisor Georgios N. Yannakakis for his trust in my vision and his subtle guidance towards achieving that; his input and suggestions helped shape this thesis and the research output that came along with it. I would also like to thank Julian Togelius for his extensive feedback, discussion and collaborations that we had over the years of my studies. I am also very grateful to the Center for Computer Games Research and all its members for providing guidance as well as helpful criticism based on their different viewpoints on games, research, and design. More specifically, I would like to thank my colleagues Héctor P. Martínez, Christoffer Holmgård, Mark J. Nelson, Miguel Angel Sicart Vila, Espen Aarseth, Rilla Khaled, Florian Berger, Noor Shaker, Paolo Burelli, Corrado Grappiolo and so many more to list. I would especially thank the members of my thesis proposal committee, Miguel Angel Sicart Vila, Susana Tosca, and Mark J. Nelson, for their feedback during those early stages of my research. Of course I also thank Michael Matheas, Penousal Machado and Kasper Støy for being in my thesis committee. It would be remiss of me if I did not thank the IT University of Copenhagen for believing in me, supporting me and the Center, and providing a great work environment with a friendly, informal atmosphere. Of special note are the PhD Administrators Christina Rasmussen, Freja Krab Koed Eriksen and Jane Andersen — along with the PhD council — for supporting all my travels to conferences across the world and for being responsive and flexible when that was necessary. I would also like to thank Kenneth O. Stanley and the EPLEX research group in the University of Central Florida for hosting me during my PhD stay abroad and for providing me with excellent feedback and supervision. Finally, I would like to thank my friends Rune Kristian Lundedal Nielsen, Christian Sinding Nellemann, Rasmus Bille Fynbo, Athanasios Kastanidis and Lena Liapi for keeping me sane during the more laborious periods of my studies, as well as my family for their support over the years. This thesis was a grand undertaking, and it would have been very different without all these people and many more which I have surely overlooked.

Searching for Sentient Design Tools for Game Development

Antonios Liapis

Center for Computer Games Research
IT University of Copenhagen
Copenhagen, Denmark
2014

ABSTRACT

Over the last twenty years, computer games have grown from a niche market targeting young adults to an important player in the global economy, engaging millions of people from different cultural backgrounds. As both the number and the size of computer games continue to rise, game companies handle increasing demand by expanding their cadre, compressing development cycles and reusing code or assets. To limit development time and reduce the cost of content creation, commercial game engines and procedural content generation are popular shortcuts. Content creation tools are means to either generate a large volume of game content or to reduce designer effort by automating the mechanizable aspects of content creation, such as feasibility checking. However elaborate the type of content such tools can create, they remain subservient to their human developers/creators (who have tightly designed all their generative algorithms) and to their human users (who must take all design decisions), respectively.

This thesis argues that computers can be creative partners to human designers rather than mere slaves; game design tools can be aware of designer intentions, preferences and routines, and can accommodate them or even subvert them. This thesis presents Sentient Sketchbook, a tool for designing game level abstractions of different game genres, which assists the level designer as it automatically tests maps for playability constraints, evaluates and displays the map's gameplay properties and creates alternatives to the user's current design in order to speed up the creation process and inspire the user to think outside the box. Several AI techniques are implemented, and others invented, for the purposes of creating meaningful suggestions for Sentient Sketchbook as well as for adapting these suggestions to the user's own preferences. While the thesis focuses on the design, performance, and human use of Sentient Sketchbook, the same algorithms and concepts can be applied to different mixed-initiative tools, a subset of which has been implemented and is presented in this thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last twenty years, digital games have grown from a niche market targeting young adults to an important player in the global economy, engaging millions of people from different cultural backgrounds [67]. Since the early arcade games such as *Pong* (Atari 1972) or *Pacman* (Namco 1980) the number of digital games has been increasing exponentially every year. Games nowadays are being played on the personal computer, on mobile phones, on home or hand-held consoles; they are played with mouse and keyboard, with dedicated game controllers, via natural movement, via a touch interface or even via virtual reality head-mounted displays such as the *Occulus Rift* (Occulus 2012). Games are played on the bus (mobile games), on the couch with friends (home consoles), on social media with online friends or in sprawling virtual worlds with vast populations of like-minded gamers (massive-multiplayer online games). Games can captivate their audience with an engrossing plot and dialogs as with *Planescape: Torment* (Black Isle 1999), with stunning visuals as with *Mirror's Edge* (Electronic Arts 2008), with frantic, adrenaline-fueled gameplay as with *Left 4 Dead* (Valve 2008), with difficult choices as with *Endless Space* (Amplitude 2012), with enchanting soundtracks as with *Proteus* (Key and Kanaga 2013) and as many more factors as there are players.

A game's aesthetic quality can be therefore determined by several factors such as its art style, the pacing of its interactions, the emotions evoked by in-game or pre-scripted events, or the departures from typical games of its genre. All of these elements can be traced back to creative members of the design and art team: game designers, level designers, visual and sound artists or writers. The design process of these creators differs depending on the type of creative task, the structure of the design team and ultimately on the preferences and vision of each individual. However, a unifying trait in modern game development is the almost ubiquitous use of the computer and its sophisticated computer-aided design tools. From game editors such as the *Unreal Development Kit* (Epic Games 2009) to graphic design tools such as *Photoshop* (Adobe 1990) and from brainstorming tools such as *Mindjet* (Mindjet 2012) to procedural generators such as *SpeedTree* (IDV 2002), computer-aided design tools are used in almost all creative tasks in order to minimize development time and cost, reduce human effort, support collaboration among members of a design team or elicit a user's creativity.

While many of the computer-aided design tools used during game development are commercially successful products with many applications beyond games, such as *Photoshop* (Adobe 1990) and *3D Studio Max* (Autodesk 1996), the more game-specific tools are often implemented from scratch in-studio or re-appropriated from previous games. The latter game editors and in-house scripting languages are often shipped along with the game, in an attempt to get the player community involved in creating new assets and mods. Examples of such editors include the *Warcraft III World Editor* (Blizzard 2002), the *Elder Scrolls Construction Set* (Bethesda 2002) and *Unreal Engine 3* (Epic Games 2004); their success in amassing a vast database

of user-generated content is an indication of their ease-of-use and modularity, but also of an increasing enthusiasm among end-users in getting involved with the making and tweaking of their favorite games.

The growing popularity of digital games attracts individuals who do not necessarily possess programming or game design experience. Such individuals often wish to contribute to existing games with user-generated content, such as custom levels, or with more extensive modifications which significantly change the gameplay (game mods). Since popular game editors have an intuitive interface and a thorough documentation (often with the help of more mature contributing modders), the programming skill required from end-users is considerably lessened. However, these tools still assume that the prospective creators have enough ingenuity, drive and experience with digital games to see an idea from its conception to its completion.

1.1 Searching for Sentient Design Tools

Computer-aided design (CAD) tools — within games and beyond — have introduced a mixed-initiative co-creation (MI-CC) paradigm in the workplace but also among design novices. Mixed-initiative co-creation, within the context of this thesis, is identified as the task of creating artifacts via the interaction of a human initiative and a computational initiative. Although mixed-initiative lacks a concrete definition [176], MI-CC considers both the human and the computer *proactively* making contributions to the problem solution, although the two initiatives do not need to contribute to the same degree. MI-CC thus differs from other forms of co-creation, such as the collaboration of multiple human creators or the interaction between a human and non-proactive computer support tools (e.g. spell-checkers or image editors) or non-computer support tools (e.g. artboards or idea cards).

This thesis focuses on game development tasks — mostly for level design — where co-creation occurs between a human and a machine in a mixed-initiative fashion, and contends that such tasks can benefit from more proactive computer-aided design tools, where the computer has initiative in most creative efforts. Moreover, this thesis argues that computational initiative within MI-CC can support and realize mixed-initiative *co-creativity*, thus fostering human creativity. Due to the very nature of the MI-CC type examined (visual design), creativity refers to aspects of lateral thinking [55] and diagrammatic reasoning [36], which will be laid out in the following Chapters. The hypothesis is that a human designer interacting with a computational designer that is deemed to be creative is not merely assisted during the creation process; instead, under those circumstances MI-CC *fosters* the designer's creativity. However, a perception of computational creativity is achievable through various forms of computational initiative; this thesis explores such roles for the computational designer that include iterative refining, interactive evolution, designer modeling and coterminous design of alternatives to human designs. In order to accommodate both a human designer and a computational creator, several innovations are necessary both on the interface but also on the algorithmical level. The goal of the thesis is to find the right software application which allows the human user to take advantage of computer-generated suggestions without being overburdened cognitively with too much information, without being forced into overwhelming loading times, and without losing the creative control (or perception thereof) of their design. In part motivated by the requirements of real-time computational input to human designs, existing and

new genetic algorithms capable of autonomously creating game artifacts are tailored to the task and runtime requirements of the mixed-initiative tools. In order to enhance the quality of computational input in the design process, a number of designer models are implemented to adapt the generative algorithms or the evaluation of designed artifacts according to the preferences, goals and process of their human user.

The main contribution of this thesis is the implementation and evaluation of Sentient Sketchbook, a computer-aided level design tool which allows quick and effortless prototyping of level concepts by operating on sketches rather than detailed maps. Sentient Sketchbook imparts some expert knowledge regarding level design to (potentially novice) end-users by providing several evaluations of game level quality via heuristics and visual aids. Moreover, Sentient Sketchbook can foster the creativity of its users via computer-generated suggestions evolved from the users' own creations; the suggestions generated by the software are presented in real-time, while the human user interacts with the tool. Prior to or coterminously with the development of Sentient Sketchbook, a number of MI-CC prototypes have been implemented in an attempt to explore varying degrees of human and computational initiative, and are presented briefly within this thesis.

1.2 Research Question

Fundamentally, the research question behind this line of research can be summarized as:

> *"How can the computer become a proactive partner in the game design process?"*

It is naive, however, to expect an answer to this question within the scope of a single thesis. Several PhD theses [222, 218] have wrestled with this particular question already. Even more so, omitting the word "game" from this question describes the holy grail of computer-aided design tools (as well as mixed-initiative design tools and creativity-support tools, as the delineation is unclear) and likely human-computer interaction in general [148].

Without losing sight of the ultimate — if lofty — goal laid out above, how can this research question be narrowed down so that it can be answered within a PhD thesis? Here is a set of more specific questions:

Q1. Under which circumstances can search-based algorithms be integrated into the design process? How will they impact design?

Q2. What impact do different methods of search have on the resulting computational initiative?

Q3. What impact does computational initiative have on the creativity of a human designer?

Q4. How can a computer learn from a human user and adapt its own initiative to the needs of the current design process?

Q5. Wherein lie the limits for computational initiative in a mixed-initiative design tool?

This thesis sets out to explore the limits of computational initiative in design by implementing prototypes for game design tasks. These prototypes act as stepping stones towards the "proactive computational partner" of the larger research question, and each one informed the design decisions for the next prototype; however, the successes and failings of these prototypes inadvertently serve to address Q5. Most of these prototypes revolve around the subtask of level design. The main reason for this choice of domain is that the visual representation of levels lends itself well to the interface design of many CAD tools. Moreover, computational input can be displayed visually in a format which is perceived with less cognitive effort by human users than text. Finally, visual display of game levels is ideal for realizing diagrammatic lateral thinking [269], whether this occurs on the visual level (the imagistic impression of the artifact) or on the analogical level (where colors represent units with playability affordances and constraints).

While developing the mixed-initiative design prototypes, a number of algorithms were implemented in order to answer one or more of the questions above. Search was performed guided by playability constraints, visual novelty and objectives of game level quality in an attempt to answer Q2. Adaptive models for guiding search according to designer preference were devised and tested in several prototypes in order to answer Q4. Sentient Sketchbook, as the most mature (and arguably elaborate) prototype, was tested with expert human game developers; their design sessions, interactions and post-test feedback serve to address Q3. Admittedly, it is hard to make claims on having answered any of these questions (let alone the larger research question); however, the extensive experiments on different ways of incorporating AI into design tasks have provided sufficient wisdom on the next steps which need to be taken towards a computational designer that can be as creative — and as inspiring — as its human counterpart.

1.3 Contributions

This thesis makes several contributions to existing research, summarized below:

- The demonstration of sketching as a method of rapid level prototyping in Sentient Sketchbook, as well as the potential of mixed-initiative co-creativity within computer-generated suggestions evolving along the human's creative process.
- The introduction of two-population novelty search, which combines the desirable properties of objective-free search with a degree of control in the resulting artifacts via constraints.
- The introduction and formalization of *designer modeling* as a concept, i.e. adapting the computational input to the design process according to the goals, process, preferences and affective states of the human user. A demonstrative prototype of designer modeling is implemented in Sentient Sketchbook and evaluated through *in vitro* experiments.
- The integration of novelty search as a method for seeding neural networks for the iterative refining of map sketches in Sentient World.

- The combination of interactive and objective-driven evolution via adaptive models, and the use of human rankings to drive content generation.
- The invention of computer-computer collaboration in DeLeNoX, where distinct computational inputs act as designers and critics.

1.4 List of publications

While attempting to answer the questions laid out in Section 1.2, a number of papers were published as listed below. While each paper attempts to address different facets of mixed-initiative game design, this thesis is largely based on the concepts described within these papers — and motivated by the findings reported therein.

1. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Adapting models of visual aesthetics for personalized content creation," IEEE Transactions on Computational Intelligence and AI in Games 4(3), 2012, pp. 213–228.
2. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Limitations of Choice-Based Interactive Evolution for Game Level Design," in Proceedings of the AIIDE Workshop on Human Computation in Digital Entertainment, 2012.
3. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Generating Map Sketches for Strategy Games," in Proceedings of Applications of Evolutionary Computation, vol. 7835, LNCS. Springer, 2013, pp. 264–273.
4. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Sentient World: Human-Based Procedural Cartography," in Proceedings of Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMusArt), vol. 7834, LNCS. Springer, 2013, pp. 180-191.
5. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Sentient Sketchbook: Computer-Aided Game Level Authoring," in Proceedings of the 8th Conference on the Foundations of Digital Games, 2013, pp. 213–220.
6. Antonios Liapis, Héctor P. Martínez, Julian Togelius, Georgios N. Yannakakis: "Transforming Exploratory Creativity with DeLeNoX," in Proceedings of the Fourth International Conference on Computational Creativity, 2013, pp. 56-63.
7. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Enhancements to Constrained Novelty Search: Two-Population Novelty Search for Generating Game Content," in Proceedings of Genetic and Evolutionary Competition Conference, 2013, pp. 343-350. **[Best Paper Award of the DETA/Self* track.]**
8. Antonios Liapis, Héctor P. Martínez, Julian Togelius, Georgios N. Yannakakis: "Adaptive Game Level Creation through Rank-based Interactive Evolution," in Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2013. **[Nominated for Best Paper Award.]**

9. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Towards a Generic Method of Evaluating Game Levels," in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2013. **[Best Student Paper Award]**

10. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Designer Modeling for Personalized Game Content Creation Tools," in Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics, 2013.

11. Georgios N. Yannakakis, Antonios Liapis, Constantine Alexopoulos: "Mixed-Initiative Co-Creativity," in Proceedings of the 9th Conference on the Foundations of Digital Games, 2014.

12. Christoffer Holmgård, Antonios Liapis, Julian Togelius, Georgios N. Yannakakis: "Generative Agents for Player Decision Modeling in Games," in Poster Proceedings of the 9th Conference on the Foundations of Digital Games, 2014.

13. Julian Togelius, Mark J. Nelson, Antonios Liapis: "Characteristics of Generatable Games," in Proceedings of the FDG Workshop on Procedural Content Generation, 2014.

14. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Constrained Novelty Search: A Study on Game Content Generation," accepted for publication in Evolutionary Computation.

15. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Computational Game Creativity," in Proceedings of the Fifth International Conference on Computational Creativity, 2014.

16. Antonios Liapis, Georgios N. Yannakakis, Julian Togelius: "Designer Modeling for Sentient Sketchbook," in Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2014. **[Best Paper Award]**

17. Mike Preuss, Antonios Liapis, Julian Togelius: "Searching for Good and Diverse Game Levels," in Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2014.

18. Christoffer Holmgård, Antonios Liapis, Julian Togelius, Georgios N. Yannakakis: "Evolving Personas for Player Decision Modeling," in Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2014.

19. Christoffer Holmgård, Antonios Liapis, Julian Togelius, Georgios N. Yannakakis: "Personas versus Clones for Player Decision Modeling," in Proceedings of the International Conference on Entertainment Computing (ICEC), 2014.

1.5 Thesis Structure

Since this thesis tackles the problem of human and computational input co-creating in a mixed-initiative fashion, it is challenging to find a balance between presenting human design interfaces and algorithmical innovations to support them. As such, the thesis consists of four parts describing related work in similar areas or techniques, the Sentient Sketchbook

user interface and how designers have interacted with it, a more thorough presentation of the algorithms implemented for Sentient Sketchbook and their performance evaluation, and other tools or methods for mixed-initiative design. Related work is presented in Chapter 2, which presents the research areas touched upon by this thesis, and Chapter 3, which provides an overview of artificial evolution and machine learning methods that the algorithms of this thesis fall into. The algorithmic techniques used in this thesis are covered in Chapter 4, while each of the three techniques are evaluated through extensive experimentation in Chapter 7 for constrained optimization via FI-2pop GAs, Chapter 8 for constrained novelty search via FINS and FI2NS, and Chapter 9 for adaptive models of user taste. The design of the interface for Sentient Sketchbook, and details of how the algorithms of Chapter 4 are integrated in it, is provided in Chapter 5; the design sessions where Sentient Sketchbook was used by experts in the game industry are analyzed in Chapter 6. Other tools and protocols implemented during this thesis — either prior to or concurrently with the development of Sentient Sketchbook — for testing how far computational initiative can extend in a mixed-initiative setting are elaborated upon in Chapter 10. The thesis concludes with Chapter 11, which discusses the limitations found in the algorithms, the domains, and the experiments at large and proposes extensions which can address these limitations and transfer the thesis' findings to different instances of design.

Many elements of the thesis are based on previous publications listed in Section 1.4: the algorithms of Chapter 4 have been reported in publications 3,7,14 and 8 in the above list. The description of Sentient Sketchbook in Chapter 5 is based on publication 5 and its user test was reported in 5 and 12. While the experiments in Chapter 7 have not been reported in previous publications, they are inspired by those of publication 9; experiments of Chapter 8 are reported in publication 14, and most of the experiments of Chapter 9 are reported in 16. The other mixed-initiative prototypes of Chapter 10 have been reported in 4, 8 and 6 while the algorithms and related domains of Chapters 2 and 3 are re-appropriated from many publications.

1.6 Summary

This Chapter lays down the fundamental questions that this report aims to answer. Describing the problem of computer-aided design within game development, the need for proactive computational designers which can enhance both the process and the creativity of the human user is asserted. Several complementary outlooks on how computational initiative can be enhanced and integrated into design are enumerated. The remainder of this thesis sets out to answer the questions and arguments raised in this Chapter as reliably as possible.

Chapter 2

Related Domains

The core characteristic of mixed-initiative design tools is the inclusion of sophisticated computational creators working alongside human designers. The design problem chosen is that of game design, and Section 2.1 summarizes the creative tasks of game design and game development. Section 2.2 provides an overview of the history, motivations and most important works in computer-aided design within and beyond games. Granted the importance of an *intelligent* computational creator, Section 2.3 provides a brief survey of what game artificial intelligence has been focusing on in the past and present. Since the computational creator in a mixed-initiative tool must be able to be autonomously creative without human input, Section 2.4 presents the dominant techniques for procedurally generating content for games. The issue of recognizing and accommodating a user — or in the case of games, a *player* — falls under user/player modeling in Section 2.5, which elaborates on these concepts and presents some of the work in these fields. On the other hand, the larger problem of autonomous computational creativity has its own research community, which tends to favor procedurally generating works of art, music or narrative, i.e. fields which are more easily considered to require human-level creativity. The concepts of computational creativity and works of generative art are presented in Section 2.6. Finally, a brief discussion of human creativity and the principal theories which lend themselves well to the hypothesis of mixed-initiative co-creativity of this thesis are provided in Section 2.7.

2.1 Game Design and Development

Creating a commercially viable digital game is a massive undertaking, most importantly due to the need for the game to incorporate stunning visuals, an immersive soundtrack, and most importantly an entertaining gameplay experience. While games may be awarded for their storyline or technical achievements, at the core of it all is the game design — i.e. defining the rules, mechanics, winning conditions, interaction methods which turn playing the game into an engaging, fun experience. Given a vast literature on the definition of *games* [31, 5, 236, 202, 74, 120], the definition of game *design* is likely as contentious (given that design itself describes a rather broad set of activities); this thesis uses the definition of game design as "the act of deciding what a game should be" [204, Page xxiv]. While most game designers focus on the rules, mechanics, winning and losing conditions of a game, the above definition is important — despite its ambiguity — because game design needs to account for all of the constituents of the game, from its historical background to its mood and from its departures from current genres to its intended message. This thesis uses this broad definition of the term "game design", although the prototypes presented within it only scrape few of the facets

which comprise it. This Section will briefly touch upon the different facets which should be considered when designing a game.

Visuals: As the vast majority of digital games are displayed on a screen, games primarily rely on visual output to convey information to the player. Game visuals can range from photorealistic, to caricaturized, to abstract [116]. While photorealistic visuals as those in the *FIFA* series (EA Sports 1993) are direct representations of objects, in cases where no real-world equivalent exists (such as in fantasy or sci-fi settings) artists must use real-world reference material and extrapolate them to fantastical lengths with "what if" scenarios. Caricaturized visuals often aim at eliciting a specific emotion, such as melancholy in the black and white theme of *Limbo* (Playdead 2010). Abstract visuals include the 8-bit art of early games, where constraints of the medium (low-tech monitors) forced game artists to become particularly creative in their design of memorable characters using as few pixels or colors as possible.

Audio: While often overlooked when discussing games, a game's audio is an important contributor to the overall experience; its recognition is demonstrated by two BAFTA Game Awards (music and sound) and, briefly, by a MTV Video Music Award for Best Video Game Soundtrack. Game audio usually includes background music, sound effects and voice-acted dialog. Game audio usually includes background music such as the fully orchestrated soundtrack of *Skyrim* (Bethesda 2011), sound effects such as the pellet-eating sound from *Pac-Man* (Namco 1980) or the rewarding sounds of *Bejeweled* (Popcap 2001), and voice-acted dialog which is deemed essential for large-scale commercial games and often includes Hollywood names such as Liam Neeson in *Fallout 3* (Bethesda 2008).

Narrative: The term 'narrative' here is used loosely to include the game's main story arc as well as quests and NPC dialog. Not all games include narrative (as per the definition above) if they favor short interactions (e.g. casual or sandbox games); however, many AAA games rely on an engrossing plot. Therefore, narrative generation has potential applications in game development as demonstrated by the game-like interactive drama prototype Façade [158]; game AI methods for storytelling will be covered in Section 2.3. Stories in games, however, often transcend the "traditional" media such as dialog or text: stories can be implicitly communicated via the gameworld and its visuals or soundtrack. Moreover, the actions of a player's avatar can be retold in the form of narrative from the part of the player. Having the players do, feel and retell what the game design intends them to — without destroying their perception of player freedom — is a challenging task for game design, although several methods for indirect control have been laid out [204, Chapter 16].

Ludus: While games have the previous facets in common with other media, there are also those that are unique to games. The term *Ludus*, established by Caillois (1961) and elaborated by Frasca (1999), refers to an "activity organized under a system of rules that defines a victory or a defeat, a gain or a loss." The uniqueness of the *ludic facet* stems from the fact that rules define the limits of player freedom and pose as player goals; this allows room for creativity in defining the limits and goals of player interaction.

A game's play experience is primarily defined by the game's rules. Rules provide the structures and frames for play (e.g. winning and losing conditions), as well as the game's mechanics, i.e. the actions available to the player. In commercial games, rules are carefully crafted by human game designers. More often than not, such rules follow the standards of the game's genre which constrains the creativity of the designers. While often a sequel to an established

series has minor rule changes from its predecessors, there have been cases where a minor tweak in the rules has caused the literal transformation of a genre. An exemplar of this is a fan-made modification of the strategy game *Warcraft III* (Blizzard 2002) which removed base building and most unit control, allowing the user to control a single 'hero' unit; the resounding success of these tweaks has since given rise to a new, popular game genre: Multiplayer Online Battle Arenas.

While the ludic facet during game design focuses on the definition of mechanics and rules (neatly arranged in the design document and in a format that can be easily implemented in code), the play experience that stems from these rules is much less predictable due to the player's interpretation of these mechanics in the greater context of the game and its other facets. The game's play experience therefore does not stem from each individual mechanic, but from their interaction with each other and with the game's other facets, as argued by Hunicke et al.: "From the designer's perspective, the mechanics give rise to dynamic system behavior, which in turn leads to particular aesthetic experiences. From the player's perspective, aesthetics set the tone, which is born out in observable dynamics and eventually, operable mechanics." [112]

Level Architecture: Most games are built upon the spatial navigation of levels which determine how the player agent can progress from one point in the game to another. Some examples of levels include the two-dimensional arrangement of platforms and coins in *Super Mario Bros* (Nintendo 1985), the three-dimensional arrangement of houses, trenches and enemies in the World War 2 shooter *Call of Duty* (Infinity Ward 2003), the elaborate structures that the player tears down in *Angry Birds* (Rovio 2009), or the expansive open gameworld in *Minecraft* (Mojang 2011). A game's tone is often set by its levels and the challenges they pose; digital games often have a constant or near-constant set of mechanics throughout, but vary the gameplay and challenge through level design.

Like real-world architecture, level design must take into account both visual impact and functional affordances of the artifacts it creates. Depending on the type of game, functional affordances may include a reachable end-goal for platform games such as *Super Mario Bros*, challenging gameplay for driving games such as *Forza Motorsport* (Turn 10 Studios 2005), or good action pacing with breathing room between difficult sections as in *Resident Evil 4* (Capcom 2005). On the other hand, the level's appearance plays a significant role not only for the visual stimulus it provides but also for the purposes of navigation: a sequence of identical rooms can easily make the player disoriented as was intended in the dream sequences of *Max Payne* (Remedy 2001), while dark sections can add to the challenge level of the ludic elements due to low visibility as well as psychological anxiety as is the case of *Amnesia: The Dark Descent* (Frictional Games 2010). The design of larger, open levels or gameworlds borrows less from architecture and more from city planning [149], with edges to constrain player freedom, districts to break the level's monotony and landmarks to orient the player and motivate exploration.

2.2 Computer-Aided and Mixed-Initiative Design

Since its early stages of development, the computer was expected to assist in solving engineering problems by being involved in the creative design process and by automating tedious tasks.

However, computers were usually expected to be mere servitors while their human "operator supplied the initiative, the direction, the integration, and the criterion" [144]. Licklider envisioned, in 1960, that human and computer could have a more symbiotic relationship where "the contributions of human operators and equipment will blend together so completely in many operations that it will be difficult to separate them neatly in analysis" [144]. During the same period, Sutherland was finalizing the first Computer-Aided Design software, Sketchpad [237]. Sketchpad allowed the user to interact with the computer not via magnetic tape reels or card punches but by drawing directly on the computer monitor using a light pen. This nascent graphical user interface spurred interest in developing and using computer-aided design tools. *Computer-aided design* (CAD) tools are software programs that assist engineers and designers to create or change a design and are used in a multitude of domains such as engineering, architecture, gadgets, vehicles and computer games. Even long after the position of Licklider was published, CAD tools have often been identified by their dual role as "the designer's slave" [194] — performing simulations, analysis and constraint satisfaction tests — and as advisors when certain requirements are not met. As computers are becoming efficient at performing the former role, more researchers focus on the latter: the most ambitious role for the computer is that of the colleague, although other useful roles include those of nanny, penpal or coach [148]. According to Lubart, as a colleague a computer should contribute equally to the design discourse but could also incite creativity via "semi-random search mechanisms to generate novel, unconventional ideas" [148].

While Licklider mentioned "initiative" as early as 1960 [144], the term *mixed-initiative* to describe computer-aided design tools has yet to be properly defined [176]. For the purposes of this thesis, mixed-initiative design (or co-creation) considers that both the human and the computer *proactively* make contributions to the problem solution, although the two initiatives do not need to contribute to the same degree. This differentiates mixed-initiative co-creation from the collaboration of humans in a group or the use of non-proactive software (e.g. spell-checkers) or non-computer support tools (e.g. artboards). Often, initiative is discussed in the context of dialog [257], even going as far as to use an imaginary spoken-dialog computational system. In that context, Novick and Sutton ascribe initiative in terms of (a) the choice of task, i.e. what problem needs to be solved, (b) the choice of speaker, i.e. who talks, when, and whether they can be interrupted, and (c) the choice of outcome, i.e. which is the desirable solution to the problem [176].

Computer-Aided and Mixed-Initiative Design in Games

In commercial games and game-like applications, computer-aided design speeds up the development process in the form of game editors. Game editors use an intuitive graphic interface, allowing designers with little programming experience to script behaviors and create content, usually as part of a game level. Many of these tools ship with the final game, allowing end-users to generate content which increases replayability and fan loyalty. *Modding* via the provided game editors has often transcended the original game's concept and mechanics, leading to new game titles such as *CounterStrike* (Valve 2000) or new subgenres such as Multiplayer Online Battle Arenas. Over the years, game editors have become very sophisticated, driven by a desire to support the modding community or to reuse code across products. As an example, the *Unreal Development Kit* supports landscape sculpting, asset organization, scaled render-

ing accuracy and code-free visual scripting. On the other hand, game-specific editors such as the Creation Kit of *Skyrim* (Bethesda Softworks 2011) allow less customization but offer game-tailored easy-to-use automations such as leveled item lists, navigation path generation and quest scripting.

Commercial games focus more on computer-aided design tools, and assume that the prospective creators have enough ingenuity, drive and experience with digital games to see an idea from its conception to its completion. Research in mixed-initiative tools for games follows a different agenda. Such tools attempt to impart some expert knowledge regarding level or game design to end-users, making it more accessible to untrained novices to express themselves through games while keeping their experiences playable (and to a degree, entertaining). Moreover, such tools aim to foster the creativity of the designers by providing computer-generated inspirational artifacts (or pieces thereof). Apart from Sentient Sketchbook and other tools presented in more detail in this thesis (e.g. in Chapter 10), a handful of mixed-initiative tools for level design will be presented in some detail in the following paragraphs.

Tanagra [223] allows human-computer collaboration in the creation of a 2D platformer, by taking control (computationally) of the more menial aspects of level design from the human user. The generator ensures that the levels it creates are playable via A Behavior Language (ABL), a reactive planner [157] which satisfies all constraints of playability. Since playable 2D platformers need maximum limits on the height of platforms and the width of gaps, by automating this rather time-consuming and menial process the human user can operate on a higher-level, e.g. evaluating if the generated level is entertaining or exploring the space of generated levels. The human designer can at any time "pin" parts of the geometry, forcing the generator to keep the height and width of pinned elements when it generates new levels. Finally, the generator also accounts for the "rhythm", where each beat in the rhythm corresponds to a single action taken by the player; the generator or the human can adjust the frequency of beats, thus affecting the difficulty and required actions to complete the level, as well as any rhythm highs and lows within the same level.

Ropossum [211] allows for the authoring and testing of levels for the physics puzzle game *Cut-The-Rope* (ZeptoLab 2010). Ropossum allows for a fully human-controlled authoring environment, but can also operate on partial human designs but adding (or modifying) the necessary elements for the level to be playable. Playable in this case amounts to at least one sequence of user actions (such as cutting ropes, bursting bubbles, firing rockets etc.) getting the candy to reach the Omnom creature. Testing whether the level is playable is done by encoding the level's elements as facts, which are sent to an AI reasoning agent [210]. The reasoning agent infers the best action to perform in that state, and sends it to the physics simulator which updates the game state accordingly. Performing enough iterations of this state update process, the agent builds a state tree in which the agent can backtrack and try other actions in earlier states. Actual generation of Cut-The-Rope levels is achieved through grammatical Evolution (GE), which uses a Design Grammar describing the level structure, and position and properties of the level components.

Sketchaworld [217] allows for the iterative modeling of virtual worlds, where a human user can adjust the height of the terrain (adding mountains and valleys) and place cities, roads and rivers in it. Perhaps the primary role of the computer in Sketchaworld is to convert the user's specifications into elaborate, photorealistic and interactive terrain; however, several other modules are in place to enhance the computational initiative. An adaptation step makes

the necessary local changes in the world when a user adds a new feature, in order to ensure smoothness and correctness; as an example, if a user sketches a river which passes through a previously added city, the city is split in half along the river banks and the neighborhoods are restructured to allow for road access to all parts of the city. This adaptation step acts as a constraint checker as well, reporting back to the user when their additions cause non-smooth or incorrect maps that the adaptation module can't fix.

2.3 Game AI

Artificial Intelligence (AI) in games arguably — and depending on the definition of AI — dates back to the earliest forms of digital games in the late 1970s. However, Game AI as a viable and independent research field likely traces its origins to the seminal paper by Laird and van Lent which convincingly argued that games are ideal application domains for the (long-term) goal of human-level AI as they provide elaborate, visually appealing simulations which allow AI research to "concentrate on cognitive capabilities and finesse [without] many of the pesky issues of using real sensor and real motor systems" [127]. Early research in game AI revolved around non-player character (NPC) control, focusing on planning [179, 180, 97], pathfinding [235, 20, 34], or locomotion [114, 107]. It is unclear whether this focus was due to the robotics background of many game AI researchers, due to the research problems and areas laid out by Laird and van Lent in their paper, or due to the push of contemporary games during that era for more realistic and believable adversarial AI. Notable examples of sophisticated NPC control in games are *Black & White* (Lionhead 2001) where the creature-avatar learns from player actions as well as via its own belief-desire-intention model, *Thief* (Looking Glass Studios 2008) where the guards exhibit an elaborate sensory model of light and sound and *The Sims* (Maxis 2000) where all avatars have elaborate interactions with each other and with household objects according to their desire models. The game industry has somewhat settled, in later years, into a small set of computationally efficient and easy-to-implement algorithms for most "traditional" NPC control tasks such as pathfinding or aiming in First Person Shooter Games. However, adversarial control in games with more complex game states such as real-time strategy games has a long history of academic and industrial interest [29, 70, 53]; strategic gameplay proves to be a challenging problem to this day [260, 259, 38] and attracts attention through multiple competitions of the *Starcraft* (Blizzard 1998) video game[1]. Outside the immediate interests of the game industry, on the other hand, tasks such as generic game playing (i.e. agents able to perform well in dissimilar or unseen games) via machine learning has gathered considerable academic attention [82].

As NPC control in commercial games has reached a level of sophistication and robustness that satisfies most of the industry's needs (or, arguably, since games have shifted from single-player to multiplayer without adversarial AI), both research and commercial interest in AI has moved to other domains [171, 268]. Among non-traditional uses of AI, some of the most promising (and popular) directions include interactive/dynamic narrative, data mining, procedural/automated content generation, player experience modeling and AI-assisted game design. Since procedural content generation and player modeling will be covered in more detail in Sec-

[1] In 2013, Starcraft competitions were held in the Computational Intelligence and Games conference and in the AIIDE conference.

tions 2.4 and 2.5 respectively, and AI-assisted game design is the focus of the entire thesis, the following paragraphs will cover work in narrative intelligence and data mining.

Narrative Intelligence

Although computational narrative has a long history [163, 63] within the larger domain of artificial intelligence, it has found in digital games a fruitful application domain for the interactive, engaging experience that they can afford. Narrative intelligence [21] is a highly multidisciplinary field as it finds itself drawing inspiration from longstanding domains such as art, psychology, cultural studies, literary studies and drama [156]. In commercial games, however, traditional narrative structure is rarely possible since the exposition of pre-written plots clashes with player freedom. Unlike traditional stories (including computationally created ones), therefore, the highly interactive nature of games necessitates the use of *interactive storytelling* [50]. Like more traditional forms of narrative generation, the design of interactive storytelling relies heavily on a large database of world knowledge — both textual and logical. Games acclaimed for their narrative, such as *Heavy Rain* (Quantic Dream 2010) and *Mass Effect* (Bioware 2007), include thousands of lines of dialog authored by multiple game writers. While game-like interactive narratives such as Façade [158] or Prom Week [161] similarly include a large number of pre-written dialogs, the computer is much more proactive and selects a fitting response of a virtual character based on the context of the current discussion, the player's assumed knowledge and the future intended outcome. Since many applications of interactive narrative use planning, they usually operate with a specific goal in mind, i.e. have an embedded aim to tell a specific kind of story, with a specific structure and/or outcome — for instance, Façade wants the game to tell a story of a couple with marriage problems. Although multiple goals can of course be included in the system, the novelty of the story's conclusion is often not exceptional but the events leading to this conclusion may well be.

However, the burden of imparting world knowledge to an interactive narrative system can be somewhat alleviated by directly using real world data to inform the creation process. Human-based computation can use previous user interactions, current world events or online encyclopedias in order to detect items of interest or logical connections between story elements. For instance, Orkin and Roy use a lexicon of actions and utterances from data of over 5000 players in a simple restaurant game to train virtual agents' verbal responses based on N-grams [181]; of note is the evaluation of this machine learning method which required an audience of human judges to rate the agents' behavior in terms of typicality, i.e. whether they were likely to be heard in a restaurant. Swanson and Gordon created a co-operative storytelling system where human and computer take turns adding sentences to an emerging story [239]; the computer analyzes the current story, matches it to a database of over a million stories from web blogs and uses the corresponding next sentences from the closest matching story. Cook et al. used current news items from an online news site as well as wikipedia images of their protagonists (tailored to the story's mood) in order to implicitly tell a story in the background of a platformer game [45]. Finally, the artificial improv actors of Fuller and Magerko draw inspiration from human interactions and alter their own cognitive models to match the perceptions and proto-narratives that the human improv actor wishes to act out [73].

Data mining

Data mining in games is one of the newest applications of game AI, emerging from the development of massive-multiplayer games such as *World of Warcraft* (Blizzard 2003) as well as from inclusive online distribution and community platforms such as *Steam* (Valve 2003) or *Xbox Live* (Microsoft 2002). The ever-increasing number of players in online games and the ever-increasing volume of data collected regarding their play behaviors — and purchasing patterns — has incentivized game companies to parse that data in order to find patterns which would allow them to make more engaging, more successful, and ultimately more profitable products in the future. Data mining can be used to provide meaningful visualizations of players' gameplay behavior in a level [58] or to study the data through game design lenses such as play-personas [249]. Most often, data mining is used for data-driven player modeling, in order to find correlations between demographic data and gameplay [243], to cluster players based on common behavioral trends [60] or to predict user retention from in-game performance [153]. While player modeling will be discussed in more detail in Section 2.5, player modeling via data mining makes use of massive databases of player information such as 9367 players of *Battlefield 3* (Electronic Arts 2011) [243], 220000 players of *Forza Motorsport 4* (Turn 10 2011) [146], or 6430 players of *Tomb Raider: Underworld* (Crystal Dynamics 2008) [153].

2.4 Procedural Content Generation

The game industry has been using algorithmically generated content since the 1980s with games such as *Rogue* (Toy and Wichman 1980) and *Elite* (Acornsoft 1984). The processing power of the computer was harnessed, during those early years, to generate elaborate levels or the entire gameworld based on a small set of parameters (such as a save-file) or based on a few random numbers. The compression potential of such algorithms is still harnessed today in several ways, such as storing complete game states in save files or transferring compact data in online or multiplayer games. However, since the processing power or storage space in computers has become less of an issue in modern days — and arguably due to compression being more related to engineering and computer science than to the games themselves — the term *procedural content generation* usually encompasses game content which somehow varies from one playthrough of the game to another.

Following the typology of Togelius et al. [246], algorithms that generate game content can be broadly classified into constructive, generate-and-test and search-based. Each category will be described, along with examples, in the following subsections.

Constructive Algorithms

Constructive algorithms are agnostic to the artifacts they generate, in the sense that they do not validate the quality of the generated content before presenting it to the user (or player or designer). The game industry traditionally uses constructive methods since the lack of a quality control module saves processing power and allows for faster content generation in real-time, e.g. creating treasure when a monster is killed in *Diablo* (Blizzard 1998). Often such constructive algorithms are carefully tailored to generate content which is both playable

and "sensible", i.e. that resulting artifacts follow the same patterns in multiple runs of the algorithm. An example of such finetuned algorithms is the world generation of *Civilization V* (Firaxis 2010) which has specific descriptions for its maps (e.g. "inland sea"); since the world is generated once at the start of the game (while a loading screen is being displayed), the extra computations of the elaborate script do not break immersion. A more lightweight approach in games is to generate content based on a random number against a designer-authored list of possible outcomes (with their respective probabilities). Such probability tables may be nested, e.g. with a roll on one table needing further elaboration by rolling on a second and possibly third table, or conditional, e.g. with certain tables being rolled at only when the player character is in a specific level-range or region. An example of such random tables can be found in *Diablo*, where treasure dropped by monsters is chosen based on a random roll on different tables determined by the character's level; in the case a weapon is rolled, it is further elaborated by rolling a weapon type (also determining its in-game appearance) and an attribute which affects the weapon's stats, thus producing a "Short Staff of the Fox".

Constructive algorithms were the first method, historically, for generating content within the game industry; this however does not mean that they can not be sophisticated and worthy of academic attention. Constructive algorithms can integrate constraints which ensure the playability of a generated game or a generated level, or that some designer criteria are met. Such algorithms use declarative programming such as Answer Set Programming [81]. Whether such constraint programming generative methods should be classified as constructive, generate-and-test or search-based is debatable; this thesis considers such methods constructive since the final generated artifact does not need to be tested for constraint satisfaction. Constrained programming has been used successfully to generate complete arcade-style games [220], levels for puzzle games [219], platformer levels [223], mazes [173], dungeons [108] and strategy game levels [221]. Constructive methods have also been used for generating levels for *Super Mario Bros* (Nintendo 1985) by biasing the random chance of a level chunk being selected based on its novelty as well as designer criteria [159]. Even more data-driven approaches have been applied for constructively generating *Super Mario Bros* levels based on patterns learned from a corpus of human authored levels of the original game [225].

Generate-and-Test Algorithms

Unlike constructive methods, generate-and-test algorithms validate the generated artifact's quality before presenting it to the user. If the artifact fails to satisfy certain constraints (such as playability, game balance or even engine-specific constraints, e.g. can not be rendered) then the algorithm is re-run until a valid artifact is generated. Defining constraints — or evaluating quality as a scalar value and determining a minimum threshold for "acceptable" quality — is the primary focus of generate-and-test algorithms which differentiates them from constructive methods. The design decisions taken when choosing constraints largely determine the usefulness of the generate-and-test algorithm: if the generative algorithms often create (by random chance) a valid artifact that satisfies the constraints, then the algorithm may need only few retries (or none at all) before it shows valid content to the player. On the other hand, if the generative algorithms have a very broad expressive range and can create content of varying quality — or if the constraints are very specific — then the generate-and-test algorithm can potentially run forever without stumbling (by random chance) on a valid artifact.

Search-based Algorithms

Search-based algorithms for generating game content have an evaluation module similar to the generate-and-test methods, but use the evaluations to select the most promising artifacts and guide the generation of future artifacts based on those. The most common method for search-based procedural content generation [246] is through artificial evolution which abstracts Darwin's notion of survival of the fittest with fitness determined from the algorithmic evaluations of game quality. The details of artificial evolution and genetic algorithms are provided in Section 3.2. Evaluating the quality of game content is more elaborate (and challenging) than that of generate-and-test methods, as the fitness function should provide a smooth "gradient" to guide the (usually stochastic) search of these methods. As an example, an average-looking artifact would need to score higher than a worse artifact for search-based methods while for generate-and-test they could both be equally ranked if they both failed the quality constraints imposed.

Although the method of search is a primary concern of search-based approaches, an equally important factor is the evaluation of generated game content (since it also affects which method of search is most appropriate). Game content can be evaluated based on a mathematically defined quality formulation inspired by game design theory, expert knowledge, or a clear vision of the intended outcomes. Such mathematical formulations can evaluate the similarity with an archetypical "example" [3], a popular design pattern [141], or an intended visual impact [137, 109]. The evaluations can also be derived from simulations of artificial agents interacting with the content, such as rule-based controllers traversing a generated level [185], agents controlling generated spaceships [136], or two adversaries playing a board game based on simple rules [28]. The evaluations can also be tailored to the players interacting with the generated content. This can be achieved via the user's explicit choices with interactive evolution as described in Section 3.2.3; these choices can be direct, by having players select which individual breeds in the next population [195, 33] or more obfuscated as in *Galactic Arms Race* [93] where preference for a weapon is derived from the number of times it is used. Another way of estimating user preference on unseen content can be achieved via predictive computational models which learn from previous interactions with this user or a larger pool of users; these methods of experience-driven procedural content generation [271] will be elaborated in Section 2.5.

2.5 Player, User and Designer Modeling

Player modeling is "the study of computational models of players in games" [270]. Player modeling in its various expressions facilitates at least one of four purposes: the description, prediction, interpretation, and in some cases reproduction of player behavior. At its most basic, a player model should take some form of input regarding the player and (usually) produce some form of output which can be used to describe the player. Input can take the form of interaction data from their in-game behavior, their physiological responses while playing the game, general information about the game (e.g. difficulty setting, played level properties) and general information about the player (e.g. demographics, age, gender, etc.). Example inputs to a player model include level information (such as the number of gaps in a platformer) along with gameplay data (such as number of deaths) [184], or posture information

during gameplay [212]. The output of the player model, on the other hand, can vary depending on the model's purpose: examples include reported player affective states (e.g. challenge or frustration), third-person annotations of player states, predicted player behavior in unseen circumstances (e.g. in a new game level) or classifications of the player (e.g. if the player is an achiever or an explorer). The way the model is constructed, finally, falls into two categories: top-down approaches, which are based on a theoretical framework [124, 72, 240, 68] (earning them the name model-based approaches) and follow the framework's hypotheses on deriving the output from the input. Experiments in top-down player models usually serve to determine to what extent, if any, the hypotheses of the theory fit the observations. On the other hand, player modeling can be achieved via a bottom-up (model-free) approach, where the data are used directly, without prior assumptions, to derive the correlations between different input dimensions. This can be achieved via data mining and in particular regression (in cases where both input and output is known), preference learning (in cases where players report their preference of one playthrough or type of content over one or more others) and classification which can be used even in cases where there is no known output. An example of model-free player modeling via regression is the use of a large pool of gameplay data from the first level of *Tomb Raider: Underworld* (Crystal Dynamics 2008) to predict the level in which players quit the game [153]. An example of model-free player modeling via clustering is the use of self-organizing maps to cluster players of *Tomb Raider: Underworld* (represented as six metrics derived from an entire playthrough of the game) into what turned out to be four player types [59].

User models, on the other hand, are larger concepts which relate to human-computer interaction. According to Benyon and Murray, a user model is "a representation of the knowledge and preferences which the system 'believes' that a user (which may be an individual, a group of people or non-human agents) possesses" [14]. Similar to the way a player model often aims to recognize the players' engagement with the game via their in-game or out-of-game behavior, so does the user model often aim to recognize users' literacy and frustration via their interactions with the system. The user model can be informed implicitly by the user's mouse clicks (and other interaction data), via tests (likely with a subset of the general userbase) or from classifying users to pre-existing user 'stereotypes'. A user model is particularly useful for adapting an interface to suit its user's needs. In such a way, the burden of customizing the interface manually is alleviated but a number of other potential problems arise. Adaptive systems must retain their predictability and consistency, i.e. the system should not autonomously or "silently" change the interface as that behavior would be confusing. Explaining the changes made to the system (and their perceived cause) to the end-user may alleviate this problem somewhat, although co-operative adaptation (with the user's consent) may be equally or more desirable. Benyon and Murray argue that "the pertinent problem is one of intervention at an appropriate point, identifying the natural and suitable breaks in an interaction at which adaptation can be effective and efficient with respect to the tracking of user task needs and the interpretation of user actions." It should be noted that a user model does not exist in a void, and is combined with a *domain model* and an *interaction model* [14]. A domain model consists of abstractions of the system which allow inferences to be made from user-system interactions, and includes aspects of the application which can be adapted — thus making it application-specific. An interaction model defines the intended and the actual interaction between the user and the system, and thus captures raw interaction data and can evaluate them in comparison to a "stereotyped" interaction. A large set of methods for processing

human-computer interaction data follows the paradigm of Goals, Operators, Methods, and Selection rules (GOMS) [117]. The GOMS model breaks down the interaction into its elementary actions (physical, cognitive or perceptual) in order to study them based on Goals, i.e. what the player has to accomplish (potentially fragmented into subgoals), Operators, i.e. perceptual, cognitive or motor actions performed in service of a goal, Methods, i.e. sequences of operators which accomplish a goal or a subgoal, and Selection rules, i.e. the implicit rules according to which users select one method over others for accomplishing their goal. Different techniques can be used for a GOMS analysis and may incorporate varying assumptions of human cognition while at the same time being designed to predict execution time, or learning time.

Designer Modeling

As part of the thesis work, the concept of designer modeling was introduced in [140] in an attempt to appropriate many of the general concepts of user modeling and apply them directly to the interaction between a designer and an authoring tool during the creative process. That paper introduced the term *designer modeling* as the mechanism which identifies the style, preference, intentions and affective states of designers and artists in a similar fashion that player modeling attempts to identify behavioral, cognitive and affective patterns of a player. However, designer modeling is a far more complex and challenging task than player modeling and can be viewed as a *second-order player modeling* process since a successful designer model incorporates the intentions of the designer to satisfy the player. In a sense, a designer model evaluates the designer's evaluation of player experience and playstyle.

As the interaction between a designer, the CAD software and the designed object is multi-faceted, there are many aspects of the process that could be modeled — often at different levels. Aspects of the designer's preferences, style, goals, process, affective states and even role within a group can be modeled; the following paragraphs analyze how such aspects could be modeled in a meaningful way.

Modeling Preferences: As any human user, so do designers have their own preferences on the types of content they wish to create. Such preferences may be personal, such as visual taste or favored playstyle, or they can depend on the game's setting, storyline or intended player experience: one game may require a frantic pace, while another may require vibrant visuals. Even within the same game, designer preference could pertain to the design patterns of a specific level, to the appearance of an alien race, or to the challenge level of a specific puzzle. A designer model could identify all of these preferences, regardless of how specific they are; once such preferences are identified, new content which aligns to these preferences can be generated. How such a design model is constructed depends on whether preferences are pertinent to the task at hand or a matter of personal taste; in the former case the model can be trained online, adapting solely to the user's current interactions, while in the latter case the model can be trained offline based on a large body of past interactions. This aspect of designer modeling shares the same computational challenges as *preference learning* [75]. Beyond machine learning, modeling designer preferences for CAD tools also needs to address challenges pertaining to human-computer interaction, since intrusive tools which use learned preferences in a way that limits a user's creative control can have adverse effects [139].

Modeling Style: Many designers have a particular style which can be easily identified across different levels of the same game or even across different games. Telltale signs include visual details such as rounded edges or particular color schemes as well as more abstract features such the presence or absence of open spaces, rhythm and pace, player choice, explicit or implicit reward mechanisms etc. As an example, consider the classic level designs of Shigeru Miyamoto in several early Nintendo games such as *Super Mario Bros* (Nintendo 1985) and *The Legend of Zelda* (Nintendo 1986), where the designer's preference for hidden features and quirky passages that reward replay and exploration can be seen across dissimilar games. Such designer style can be modeled as a form of preference in the sense that, when a design problem occurs, the designer's style is implemented as their preference for one among several possible solutions to that problem; faced with the problem of how a player can progress through a door, Miyamoto might prefer hiding a key behind a crack in the wall whereas another designer might prefer putting a strong enemy in front of the door. Since style is ultimately a form of preference, the same computational challenges of modeling designer preferences apply; however, since style is more pervasive across games or genres, larger datasets may be necessary to discern styles, moving the modeling of a designer's style closer to the larger problems of data mining, clustering and pattern recognition.

Modeling Goals and Intentions: A CAD tool is expected to help the designers achieve their goals. While *goal recognition* via machine learning has been identified from early on [87] as a significant enhancement to user/system interactions, there has been limited interest in incorporating it in tools for game development. A prediction of a designer's current goals can be made from past interactions; assuming that the designer has a consistent style, the system can predict the designer's next steps by matching the user's current creative step with previously seen interactions. The end result of such previous interactions can be shown, as advice, in order to speed up the creative process. Alternatively, the system can suggest likely next steps based on a probabilistic model of cognitive associations [258], which is trained on past design sessions. There are however several challenges in the prediction of goals: for instance, the system is likely to present no new content, as it uses past artifacts more or less directly. This is as likely to stifle creativity and enhance designer fixation as it is to speed up the creative process. Additionally, the goals of the designer are rarely known in advance, even to themselves; it is therefore difficult to evaluate if the tool succeeds in predicting or helping achieve the designer's goals.

Modeling Process: Different designers follow different design processes: some level designers start with a rough sketch of the map layout, others start from identifying the intended challenge level while yet others have fleshed out specific portions of the level (i.e. a chokepoint, a sniper location or a secret passage) before worrying about the big picture. A challenging goal of designer modeling, therefore, would be to identify such different processes and integrate them into their support mechanisms. Following the level designer example, a design tool could alternate between generating suggestions which increase the 'resolution' of a rough sketch [143], suggestions which maintain the intended challenge level, and suggestions which add details only to the portions of the level which are not fully fleshed out yet. The design process of the human and the computational designer can be even be matched more closely, such as for instance identifying if the level designer is currently placing resources or obstacles. Modeling a designer's process can be viewed as a short-term *plan recognition* problem [122], and thus actions such as resource placement can be clustered together for the purposes of activity

recognition [250].

Modeling Affective States: As emotion plays a critical role in human decision making [51], it is central to the drive of creativity. Arguably, a designer's emotion is far more complex to identify and measure when compared to modeling behavioral patterns of interaction. However, research in affective computing [187] has shown the potential of reliably modeling user affect via emotional manifestations. We argue that similar to modeling player's affect during gameplay interaction through head motion [4] or physiology [155], affective computing methods can be applied to detect designer frustration or surprise which are critical during creative processes. Conceivably, using a designer model that predicts which content will surprise a designer, a computational creator could generate and present surprising suggestions in order to break designer fixation.

Individual versus Collective Designer Modeling: As in player modeling, there is the potential to model both individual designers and groups of designers such as the members of a game studio. A designer modeling tool that captures the preferences, goals and processes of a large community of designers would be useful, as it could learn much more from the abundance of interaction data than it could learn from a single designer. Having a good initial model would also be beneficial to novice designers, who might need more help in the design process. Modeling individual designers on the other hand allows for personalization, especially considering that designers often have highly divergent styles and ways of working, as noted above. Conceivably, an effective designer model would include both collective and individual models, with the former describing a baseline and the latter deviations from that baseline. Supervised machine learning algorithms would likely be useful for collective models based on large amounts of data whereas instance- or case-based models could be used for individual designer modeling.

2.6 Computational Creativity and Generative Art

Computational creativity as a multidisciplinary field of study primarily aims to construct algorithms capable of human-level creativity. In order to accomplish that, the field also takes interest in better understanding the psychological causes and effects of creative human behavior. Indeed, defining creativity remains an open question which has caused numerous definitions to be put forth, often revolving around the novelty and intrinsic value of the generated artifacts. Boden categorizes creativity into combinatorial (revolving around the combination of human created atoms), exploratory (revolving around usually iterative search in a large search space) and transformational (revolving around the redefinition of the search space and the breach of constraints) [22]. Evaluating the creativity of computational systems has similarly been a topic of debate in this community, and numerous evaluation frameworks and formalisms have been proposed [197, 264, 152, 44, 119, 27].

Applications of computational creativity often revolve around tasks which, when performed by a human, would be considered "creative" [197]. Such tasks often revolve around artistic expression, including visual art, music, narrative or other forms of linguistic creativity such as humor or poetry. Each of these domains has a large corpus of human-made artifacts as well as extensive studies and theories on those; indicative handbooks of music appreciation [162, 192, 201], visual perception [2, 18, 77] and narrative structure [6, 191, 24] only graze the

surface of the studies in such creative activities which date back to antiquity [1]. Due to this accumulated wisdom, computational creativity in these domains is often paired with domain-specific theories and principles. For instance, music generation often takes advantage of the expectations of a listener who has been "trained" to a specific musical style by using probabilistic models such as Markov-chains trained on an extensive corpus of human compositions [66, 182]. Genetic algorithms which generate sound pieces may use objectives "based on material gleaned informally from many sources (music books, articles) which aim to explain and model the process of improvisation" [265] or human-based evaluations [15, 113, 104]. Linguistic creativity on the other hand may revolve around phonetic or semantic similarities in the case of punning riddles [154, 17, 16], can generate similes or analogies based on world knowledge mined from human web search entries [251, 253, 92], or may use extensive evaluations of story quality to generate complete stories [186, 83] or interactive narrative [172, 84].

Visual art has also often been a focus of computational creativity and algorithmic generation in general. The affordances of modern technology to move art from a framed canvas to a high-resolution screen has also opened the potential for a dynamic and/or interactive display of artworks. Examples among generated art pieces are the works of Edmonds et al. [65] and Bowman [26]. As in other facets of creative activity, visual art generation is often grounded in theory of visual appreciation. Several evolutionary art projects distill studies of cognitive psychology [2], neuroscience [193] and perception [226] into mathematical formulas of visual impression, for instance based on image compression [199, 198] or shape balance [137]. Other methods of evaluating visual quality can be learned from human ratings in a collection of generated content [7], from classifiers between man-made and generated images [151, 48], from filters used in computer vision for detecting common patterns such as faces [47] or by learning associations between images and human-ascribed picture qualities such as "happy" or "eerie" [175]. Finally, generated visual artifacts can also be evolved by humans, either individually [214] or collaboratively [208] as an instance of interactive evolution; using humans directly in such a fashion can lead to a faster and better quality in the algorithmic output but the computational creativity of such methods has been questioned [78].

2.7 Human Creativity

As elaborated in Section 1.1, this thesis explores the potential of a computationally creative design tool not only to assist its human users but also to foster their creativity. This presumed *mixed-initiative co-creativity* should be construed using theories on human creativity. The understanding of human creativity has relied on diverse philosophical (e.g [266]), neuroscientific (e.g. [52]) and psychological (e.g. [234]) perspectives. While mixed-initiative co-creation can potentially be linked to several theories of human creativity, the visual design tasks tackled by tools within this thesis mesh well with principles of diagrammatic cognitive reasoning and aspects of *lateral thinking*.

Lateral Thinking

Mixed-initiative co-creation is aligned with the general principles of *lateral thinking* [55] and *creative emotive reasoning* [203], the latter being an instance and specialization of the former.

Lateral thinking [55] is the process of solving seemingly unsolvable problems or tackling non-trivial tasks through an indirect, non-linear, creative approach. According to De Bono, lateral thinking skills can be taught. The prototypes described in this thesis realize the very nature of lateral thinking which, as a creativity process, is boosted through (increasingly) constrained spaces of solutions [55]. Co-creation with computational creators of visual art and design (e.g. for game level design) encapsulates the very core principles of *diagrammatic reasoning* as human creativity, and especially lateral thinking creativity, is often associated with construction and the principles of customization [55].

The *random stimulus* principle of lateral thinking [10] relies on the introduction of a foreign conceptual element with the purpose of disrupting preconceived notions and habitual patterns of thought, by forcing the user to integrate and/or exploit the foreign element in the creation of an idea or the production of a solution. Randomness within lateral thinking is the main guarantor of foreignness and hence of stimulation of creativity [10]. According to creative emotive reasoning — which enriches the basic notions of lateral thinking with semantic, diagrammatic and emotive dimensions — the creative act is understood as an intervention that results in *re-framing*; frames can be viewed as systems or established routes which divide the possibility space (e.g. the game design space) into bounded, meaning-bearing sub-areas. On that basis, the random stimulus and the re-framing principles have one element in common: they are enablers of a change in the lateral path. The *re-framing* and the *random stimulus* principles are embedded in this thesis' design tools as machine creativity offers heuristically-driven stimuli that are often altered through e.g. mutations within a genetic algorithm; that can, in turn, alter the user's framing on a particular task/problem. An artificial mutation to a visual diagram, an image, or a game map, resembles the random stimulus that can act as a potentiator of creativity and cause an alteration of lateral thinking.

Diagrammatic Reasoning

Diagrammatic reasoning can be defined as reasoning via the use of visual representations; a cognitive process which is enabled during game level design, interaction design and visual art. These representations can include all forms of imagery incorporating visual features (object shape, size, color, spatial orientation etc.) [36]. Literature suggests that complex information processing is benefited by the use of diagrams, due e.g. to the fact that information in diagrams is indexed by spatial location, thus preserving explicitly the geometric and topological relations of the problem's elements (see e.g. [128]). Diagrammatic reasoning is premised on the background knowledge of the relevant domain, as well as the specific nature of the diagram and its interconnections with the context within which one encounters it [36].

Diagrammatic Lateral Thinking fuses the principles of diagrammatic reasoning and lateral thinking. Diagrammatic lateral thinking builds upon the *extended mind* theory [39] and its core idea is that a diagram, through its use, serves as a vehicle of cognitive processes, embodying the various aspects of the problem. The user's (e.g. designer's) mind is extended onto the diagram and reasoning proceeds through structural (rather than semantic or syntactical) entailment. One therefore *thinks* through the diagram rather than its use as a simple image. According to diagrammatic lateral thinking, the process of constructing a diagram (an image, a map, or a character) is more important that the final product [254]. Moreover, the possibilities one sees for constructing, altering or transforming a given diagram are part of one's comprehension of

the diagram itself; the functions of the diagram both on the semantic and pragmatic level are determined in part by these possibilities [216].

The prototypes described in this thesis can not only be viewed as being closely related to lateral thinking but, furthermore, they often constitute a type of diagrammatic lateral thinking. Mixed-initiative co-creation occurring through diagrammatic representations (e.g. in level design) offers visual (diagrammatic) alternative paths that satisfy a number of conditions. These define non-linear *lateral paths* within the creative (possibility) space as they promote deep exploration of the space of possibilities which is, in turn, a core lateral thinking characteristic. Diagrammatic lateral thinking, as MI-CC, does not necessarily embed transformational creativity processes as identified by [23]. MI-CC expands the very notion of diagrammatic lateral thinking as it dichotomizes diagrammatic lateral thinking into two main creativity dimensions: one that is based on *analogical* thinking from diagrams and images and one that works purely on the *visual* level through imagistic lateral thinking pathways [203]. Game design tasks, such as the creation of game levels, rely on both visual and analogical reasoning, since the game visuals have aesthetic value but also come with gameplay affordances and constraints.

2.8 Summary

This Chapter provided a brief overview of the domains directly or indirectly connected to the goal of mixed-initiative game design automata. The task of game design and its subtasks was thus briefly presented. While any tool for this task falls under the domain of Computer-Aided Design, this thesis argues for the need of artificial designers which not merely support the human designer but are autonomously creative. For that reason, the domains of procedural game content generation, which revolves around the algorithmic creation of game content, and that of computational creativity, which revolves around the actual creativity (human-like or not) of such generative algorithms, are covered as well. As this thesis also regards recognition and accommodation of the designer's goals, process and preferences to be an important feature of a mixed-initiative tool, the domains of user modeling — and the more game-focused domain of player modeling — were also included. A survey of the history of artificial intelligence in games provided some insight on how that domain's goals have shifted over the years from in-game competitiveness of game playing agents to believable behavior to, at this time of writing, other topics such as player modeling, content generation, data mining and interactive narrative. The writeup of this Chapter focused more on the main goals and research questions that each domain sets out to answer, and less on details of the algorithms and techniques used to achieve these goals. The next Chapter will present algorithmic problems which the algorithms described in this thesis wish to address, along with a more thorough writeup of existing algorithms for performing these tasks. Unlike the current Chapter, Chapter 3 does not focus on a specific domain such as procedural game content generation, as the algorithms can and have been applied to many different problems such as industrial design, evolutionary art, agent control and many more.

Chapter 3

Related AI Methods

While Chapter 2 provided context regarding research areas which aim to answer similar questions as this thesis, this Chapter will provide context for the algorithms used within the mixed-initiative tools of this thesis. One of the arguments of this thesis is the need for computational designers to acknowledge and adapt to their human collaborators. In order to achieve that, some form of machine learning must take place; Section 3.1 will give a brief overview of this domain and specific instances thereof which will be covered in this thesis. Moreover, this thesis argues that mixed-initiative tools need computational creators which can autonomously generate content via search-based methods (see Section 2.4); Section 3.2 describes different algorithmical approaches to evolutionary search.

3.1 Machine Learning

Machine learning is a broad term which encompasses computer software which is able to learn — and thus improve its performance according to some measure — based on experience and without being explicitly programmed (paraphrasing the formal definition of Mitchell [168]). Machine learning can be performed in a supervised fashion (using labeled training examples), in an unsupervised fashion (using unlabeled training data) or via reinforcement (using rewards from the environment embedded in the training data). A vast literature of machine learning approaches has been accumulated including reinforcement learning techniques [238, 121], support-vector networks [25, 49], Bayesian networks [183] and many more. Since certain algorithms within this thesis use artificial neural networks in supervised tasks, the following paragraphs will focus on those particular instances of machine learning.

Inspired by neurobiology and the structure of the brain, an artificial neural network (ANN) is fundamentally "a densely interconnected set of simple units" [168, Page 82]. These units are named neurons, and can take multiple real-valued inputs and return a single real-valued output. In most cases, the output is the result of an activation function applied on the weighted sum of the inputs; popular activation functions include the threshold function, the sigmoid function or the linear function. An ANN is a set of neurons, and the network has a set of input neurons (where external variables are fed to the network) and a set of output neurons (which consist of deductions from the external variables). Popular examples of ANN use include pattern detection tasks, where the inputs are e.g. the pixel values of an image and outputs can be e.g. the similarity of the image to a letter, and robot control, where the inputs may be (in a concrete example) the pixels of the road ahead of a vehicle and the outputs are a representation of the vehicle's steering direction [189]. ANNs can be feed-forward, where the information flows from the input to the output, sequentially passing through layers of hidden

neurons, or they can be recurrent, in which case information can flow backward, acting as internal variables for a sense of "past state" [35]. Feed-forward ANNs can be fully connected, where all neurons of the previous layer are connected to the neurons of the next layer (and only to those), while both feed-forward and recurring ANNs can be sparse, i.e. where some pairs of neurons are disconnected.

Training an artificial neural network can be accomplished in several ways, including evolution as will be shown in Section 3.2.4. However, the most popular and "surprisingly successful in many practical problems" [168, Page 81] algorithm for training a neural network is *backpropagation*. Backpropagation is, in short, the update of the weights of each neuron of an ANN in order to minimize that neuron's contribution to the ANN's performance error. Backpropagation is applied on supervised tasks, where the performance error is calculated from the sum of squares of differences between the intended output values and the real output values from the output nodes of the ANN. In cases where networks are larger than a single neuron (also identified as *perceptron*), then backpropagation can derive the contribution of each neuron to this compound error; in fact, backpropagation is necessary if an ANN has more than one layer of neurons and those neurons contain nonlinearities. Backpropagation can be applied on neurons with a continuous, differentiable, bounded and monotonic activation function; the sigmoid function is ideal for such a task. Interested readers in the mathematical components of the gradient descent algorithm used to compute the weight update of each neuron are directed to the thorough description of Haykin [95, Page 183-197] and a wealth of online tutorials on its implementation.

Machine learning techniques applied on supervised tasks, however, run the danger of overfitting to the patterns of data they train on. *Overfitting* occurs when an algorithm accurately learns the patterns of known data (with a low error) but can not accurately predict new data. To test whether the algorithm overfits, often the data is divided into training data (used as input and desired output pairs for the algorithm to learn from) and validation data (used for testing the trained network's predictive capability on unseen inputs). When neural networks are trained via backpropagation, overfitting often becomes an issue when ANNs are larger and more complex than necessary and when the ANNs are trained on extraneously long periods (overtraining) [244].

3.1.1 Preference Learning

Machine learning has often been applied to predict real-valued outputs of unseen data; preference learning is an instance of machine learning where the predictions are made regarding "order relation on a collection of objects" [76]. Instead of comparing, therefore, the set of real-valued outputs of an ANN using an object's parameters as input, preference learning directly outputs the order between such objects. A great strength of preference learning is that its training set, used for supervised learning, does not require explicit ratings of objects and can learn directly from their rankings. Such data is likely less noisy especially in tasks such as human self-reports in user survey questionnaires [267]. In such tasks, the human user may rate their assessment as a scalar value, often in the Likert scale [145]; however, learning to predict these values via machine learning on a large pool of users will introduce noise due to each user's personal bias towards certain ends of the scale. On the other hand, using rankings for assessing user experience is especially appropriate when reporting emotional rapport; this

can be achieved via pairwise comparisons ("I find this gameplay experience more entertaining than the previous gameplay experience") or via fully ranking multiple items in an ordered list (e.g. in order of entertainment value).

Preference learning problems can be categorized as object ranking problems, where the goal is to predict the ranking of objects outside of a training set, and label ranking problems, where the goal is to find a general ranking of labels for all instances of a training set [111]; each of these types of problems can be approached via modeling the relation directly or by indirect induction [76]. Interested readers are directed towards the comprehensive work of Fürnkranz and Hüllermeier [75]; this thesis will focus on learning object rankings by inducting them via a utility function. The task of object ranking can therefore be formalized as finding a utility function $F(x)$ that satisfies:

$$\forall(x_i, x_j), r_{x_i} > r_{x_j} : F(x_i) > F(x_j) \tag{3.1}$$

where x_j is an object and r_{x_j} is its corresponding rank.

Without losing information of the objects' relative ordering, a ranking can be transformed into a set of pairwise relations P such as $\forall x_i, x_j : r_i > r_j \implies (x_i \succ x_j) \in P$, i.e. for all possible combinations of two objects with different ranks, the object with a higher rank is assumed to be preferred over the object with lower rank. A cost function is then defined as:

$$C = \sum_{\forall(x_i \succ x_j) \in P} \max\{0, b - F(x_i) + F(x_j)\} \tag{3.2}$$

where $F(x_i)$ is the expected preference score for the preferred object and $F(x_j)$ is the expected preference score for the non preferred object. This cost function equals zero when the expected preference score for the preferred object is greater than the expected preference score for the non preferred object over a constant threshold b in every pair.

Assuming an artificial neural network with certain characteristics of the object (e.g. pixels of an image, size and weight of a machine component) as inputs and the F utility function as the output, the cost function C can be used as the error for regression methods such as backpropagation (mentioned in Section 3.1). By applying regression on the ANN, the weights are updated as to minimize the ordering discrepancies among all objects being ranked. Rank-based Interactive Evolution in Section 4.3.2 provides a concise example of regression applied to the task of learning user rankings of evolved artifacts.

3.1.2 Deep Learning

While machine learning via artificial neural networks has shown great success in detecting patterns, controlling virtual or physical robots and many other tasks, the architecture of such ANNs was limited to a few layers (usually one or two) beyond the input and output layers. The preference for this "shallow" architecture was primarily due to the poor performance of deeper architectures with numerous layers of hidden neurons [12]. Deeper networks can theoretically represent complex functions which map the input to the output more efficiently (i.e. with fewer nodes) than shallow networks; however, if a deep network is trained via backpropagation, the error signal will decrease exponentially with each layer and thus the weights of neurons

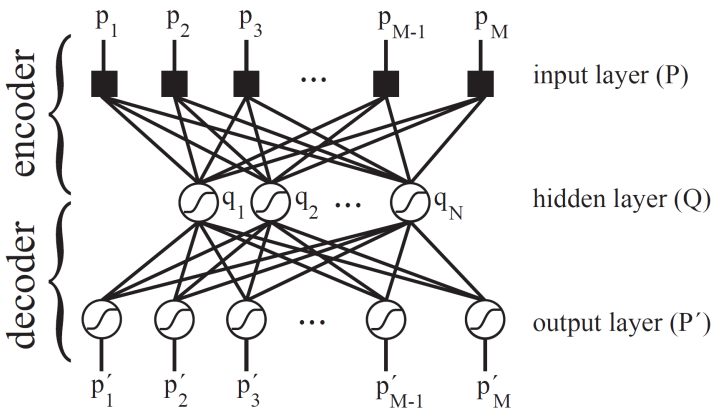


Fig. 3.1: The autoencoder architecture, consisting of the encoder where $Q = f^{\mathbf{w}}(P)$ and the decoder where $P' = g^{\mathbf{w}}(Q)$. The higher-level representation in $q_1, q_2, \ldots, q_N$ contains compressed information which exploits typical patterns in the training set.

closer to the input layer will not sufficiently change. This behavior is more likely to lead a deep network to a local optima or plateau at which point it can not learn further, and such deep networks tend to be less successful at generalizing from the trained data [11]. Recent work on deeper architectures has led to several breakthroughs in performance, however, by attempting to "decompose the problem into subproblems and multiple levels of representation" [11]. In an image recognition task, for instance, the network can find low-level features which can be directly derived from the pixel input (e.g. via edge detectors) and use those to create progressively higher-level features (each informed from the level of abstraction below it) until a sufficiently abstract and meaningful representation is achieved, such as the pattern of a human face. Intuitively, training an ANN with image pixels as input and the similarity to a human face as output would be much more challenging without the intermediate representations.

Deep learning, as described above, can be achieved through energy-based models such as Restricted Boltzman Machines [129, 224] but also through backpropagation. The latter is achieved by training a deep network one layer at a time; essentially, the deep network is treated as a stack of shallow networks where the output of the previous shallow network is the input to the next [256]. Training such shallow networks sequentially is possible by treating each shallow network as an autoencoder [96]. Autoencoders are non-linear models that transform an input space P into a new distributed representation Q by applying a deterministic parametrized function called the *encoder* $Q = f^{\mathbf{w}}(P)$. This encoder is often represented as a single layer of neurons with sigmoid activation functions, which is trained alongside a *decoder* (see Fig. 3.1) that maps back the transformed representation Q into the original representation ($P' = g^{\mathbf{w}}(Q)$) with a small reconstruction error, i.e. the original and corresponding decoded inputs are similar. By using a lower number of neurons than inputs, the autoencoder is a method for the lossy compression of data. By training an autoencoder via backpropagation to learn what is essentially the identity function, information is compressed in the hidden layer Q by exploiting typical patterns observed in the training set.

The robustness of this compression can be increased via *denoising autoencoders* (DA), an autoencoder variant that corrupts the inputs of the encoder during training while enforcing that the original uncorrupted data is reconstructed [255]. Forced to both maintain most of

the information from the input and undo the effect of corruption, the DA must "capture the main variations in the data, i.e. on the manifold" [255], which makes DAs far more powerful tools than linear models for principal component analysis.

3.2 Evolutionary Computation

As its name suggests, artificial evolution is designed to simulate real-world evolution following the Darwinian principle of "survival of the fittest". While artificial evolution was permitted by the advances in the processing power of computers from 1960 and onward, its various implementations were motivated by evolutionary biologists looking to test theoretical models of natural evolution, by engineers looking for faster, more efficient, or simply better methods for creating new artifacts, as well as by artificial-life researchers wishing to build new artificial evolutionary worlds which may or may not reflect natural evolution [118]. While there is a vast number of variants within artificial evolution (which in itself has not always been considered a unified field), most follow the Darwinian paradigm summed up by the following iterative process: individuals in a population are evaluated according to their desirability as mates, the most desirable usually then produce offspring which share their parents' characteristics, and the offspring replace a subset of the older population which dies off. The evolutionary process is thus built around the concepts of *reproduction* and *death*, with *selection* of individuals to reproduce or of individuals to die being of paramount importance. Selection, as per Darwinian theories, favors the fittest; in artificial evolution this usually is represented by a *fitness function* which guides both which individuals are the most promising for reproduction as well as which individuals are least promising and should thus die. It should be noted here that similar to human DNA, an individual's *genotype* is what is recombined during mating but the *phenotype* is what is usually evaluated in terms of fitness. Just like a human (i.e. the phenotype) is represented by 46 chromosomes (i.e. the genotype), the phenotype in artificial evolution is often stored in much more compact information in the genotype as will be presented in Section 3.2.4. Another important factor on the behavior of artificial evolution is the reproduction process itself, i.e. whether it involves two parents which mate, whether both parents contribute an equal amount of genetic material to the offspring or whether there is a chance of random mutation due to unexpected factors. Although human conception operates under the assumptions delineated above, artificial evolutionary methods have been experimenting with numerous schemes of recombination and mutation of genotypes, giving rise to different paradigms (and terms). To avoid the confusion of different — yet often overlapping — terms to characterize paradigms of artificial evolution, the following paragraphs will detail the most important aspects of most evolutionary computation (EC) approaches.

The choice of transferring genetic information from one generation to the next is a vital component — and a source of lengthy debates — of artificial evolution. The most rudimentary distinction is between *recombination* (or crossover) and *mutation*. Crossover requires at least two parents, whose genetic information is recombined into that of the offspring; inspired by most forms of mating in nature, sexual reproduction via mixing of two parents' genes was believed to enhance variation and thus exploration of the search space [98]. Crossover can be performed by segmenting both parents' genes at a random point and combining one parent's first segment with the other parent's second segment to create the genotype of the offspring. This simple crossover operation is labeled as the "1-point crossover"; genotypes can also be

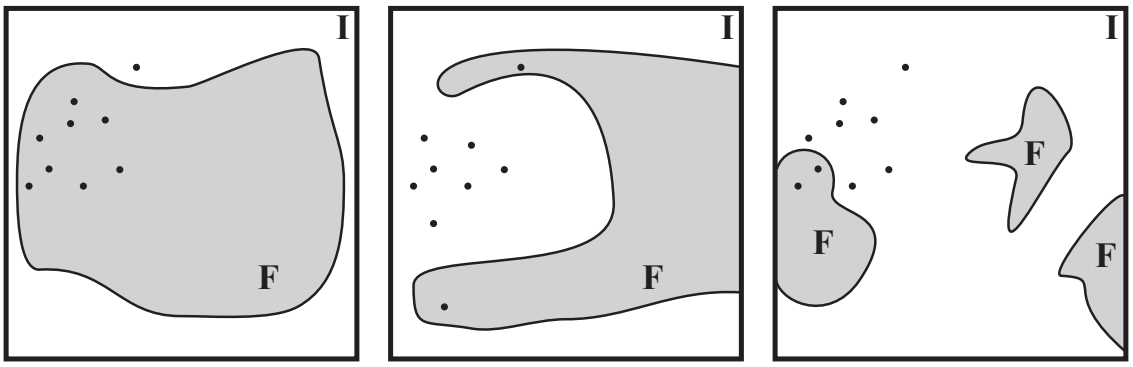
divided into more than two segments thus allowing for "n-point crossover". Crossover can also be "uniform", in which case each element of the offspring's genotype is copied from either parent, selected at random. Evolutionary methods using crossover are identified as *genetic algorithms* (which in this thesis include variants such as genetic programming methods). On the other hand, most forms of artificial evolution include some form of mutation which modifies the parent's genetic information when producing offspring. Mutation operates by copying the genetic information and modifying a part of it. Mutation can be applied to an offspring of two parents or to a single parent which creates an offspring (asexual mutation). How mutation modifies the genotype is highly dependent on the application and the form of the genotype: for instance, a genotype containing real-valued genes can have one, multiple, or all of these genes modified by a random number (which may follow a Gaussian distribution centered around 0). As a rule of thumb, mutation operators are designed as to not be particularly disruptive, i.e. to not modify much of the genetic information of the starting individual. Mutation and recombination operate on a different basis in terms of their exploration of the search space, as summarized by Spears: "[m]utation serves to create random diversity in the population, while crossover serves as an accelerator that promotes emergent behavior from components" [229]. Most EC researchers agree, however, that the details of the problem, its search space, its genotypical representation and the implementation details of the genetic operators can profusely affect the behavior of both crossover and mutation.

Selecting which individuals create offspring, and which individuals get replaced by these offspring is also of particular importance to artificial evolution. Assuming that the quality of all individuals can be computed as their fitness score (which will be covered in more detail in the next paragraph), the "canonical" artificial evolution process necessitates that a number of offspring are generated from the current population and replace a number of individuals in this population. Selection of parents can be deterministic or stochastic; stochasticity is often used to add noise and increase the robustness of the algorithm, lending to the general characterization of artificial evolution as "stochastic search". Popular selection schemes include elitist selection where only the fittest individuals of a population can create offspring, tournament selection where several potential parents are chosen at random from the population and "winners" among them are chosen based on fitness comparisons with the other potential candidates, and fitness-proportional (or roulette-wheel) selection which selects parents at random from the entire population, with the probability being proportionate to the fitness score of the parent in question compared to the total fitness of all parents. Once a number of offspring have been generated via any of these parent selection methods, the issue of selecting how (and if) these offspring are placed in the population raises the question of selecting survivors. Although many replacement strategies have been developed over the years, typically in steady-state evolution an offspring may replace an individual in the population only if it has a higher fitness, while in generational evolution all offspring replace the parents regardless of their fitness. In most implementations of generational evolution, a form of *elitism* secures that a portion of the fittest individuals in the previous generation persists in the next generation; with an elitism of 1, for instance, the fittest parent survives without any changes from one generation to the next. Several other parameters, such as the size of the population, the number of offspring created in every generation, or whether the offspring is compared only to its parent(s) or to all individuals of the previous generation can affect the performance of artificial evolution but can not be covered in detail for the purposes of this thesis; interested readers are directed towards numerous books on the topic [118, 98].

Although the selection and recombination methods connect the theory of natural evolution with evolutionary computation methods (and determine the categorization of those methods into genetic algorithms, evolutionary processes or evolutionary strategies), it is the fitness function that is of most interest in cases where EC is utilized for the purposes of problem solving. Defining the quality of a solution is tied directly to the problem at hand, and its design can affect not only the search but also the quality of the final artifacts produced. In many cases defining a fitness function is straightforward, such as in a pathfinding problem where the algorithm must minimize the distance of a path. In other cases, however, there may not be a clear way of distinguishing in which way one solution is better than another, and by how much. How much one solution is better than the other matters less for tournament or elitist selection but is of paramount importance in fitness-proportionate selection. As a concrete example, an agent controller in a computer game can use the game score at the end of its simulation (in case the agent died or completed a level) to ascertain its fitness score. However, even in the simplest of games such as *Ms. Pac-Man* (Namco 1982), the score combines information on the number of pellets eaten and number of edible ghosts eaten; in more elaborate games such as *Starcraft* (Blizzard 1998) the score can contain many more dimensions such as units created, enemy units killed, own units killed etc. When an artifact has several different objectives, then different fitness dimensions can be optimized via multi-objective evolution [42]. In such cases the different objectives are often conflicting and the best solution will be Pareto optimal, i.e. there will not be any solution which would increase the score in one fitness dimension without decreasing it in another dimension. Apart from such complicated problems with multiple variables, designing a fitness function is often not straightforward in cases where the evaluation of quality is subjective (e.g. when evaluating an art piece), when the evaluation of quality is approximate (e.g. when no accurate simulation of its performance exists or can be devised), when the problem is deceptive (e.g. when a solution is better locally than others but leads further from the global optimum in the long run) or when the problem includes constraints. The last type of problem will be further elaborated in Section 3.2.1, while deceptive problems can be dealt with by omitting a fitness function altogether as with Novelty Search, discussed in Section 3.2.2. Subjective evaluations, on the other hand, can be handled by asking human users, as will be discussed in Section 3.2.3. Finally, Section 3.2.4 will elaborate on the issue of representation (i.e. the mapping between genotype and phenotype) which was alluded to earlier, and will present methods of evolving representations which contain phenotypical information more efficiently.

3.2.1 Constrained Optimization

While evolutionary computation has a rich history of successful applications in solving numerical optimization problems, it has never been straightforward how constraints should be handled. Such constraints are ubiquitous and often non-eliminable in e.g. engineering problems [166], where solutions are often required to satisfy a minimal functional performance or safety and a maximal cost or size. Constraints divide the search space into feasible and infeasible spaces; depending on the problem, the feasible space may be fragmented, non-convex, or much smaller than than the infeasible space (see Fig. 3.2). The greatest challenge in constrained search spaces is the issue of handling *infeasible individuals*, i.e. individuals which do not satisfy one or more of the constraints. This challenge is augmented by the fact that in many cases optimal feasible solutions lie in the border between feasible and infeasible space



(a) Large feasible space. (b) Non-convex feasible space. (c) Islands of feasible space.

Fig. 3.2: A visualization of different search spaces, divided into infeasible space (I) and feasible space (F) along with a randomly initialized population (black dots). Problems with easily satisfied constraints have a large feasible space as in Fig. 3.2a, where most randomly initialized solutions are feasible. In some cases the feasible space may be particularly non-convex, and crossover between the two feasible solutions in Fig. 3.2b is likely to create infeasible offspring. In highly constrained spaces, feasible space may be fragmented as in Fig. 3.2c, making discovery of distant islands of feasible space unlikely.

[207], such as an inexpensive engineering component with a risk of failure close to the safety threshold.

During the early period of constrained optimization research, infeasible individuals were simply assigned a fitness of zero; in most selection/replacement methods this would result in infeasible individuals being killed off in favor of feasible individuals. This *death penalty* to the infeasible individuals may not be particularly destructive in cases where infeasible individuals are rare; however, it has been argued against by [164], as substantial genetic information stored in infeasible individuals is lost. In highly constrained spaces, where a feasible individual does not exist in the initial population, genetic search incorporating the death penalty amounts to random search until the first feasible individual is discovered.

The most popular method for handling constraints in genetic optimization is to reduce the fitness score of infeasible individuals by a *penalty* score. Such penalties can be a constant value [102], a measure of feasibility [56] or dynamically adapted according to the state of search [91]. Different methods of penalizing the fitness function of infeasible individuals are surveyed by [43]. Designing penalty functions is not a trivial task, as penalties which are too high may amount to a death penalty while penalties which are too low may lead to superfluous search in the infeasible space.

Infeasible individuals can also be converted to feasible individuals through some procedure of *repair*. In many cases, the "cost" of repairing such an individual is applied to its fitness score as a penalty. The repair mechanism is useful in cases where evaluating an infeasible individual is not possible via the same fitness function used for feasible individuals. For instance, if the infeasible individual cannot be simulated (e.g. for having negative mass with the physics model used) in a simulation-based evaluation process, converting the infeasible individual to feasible is the only way for calculating its fitness. However, designing a repair function requires significant domain knowledge and is often as challenging as solving the original problem, while applying a penalty based on repair costs has the same issues as other penalty functions.

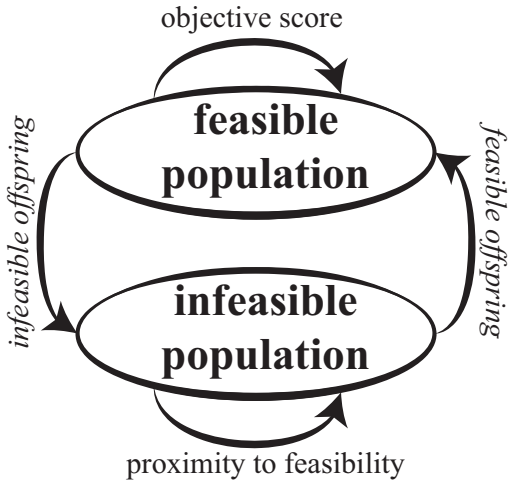


Fig. 3.3: Diagram of the two-population genetic algorithm (FI-2pop GA). For FI-2pop GA, the feasible population evolves to maximize an objective score, while the infeasible population evolves to minimize its members' distance from feasibility.

Constrained optimization can also be carried out by evolving infeasible individuals separately from feasible ones. As infeasible individuals do not have to compete with feasible ones within their population, their evaluation can be decoupled from the feasible individuals' fitness function. While other approaches mate infeasible individuals with feasible ones [125], the Feasible-Infeasible two population Genetic Algorithm (FI-2pop GA) only allows members of the same population to breed [123]. However, feasible offspring of infeasible parents migrate to the feasible population and vice versa; this indirect form of interbreeding allows the sharing of genetic information. While the feasible population evolves to maximize a problem-specific measure of quality, the infeasible population evolves to minimize its members' distance from feasibility (see Fig. 3.3). By guiding infeasible individuals towards the feasible-infeasible border, FI-2pop increases the likelihood that their offspring will become both feasible and on the border of feasibility, where the optimal solution often lies [207]. FI-2pop GAs have shown great promise in generating game content such as spaceship hulls [136], as well as platformer levels and maps for adventure games [227].

3.2.2 Novelty Search

Novelty search has been proposed by Lehman and Stanley [132] as an alternative to objective-driven search, and has shown great potential in domains where a fitness function is deceptive, difficult to quantify, or subjective. Instead of evolving towards maximizing an objective function approximating the quality of a solution, novelty search evolves towards diversifying the solutions in a population. Novelty search selects an individual according to its *novelty score*, which is the average *distance* between the individual and its closest neighbors. This distance function is usually a characterization of the phenotype; for robots or maze-solving agents, the distance function becomes a behavioral characterization, comparing the two robots' positions sampled at one second intervals [133] or the two agents' final positions [130], respectively. In order to prompt exploration of the search space, novelty search maintains a *novel archive*: in each generation, individuals with the highest novelty scores are added to the novel archive

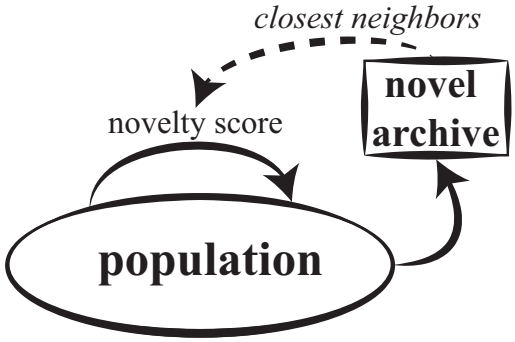


Fig. 3.4: Diagram of the novelty search algorithm. For Novelty Search, a single population evolves to maximize a novelty score which measures the average distance of an individual from its closest neighbors in the current population and in a *novel archive* which is updated in every generation with the most novel individuals.

(in some cases, only if their novelty score is higher than a threshold). When calculating the novelty score, the individual's closest neighbors are drawn both from the current generation and from the novel archive (see Fig. 3.4). As the novel archive expands with each generation, the novelty score of an area of the search space decreases; therefore, individuals in less explored areas of the search space are favored as they have fewer close neighbors in the novel archive. Therefore, the novelty score $\rho(i)$ of individual i, presented in eq. (3.3), is the average distance between individual i and its k closest individuals, either in the current population or in a novel archive.

$$\rho(i) = \frac{1}{k} \sum_{j=1}^{k} d(i, \mu_j) \tag{3.3}$$

where μ_j is the j-th-nearest neighbor of i (within the population and in the archive of novel individuals); distance $d(i, j)$ is a domain-dependent metric which evaluates the "difference" between individuals i and j.

The notion of *minimal criteria novelty search* (MCNS) was proposed by Lehman and Stanley [131] in order to limit novelty search to the "useful" portions of the search space. MCNS applies the death penalty described in Section 3.2.1 to infeasible individuals by reducing their novelty score to zero. Tested on navigating open mazes (without exterior walls), MCNS performed better than unconstrained novelty search as the latter explored areas outside the maze which led the agent away from the exit. For highly constrained problems, however, Lehman and Stanley warn that a population of infeasible individuals will simply perform random search; as a solution, they suggest that a known feasible individual is inserted in the population to jump-start evolution. Even with such measures, however, the fact that all infeasible offspring of feasible parents are almost guaranteed to be replaced results in losses of genetic information as argued by Michalewicz [164]. Depending on the shape of the infeasible space, this may result in search being carried out on a small island of feasible space with no way of exploring further.

3.2.3 Interactive Evolution

As its name suggests, Interactive Evolutionary Computation (IEC) is a variant of evolutionary computation where human input is used to evaluate content. As artificial evolution hinges on the notion of survival of the fittest, in interactive evolution a human user essentially selects which individuals create offspring and which individuals die. According to Takagi [242], interactive evolution allows human users to evaluate individuals based on their subjective preferences (their own *psychological space*) while the computer searches for this human-specified global optimum in the genotypical space (*feature parameter space*); as such, the collaboration between human and computer makes IEC a mixed-initiative approach. In interactive evolution a human user can evaluate artifacts by assigning to each a numerical value (proportionate to their preference for this artifact), by ordering artifacts in order of preference, or by simply selecting one or more artifacts that they would like to see more of. With more control to the human user, the artifacts in the next generation may match users' desires better; the user's cognitive effort may also increase, however, which results in *user fatigue*.

Interactive evolution is often used in domains where designing a fitness function is a difficult task; for instance, the criteria for selection could be a subjective measure of beauty as in evolutionary art, a deceptive problem where a naive quantifiable measure may be more harmful than helpful, or in cases where mathematically defining a measure of optimality is as challenging as the optimization task itself. Since it allows for a subjective evaluation of beauty, IEC has often been used to create 2D visual artifacts based on L-systems [160], mathematical expressions [214], neural networks [209] or other methods. Using interactive evolution in art was often motivated by a general interest in Artificial Life, as is the case with Dawkin's Biomorph [54]. In evolutionary art, human users may often evaluate not the phenotypes but situational system outputs specified by the phenotypes, in cases where the phenotypes are image filters or shaders [64]. Apart from 2D visual artifacts, IEC has also been used in generating 3D art [41], graphic movies [215], typographies [205] and graphic design [57]. Evolutionary music has also used IEC to generate the rhythm of percussion parts [247], jazz melodies [15] or accompaniments to human-authored music scores [104], among others. Outside evolutionary art and music, IEC has been used for industrial design [89], image database retrieval [37], human-like robot motion [174] and many others. The survey by Takagi [242] provides a thorough, if somewhat dated, overview of IEC applications.

Since interactive evolution is entirely reliant on human input to drive its search processes, its largest weakness is the effect of human fatigue in human-computer interaction. Human fatigue becomes an issue when the users are required to perform a large number of content selections, when feedback from the system is slow, when the users are simultaneously presented with a large number of content on-screen, or when users are required to provide very specific input. All of these factors contribute to the *cognitive overload* of the user, and several solutions have been proposed to counteract each of these factors.

As human users are often overburdened by the simultaneous presentation of information on-screen, user fatigue can be limited by an interactive evolutionary system that shows only a subset of the entire population. There are a number of techniques for selecting which individuals to show, although they all introduce biases from the part of the tool's designers. An intuitive criterion is to avoid showing individuals which all users would consider unwanted. Deciding which individuals are unwanted is sometimes straightforward; for instance, musical

tracks containing only silence or 3D meshes with disconnected triangles. However, such methods often only prune the edges of the search space and are still not guaranteed to show wanted content. Another technique is to show only individuals with the highest fitness; since fitness in interactive evolution is largely derived from user choices, this is likely to result in individuals which are very similar — if not identical — to individuals shown previously, which is more likely to increase fatigue due to perceived stagnation.

User fatigue is often induced when the requirement of a large number of selections becomes time-consuming and cumbersome. As already mentioned, fewer individuals than the entire population can be shown to the user; in a similar vein, not every generation of individuals needs to be shown to the user, instead showing individuals every 5 or 10 generations. In order to accomplish such a behavior, the fitness of unseen content must be somehow predicted based on users' choices among seen content. One way to accomplish such a prediction is via distance-based approaches, i.e. by comparing an individual that hasn't been presented to the user with those individuals which were presented to the user: the fitness of this unseen individual can be proportional to the user-specified fitness of the closest seen individual while inversely proportional to their distance [110]. Such a technique essentially clusters all individuals in the population around the few presented individuals; this permits the use of a population larger than the number of shown individuals as well as an offline evolutionary sprint with no human input. Depending on the number of seen individuals and the expressiveness of the algorithm's representation, however, a number of strong assumptions are made — the most important of which pertains to the measure of distance used. In order to avoid extraneous bias of the search from these assumptions, most evolutionary sprints are only for a few generations before new human feedback is required.

Another solution to the extraneous choices required of IEC systems' users is to crowdsource the selection process among a large group of individuals. Some form of online database is likely necessary towards that end, allowing users to start evolving content previously evolved by another user. A good example of this method is PicBreeder [209], which evolves images created by compositional pattern-producing networks (CPPNs) [230]. Since evolution progressively increases the size of CPPNs due to the Neuroevolution of Augmenting Topologies algorithm [233], the patterns of the images they create become more complex and inspiring with large networks. This, however, requires extensive evolution via manual labor, which is expected to induce significant fatigue on a single user. For that reason, the PicBreeder website[1] allows users to start evolution "from scratch", with a small CPPN able to create simple patterns such as circles or gradients, or instead load images evolved by previous users and evolve them further. Since such images are explicitly saved by past users because they are visually interesting, the user starts from a "good" area of the genotypical space and is more likely to have meaningful variations rather than if they were starting from scratch and had to explore a large area of the search space which contains non-interesting images.

Another factor of user fatigue is the slow feedback of evolutionary systems; since artificial evolution is rarely a fast process, especially with large populations, the user may have to sit through long periods of inaction before the next set of content is presented. In order to alleviate that, interactive evolution addresses it by several shortcuts to speed up convergence of the algorithm. This is often accomplished by limiting the population size to 10 or 20

[1] www.picbreeder.org

individuals, or by allowing the user to actively interfere directly on the search process by designating an estimated global optimum on a visualization of the search space [241].

To reduce the cognitive load of evaluating individuals, the most common solution is to limit the number of rating levels. In the simplest of cases, the rating levels can be two, i.e. selected and unselected, limiting the user's effort; they either like the content or they don't. Since taste is rarely a boolean value, however, usually more rating levels are included — often using existing scales such as that of "5 stars" which is popular in restaurant and cinema reviews. The interactive evolution interface for generating tracks for a car racing game [33] will be used as an example: this interface used both variations of rating levels for IEC, as well as collaborative evaluation for reducing the fatigue of a single user. In the system's *single-user mode*, human subjects were asked to play 10 generations of 20 evolved tracks each and evaluate them using two scoring interfaces: like/dislike and rating from 1 to 5 stars. The feedback provided by users about each track is used as a fitness in the evolutionary process. In the *multi-user mode*, the same population of 20 individuals is played and evaluated by five human subjects. The fitness given to each track in the population is the average score received by all users. The feedback provided by users showed improvements in the quality of the tracks and an increase in their interestingness.

3.2.4 Advanced Representations: CPPNs and Neuroevolution

From the choice of parameter encoding within a genotype [165] to the design of the process that defines how a phenotype is mapped to a genotype [13], "representation is long acknowledged a central issue for Evolutionary Computation" [206]. The most direct approach involves mapping all information of the phenotype directly to the genotype, where it is stored as bit arrays. With the genotype increasing in size proportionally to the phenotype, the added computational burden in combination with the slower convergence causes the bit-array representation to become impractical in problems with large phenotypes. Using images as an example, the sheer data contained in the phenotype (often thousands of pixels which can have millions of color values) emphasizes the need for an efficient encoding in the genotype. In such cases, the genotype must find an abstraction that somehow compresses the data contained in the phenotype; this abstraction will limit the genotype's size, reducing training time and making computations more manageable. By taking full advantage of *pleiotropy* (a single gene affecting multiple phenotypic traits) for reducing the genome's size and *polygeny* [69] (a single phenotypic trait being determined by the interaction of multiple genes) to preserve distinction between the phenotype's traits, the representational power of the genotype can be significantly increased. However, such abstractions within the genotype assume that a function exists that can translate them into the phenotype. Such a function becomes increasingly complex as more phenotypical information is compressed in a smaller genotype. Moreover, since the abstractions of the genotype often rely on unusual parameter encoding which is problem-specific (and highly deviates from the bit-array encoding and its established crossover and mutation operators), genetic operators such as mutation and crossover must be adapted in order to be able to evolve such genes efficiently. In such indirect mappings, a minor change in certain chromosomes may cause larger changes in the phenotype (due to pleiotropy) than similar changes in other chromosomes of the same genotype. The careful design of genetic operators is essential, especially in unconventional representations, in order to avoid problems in evolutionary

algorithms [13].

A vast repository of representations exists within the literature of evolutionary computation; of particular note are pieces of code evolving via Genetic Programming [188], the functions of which define the phenotype, or natural embryogenies which use simulated chemicals to "activate or suppress genes within the chromosomes, triggering patterns of cellular growth" [13]. However, another indirect representation which has been popular in evolution is that of the artificial neural network which, as mentioned in Section 3.1 can combine multiple inputs into multiple outputs; as ANNs can theoretically approximate any function, they can be a very versatile representation. Initial experiments in neuroevolution, i.e. the evolution of ANNs, revolved around the evolution of the weights of a neural network with a topology defined by the human programmer [262]. The burden of designing a neural network able to learn the task at hand or reproduce the intended behavior by hand was a burden to the algorithm's creator, relied on assumptions on ANN behavior, and was not guaranteed to take advantage of the search mechanisms of artificial evolution. Alternatives were proposed such as using subpopulations with different ANN topologies [86] or evolving the ANN structure along with the weights [90]. In the spirit of the latter, Stanley proposed the Neuroevolution of Augmenting Topologies (NEAT) algorithm [230] which evolves the topology and weights of neural networks, producing increasingly sophisticated behavior in control problems by complexifying the neural networks that determine it. NEAT begins the evolution with a uniform population of ANNs with the simplest topology (no hidden nodes) and random connection weights. As evolution progresses, more hidden nodes and links are added to the ANNs which bear a historical marking designating their order of appearance; this ensures that "genomes of different organizations and sizes stay compatible throughout evolution" [230]. Genetic diversity is maintained through *speciation*, with individuals competing primarily with members of their own species, allowing them to optimize their structure without being overwhelmed by individuals of different species with more complex (and thus more optimal) topologies.

While NEAT can be used to evolve ANNs with sigmoid functions, other representations may be more appropriate for certain tasks. In his paper "Exploiting Regularity without Development" [230], Stanley presented an encoding specifically designed to represent content with regularities. Assuming that development in nature consists of a series of progressively more localized coordinate frames (where a coordinate is a "conceptual device for describing an abstract configuration of any type" [230]). Stanley argues that development is analogous to a series of function compositions which transform the base coordinate frame to increasingly more localized coordinate frames with each transformation applied. This sequence of function compositions can be represented as a connected graph of such functions, with the initial coordinate frame as input and the most localized coordinate frames as output. Since CPPNs can be envisioned as ANNs with unconventional activation functions, they can be evolved via NEAT if a mechanism is added that determines how the activation function of a node is initialized and how it can mutate. The pattern-producing ability of CPPNs is particularly useful for art and games domains, and has been used to evolve drum tracks [103], musical accompaniments [105], dance moves [61], 2D images [209], 3D models [40], particles for weapon projectiles [94], and spaceship shapes [136]. CPPNs have also been used to represent the connectivity pattern of other ANNs via HyperNEAT [79].

3.3 Summary

This Chapter has presented previous and ongoing work on specific algorithmic problems revolving around the search for quality in different domains (including design, engineering, agent control, or art). The greater domain of evolutionary computation and its subdomains of constrained optimization, novelty search and interactive evolution were accounted for, along with the general principles of each. Moreover, the problem of machine learning was briefly presented along with its particular instances which are relevant to this thesis. Building on many of the principles described in this Chapter, a number of algorithms were designed for the purposes of this thesis. The algorithms of this thesis range from additions to existing algorithms (such as the offspring boost mechanism to FI-2pop GA) to combinations of algorithms from different domains (such as feasible-infeasible novelty search) to subversions of popular approaches (such as inserting objectives within interactive evolution with choice-based interactive evolution). These algorithms will be fully described in the next Chapter.

Chapter 4

Algorithms

Inspired by existing work on artificial evolution and machine learning presented in Chapter 3, a number of algorithms have been implemented for the purposes of this thesis. The goal of such algorithms is to provide a computational creator with sufficient (and sufficiently efficient) creative ability so that it can co-create alongside a human designer. In order to achieve that, the computational creator must be able to discern between playable and unplayable content; this is achieved via two-population genetic search techniques which distinguish feasible from infeasible individuals and place them in different populations. A computational creator can also have its own design priorities, favoring specific gameplay or aesthetic qualities which guide the search process of the FI-2pop GA described in Section 4.1. A computational creator can try to break the human designer's patterns in order to surprise the user with novel designs; this is achieved via constrained novelty search as described in Section 4.2. Finally, a computational creator can adapt its design priorities to better match those of the human user, in order to motivate collaboration between the two; a number of adaptive models of user preference have been implemented for that purpose, and are presented in Section 4.3.

4.1 Two-Population Optimization

As presented in Section 3.2.1, searching for high-quality content is not straightforward when such content must satisfy certain constraints. As the search space is divided into feasible (with artifacts which satisfy all constraints) and infeasible (with artifacts which do not satisfy all constraints), certain steps must be taken to ensure that feasible content is discovered quickly and that search is not "wasted" on exploring infeasible space. The Feasible-Infeasible 2-population Genetic Algorithm (FI-2pop GA) was developed by Kimbrough et al. to simultaneously search in both the feasible space and the infeasible space using different fitness criteria. The FI-2pop GA implemented for the mixed-initiative automata described in this thesis largely follows the principles described in [123], with the addition of a mechanism for increasing the number of offspring of feasible parents, which will be shown to have an impact both in the case of constrained optimization but also in the case of constrained novelty search.

In the FI-2pop GA paradigm, the number of offspring for each population is equal to the current population's size. However, previous experiments in constrained optimization have indicated that an *offspring boost* on the feasible population was beneficial for enhancing the optimization behavior of feasible individuals. When the feasible population is smaller than the infeasible population, the offspring boost mechanism forces members of the feasible population to create a number of offspring equal to 50% of the total size of the two populations. The number of offspring in the infeasible population is reduced accordingly to keep the to-

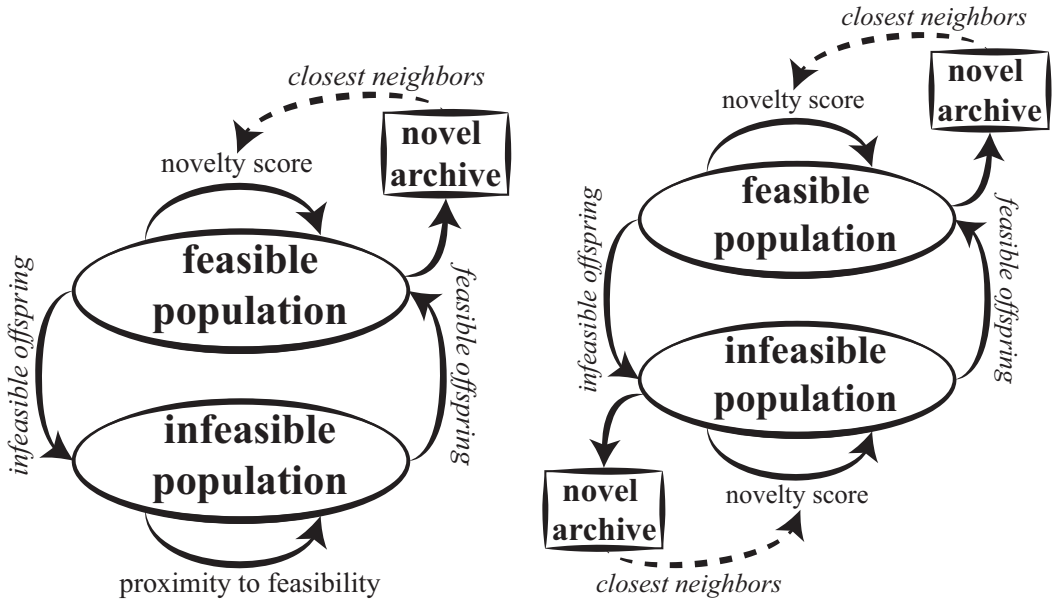
initialize population P containing N individuals;
for $i = 1$ **to** *max generations* **do**
    divide population into N_f feasible and N_i infeasible individuals, clear P;
    if $N_f < N_i$ **then**
        compute number of feasible offspring $O_f = N/2$; `/* offspring boost */`
        compute number of infeasible offspring $O_i = N/2$;
    else
        compute number of feasible offspring $O_f = N_f$;
        compute number of infeasible offspring $O_i = N_i$;
    end
    foreach *feasible individual* **do** evaluate its score based on feasible fitness;
    copy E fittest feasible individuals to the next population P;
    select parents among feasible individuals via roulette wheel;
    produce $O_f - E$ offspring from selected feasible parents, move to P;
    foreach *infeasible individual* **do** evaluate its score based on proximity to feasibility;
    copy E fittest infeasible individuals to the next population P;
    select parents among infeasible individuals via roulette wheel;
    produce $O_i - E$ offspring from selected infeasible parents, move to P;
    `/* at this point the next generation is stored in a single population` P
       `which is divided into feasible and infeasible in the next iteration */`
end

Algorithm 1: FI-2pop GA with offspring boost as used in this thesis.

tal population size steady. When the feasible population is larger than that of the infeasible population, on the other hand, the offspring boost is not applied and the number of feasible and infeasible offspring is equal to the number of feasible and infeasible parents respectively. The details of the FI-2pop GA are provided in Section 3.2.1; the implementation used in this thesis, along with the use of the offspring boost, are summarized in pseudocode in Algorithm 1. All instances of two-population genetic search (including two-population novelty search) use a roulette-wheel method for selecting parents; the same parent may be selected more than once. This design choice is due to the offspring boost mechanism, where the number of feasible offspring may be higher than the number of feasible parents; using roulette-wheel selection, fitter parents are likely to produce more offspring.

4.2 Two-Population Novelty Search

Novelty search is a relatively new paradigm for stochastic search described in detail in Section 3.2.2. By searching for divergent content, novelty search allows for the generation of designs that the user of a mixed-initiative tool can be surprised by. In many ways, novelty search can promote thinking "outside of the box" [55] by acting as a random, unexpected stimulus. However, game elements usually come with several playability constraints; this makes constrained novelty search a useful addition to current methods. Like objective-driven search before it, novelty search faces significant challenges when dealing with infeasible individuals. Early attempts at constrained optimization via Minimal Criteria Novelty Search [131] address

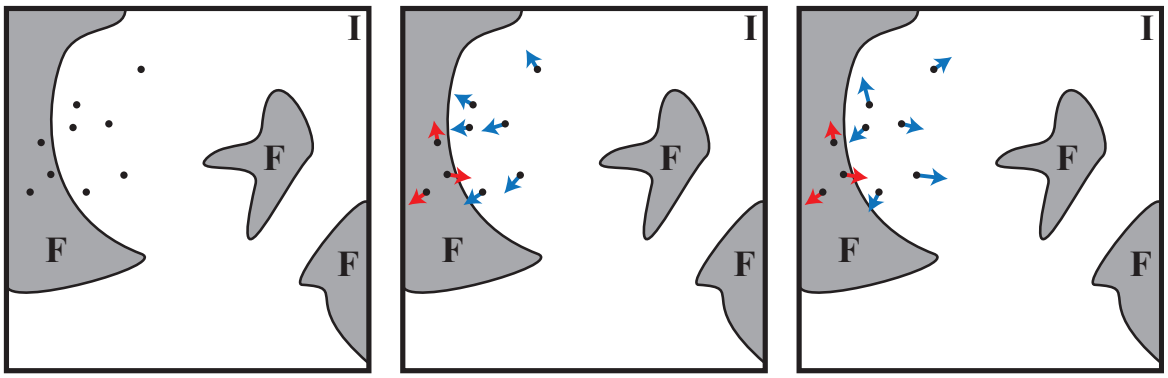


(a) FINS. (b) FI2NS.

Fig. 4.1: Diagrams of the two-population novelty search algorithms. For FINS, the feasible population performs novelty search using a novel archive containing only feasible individuals, while the infeasible population evolves to minimize the distance from feasibility. For FI2NS, the feasible population performs novelty search using a novel archive containing only feasible individuals; the infeasible population performs novelty search as well, but uses its own novel archive with only infeasible individuals and calculates the novelty score by measuring distance only between infeasible individuals.

this issue by applying the death penalty to infeasible individuals, although researchers in constrained optimization have argued against it [164]. As it follows a different selection process, many of the more advanced techniques of constrained optimization (such as penalizing the fitness score of infeasible individuals) can not be directly applied. For instance, it is unclear whether a penalty should be applied to $\rho(i)$ of eq. (3.3) for infeasible i or to $d(i, j)$ for feasible i but infeasible j. It is therefore preferable to avoid comparisons between infeasible and feasible individuals. The FI-2pop GA maintains two populations so that infeasible individuals do not compete with feasible ones for the purposes of selection; feasible parents can thus be selected using a completely different criterion (e.g. novelty search) than infeasible ones. Additionally, feasible offspring of infeasible individuals migrate to the feasible population; this is likely to increase the feasible population's diversity, which coincides with the goals of novelty search among feasible individuals.

Two variations of the two-population genetic algorithm have been adapted to the purposes of novelty search, both of which are summarized in pseudocode in Algorithm 2. *Feasible-infeasible novelty search* (FINS) evolves feasible individuals, in their separate population, towards maximizing the novelty score $\rho(i)$ as per eq. (3.3) while infeasible individuals evolve towards minimizing a measure of their distance from feasibility f_{inf} (see Fig. 4.1a and Fig. 4.2b). In order to test the impact of the f_{inf} heuristic and do away with objective-driven optimization on both populations, the *feasible-infeasible dual novelty search* (FI2NS) performs novelty search on both the feasible and the infeasible population (see Fig. 4.2c). For FI2NS, novelty search is



(a) Feasible and Infeasible Spaces. (b) FINS search process. (c) FI2NS search process.

Fig. 4.2: A visualization of the search process for two-population novelty search methods. Fig. 4.2a shows a possible search space divided into infeasible space (I) and islands of non-convex feasible space (F), along with a randomly initialized population (black dots). With FINS, the infeasible individuals evolve to minimize their infeasibility, ideally approaching the closest border with feasibility (Fig. 4.2b). With FI2NS, the infeasible individuals evolve to maximize their diversity, ideally discovering faraway islands of feasible space (Fig. 4.2c). For both FINS and FI2NS, feasible individuals evolve towards maximizing their novelty; in the example shown, novelty search is likely to cause feasible individuals on the feasibility border to become infeasible.

carried out independently in each population with two separate archives of feasible and infeasible novel individuals (see Fig. 4.1b); while both populations may use the same $\rho(i)$ metric, only the closest neighbors in the same population and archive are considered. Maintaining two populations for either FINS and FI2NS ensures that distances between feasible and infeasible individuals are not considered in the calculation of $\rho(i)$ in eq. (3.3).

4.3 Adaptive User Preference Modeling

Just as a human creator, so must the computational creator take into account affordances, constraints and stylistic preferences while creating artifacts such as art, engineering schematics, or game content. In such mixed-initiative design software, programmers *de facto* insert their personal preferences into the tool's algorithms; as in traditional art and design, the programmer assumes that the inserted preferences are representative of the preferences of the intended users or the general public. While market research informs human creators of the public's wishes, a computer program evolving designs has the potential of assessing their users' preferences by interacting directly with them. To a certain extent, interactive evolution achieves that in the same way that an art critic determines if a finished work is good or not. However, it is much more useful to understand the reasoning behind the user's response: what drove them to like a specific design and discard another? By understanding the underlying causes of these choices, such a program could guide the creative process based on the user's preferences.

A number of methods have been developed in order to realize such a program. This is accomplished via a quantifiable measure of user's taste derived from a weighted sum of quantifiable fitness dimensions, each representing a facet of user taste. Using this measure of user's pref-

initialize population P containing N individuals;
initialize empty novel archive A_f of feasible individuals;
if *method is FI2NS* **then** initialize empty novel archive A_i of infeasible individuals

for $i = 1$ **to** *max generations* **do**
  divide population into N_f feasible and N_i infeasible individuals, clear P;
  if $N_f < N_i$ **then**
    compute number of feasible offspring $O_f = N/2$; `/* offspring boost */`
    compute number of infeasible offspring $O_i = N/2$;
  else
    compute number of feasible offspring $O_f = N_f$;
    compute number of infeasible offspring $O_i = N_i$;
  end
  foreach *feasible individual* **do**
    evaluate its score based on novelty with other $N_f - 1$ individuals and A_f archive
  end
  copy E fittest feasible individuals to the next population P;
  select parents among feasible individuals via roulette wheel;
  produce $O_f - E$ offspring from selected feasible parents, move to P;
  if *method is FI2NS* **then**
    foreach *infeasible individual* **do**
      evaluate its score based on novelty with other $N_i - 1$ individuals and A_i archive
    end
  else if *method is FINS* **then**
    foreach *infeasible individual* **do** evaluate its score based on proximity to feasibility;
  end
  copy E fittest infeasible individuals to the next population P;
  select parents among infeasible individuals via roulette wheel;
  produce $O_i - E$ offspring from selected infeasible parents, move to P;
end

Algorithm 2: FINS and FI2NS with offspring boost, as used in this thesis.

erence (the *aesthetic score*) as a fitness function of a genetic algorithm, the content generator can optimize game elements with the desired properties as dictated either by a designer (offline) or a user (online). By observing the choices or rankings of a user, the algorithm can discern the underlying factors affecting their choice(s) and focus on those factors for generating future content. This approach is unique as it follows a two-step adaptation procedure for the generation of personalized content adjusting both the content generated but also the computational model that assesses content quality. The ongoing adjustment of the focus of the generative process is expected to eventually lead to personalized computational models of game aesthetics and furthermore to the creation of high-quality content matching the personal preferences of the user.

Such a preference model can be adjusted over a series of *iterations* of user interaction, weight adjustment and content evolution. In each iteration a number of artifacts is presented to the user who provides feedback based on their personal preference; this feedback is used to adjust the weights of the preference model, and the updated preference model is then used

initialize population P containing N individuals;
initialize all weights of preference model F to $1/D$ where D the number of fitness dimensions;
repeat `/* one iteration */`
  for $i = 1$ **to** *generations per iteration* **do** `/* short evolutionary sprint */`
    foreach *individual* **do** evaluate its score based current preference model F;
    copy E fittest individuals to the next population P;
    select parents among individuals via roulette wheel;
    produce $O_f - E$ offspring from selected parents, move to P;
  end
  select B presented artifacts from P ($B \leq P$);
  if *method is CIE* **then**
    user chooses one favorite s among presented artifacts, remaining are unselected (u);
    tries=0;
    while *tries*<*max tries* **and** $\exists i \in u$ *with* $F(s) < F(i)$ **do**
      tries++;
      foreach *fitness dimension* k *of* F **do** update weight k based on eq. (4.1);
      if *training error increases from previous try* **then** break from while loop;
    end
  else if *method is RIE* **then**
    user provides ranking r for p presented artifacts, as $r(x_i) \forall i \in p$;
    tries=0;
    while *tries*<*max tries* **and** $\exists i \in p$ *with* $r_{x_i} > r_{x_j}$ *but* $F(x_i) < F(x_j)$ **do**
      tries++;
      foreach *fitness dimension* k *of* F **do** update weight k based on eq. (4.2);
      if *training error increases from previous try* **then** break from while loop;
    end
  end
until *until user fatigued*;

Algorithm 3: RIE and CIE as used in this thesis.

to evolve a population of artifacts, a sample of which is shown in the next iteration. This thesis introduces two methods of user feedback: the choice of a single artifact as favorite, or *choice-based* feedback (detailed in Section 4.3.1) and the ranking of artifacts in order of preference, or *rank-based* feedback (detailed in Section 4.3.2). Both methods are summarized in pseudocode in Algorithm 3.

4.3.1 Choice-based Interactive Evolution (CIE)

The simplest approach from a user's perspective is the selection of a single artifact as the best among those presented; the *choice-based interactive evolution* (CIE) method, introduced in [138], uses an adaptive model informed by the user's choice of one favorite artifact among a range of presented content. The goal of the adaptive model in such an approach is to reward fitnesses with a higher fitness score in the selected artifact than in the unselected ones while penalizing fitnesses with a lower fitness score in the selected artifact than in the unselected

ones. Towards that end, the weight w_k of fitness k is updated on each epoch t as follows:

$$w_k^{t+1} = w_k^t + \alpha(f_{k_S} - \bar{f}_{k_U}) \tag{4.1}$$

where α is a weight update step, f_{k_S} is the selected artifact's score for fitness k and $\bar{f}_{k_U}$ is the average score for fitness k across all unselected artifacts. Assuming weight adjustment of the selected individual's fitness property k, eq. (4.1) follows the key principles of the Widrow-Hoff [263] weight update rule.

The weights are adjusted until the selected artifact has the highest preference score F among those presented; the adjustment process can be prematurely terminated if the preference score difference between the highest scoring artifact and the selected artifact starts to increase or after a maximum number of weight updates. Since the preference score only measures the relative contribution of individual fitness dimensions, the final adjusted weights are divided by $\sum_i |w_i|$ resulting in normalized weight values.

4.3.2 Rank-based Interactive Evolution (RIE)

In order to increase the amount of information that the user provides to the system on each iteration, this thesis proposes a method in which the user is asked to rank the generated content in order of decreasing preference. It is expected that for a subset of the presented artifacts, users will not have a strong preference of one over the others; in such scenarios they are encouraged to rank those artifacts equally. Equal ranking is interpreted as an unknown relative preference between those artifacts; this option might lead to a certain loss of information but also lowers the task's cognitive load and reduces the amount of *noise* inherent to subjective reports [267]. To facilitate the task, users are also allowed to rank only a subset of the presented artifacts; it is assumed that ranked artifacts are preferred over non-ranked artifacts, which would be equivalent to rank the remaining artifacts last.

Following a gradient descent algorithm, the rank-based interactive evolution algorithm iteratively adjusts the weights of the network to minimize the cost function of eq. (3.2). For the linear model used in this study, the update on each epoch t is:

$$w_k^{t+1} = w_k^t - \alpha \frac{\partial C}{\partial w_k} \tag{4.2}$$

$$\frac{\partial C}{\partial w_k} = \sum_{\substack{\forall (x_i \succ x_j) \in P: \\ F(x_i) - F(x_j) < b}} f_k^j - f_k^i \tag{4.3}$$

where α is the learning rate (0.01 for this study) and w_k^t is the weight of fitness f_k at epoch t.

This update is repeated until the cost function is equal to zero or a maximum number of epochs is reached. Similarly to CIE, the weights are normalized by dividing with $\sum_i |w_i|$.

4.4 Summary

This Chapter elaborated on the core algorithms which are used for the mixed-initiative tools contained within this thesis. The feasible-infeasible two population genetic algorithm largely

follows the literature [123], apart for an additional mechanism that boosts the number of feasible individuals in cases of severe imbalance between populations (in terms of population size). The FI-2pop GA is used for evolving map sketches for Sentient Sketchbook, and experiments which test its performance with different parameters and in different domains are reported in Chapter 7. The feasible-infeasible novelty search (FINS) and feasible-infeasible dual novelty search (FI2NS) are departures of the original FI-2pop GA which integrate the novelty search paradigm in one or both populations respectively. While only FINS with a specific parameter setup is used to evolve map sketches for Sentient Sketchbook, both two-population constrained novelty search techniques are tested extensively along with other constrained or unconstrained methods of novelty search in Chapter 8. Finally, a designer's preference for one type of design over another can be captured (and accommodated) based on their choices or rankings among presented content, via Choice-Based Interactive Evolution (CIE) or Rank-Based Interactive Evolution (RIE) respectively. Such adaptive models of the designer's style are described as potential additions to the current implementation of Sentient Sketchbook, and *in vitro* experiments with Choice-Based Interactive Evolution are presented in Chapter 9. Moreover, experiments with both CIE and RIE have been performed for the interactive evolution of strategy game maps, which are presented in Section 10.2 of Chapter 10.

The majority of these algorithms are integrated in one way or another into the Sentient Sketchbook mixed-initiative design tool; how the tool integrates the algorithms will be described in Chapter 5, along with a presentation of its functionalities and design principles. Moreover, other game design tools use these algorithms in an automated or semi-automated way; those will be described in Chapter 10.

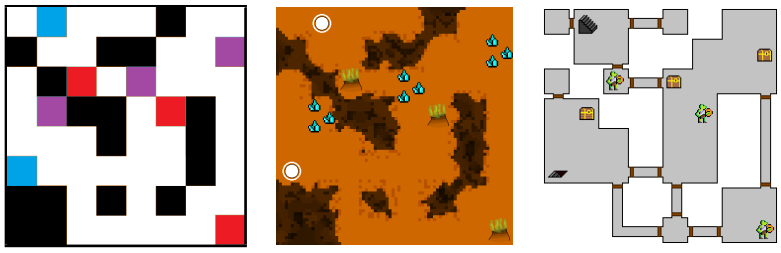
Chapter 5

Sentient Sketchbook

While a number of mixed-initiative design automata have been implemented and tested for the purposes of this thesis, by far the most successful in integrating design principles such as sketching with most of the algorithmic methods presented in Chapter 4 is Sentient Sketchbook. Sentient Sketchbook is a mixed-initiative design tool which allows a designer to create low-resolution map sketches described in Section 5.1. As a computer-aided design tool, Sentient Sketchbook tests maps for playability constraints, evaluates the map on gameplay properties, supports informative map displays and adds details to the coarse map sketch; the evaluations and constraints are presented in Section 5.2, while the user interface in Section 5.4. The tool also aims to contribute to the design dialog by generating map suggestions for the user; although the general methods have been discussed in Chapter 4, how these algorithms are integrated into Sentient Sketchbook is presented in Section 5.5. Section 5.6 describes proposed extensions to Sentient Sketchbook which can recognize and accommodate (in its suggestions) the designer's process, preference and visual goals.

5.1 What is a Map Sketch?

Map sketches are low-resolution, high-level abstractions of game levels. As a design medium, a map sketch follows several of the properties of sketches according to Hugh Dubberly: "A sketch is incomplete, somewhat vague, a low-fidelity representation. The degree of fidelity needs to match its purpose, a sketch should have 'just enough' fidelity for the current stage in argument building." [30]. The low resolution of the map sketch refers both to the map



(a) (b) (c)

Fig. 5.1: A map sketch (Fig. 5.1a) is ambiguous in the sense that it can represent a strategy game level (Fig. 5.1b) or a dungeon (Fig. 5.1c). Moreover, the meaning of red, blue, and purple special tiles is similarly ambiguous: red tiles can be roaming enemies (Fig. 5.1c) or valuable resources that can be mined (Fig. 5.1b), blue tiles can be spawn areas for players (Fig. 5.1b) or level exits (Fig. 5.1c) and purple tiles can be types of resources that must be built on (Fig. 5.1b) or collectible treasure (Fig. 5.1c).

size but also to the components which comprise it: a map sketch contains only the types of game elements which are absolutely vital for this type of game level. Such game elements may incorporate several different game elements which serve a different purpose; the *ambiguity* of such visualizations within the map sketch is an essential component of its "sketchiness", as the designer can envision elements of a map sketch as different things. The sketches' "use of ambiguity makes them evocative rather than didactic and mysterious rather than obvious" [80]. As an example, a map sketch of a strategy game may contain a single resource-type tile; this resource-type tile may be a mineral deposit or a vespene geyser in *Starcraft* (Blizzard 1998), or in *Dawn of War* (Relic 2004) it can be any of the following: a strategic point, a critical point, a relic or even a slag deposit. By removing such nuances from the design process, the human designer can focus on the big picture instead of getting bogged down in the details. As such, map sketches as described here are ideal for rapid prototyping and concept generation.

Within this thesis, a map sketch is described as a (small) number of tiles laid out in a grid. The tiles represent different game-dependent properties and have different navigational properties, i.e. allow movement through them (passable) or not (impassable). Map sketches can have passable tiles, impassable tiles as well as *special tiles* which are domain-dependent (e.g. bases, traps, spawn points, checkpoints). Special tiles can be further divided into different sets of tile types, and are usually the definitive factors of the map's gameplay. Special tiles are considered to allow movement. While this thesis focuses on top-down game maps, other types of levels e.g. for platformer and side-scrolling games can also be visualized as map sketches; example extensions of the map sketch concept are covered in Section 11.2.4.

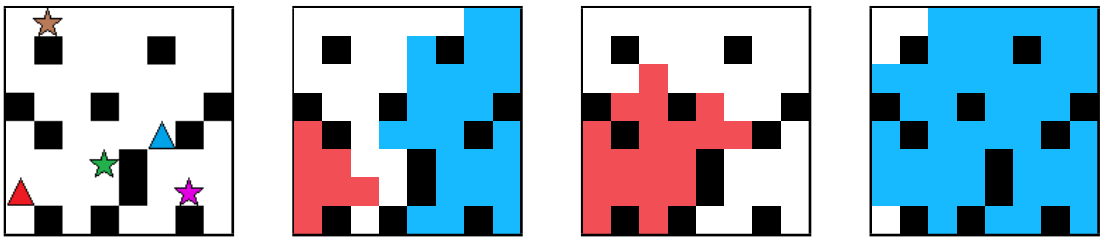
5.2 Evaluating Game Levels

In order to provide feedback to the user and to guide the generation of suggestions, a number of quantifiable metrics have been defined for evaluating map sketches. In order for certain metrics to be quantifiable, certain constraints (pertaining to connectivity and number of special tiles) must be met; these constraints are discussed in Section 5.2.2 while the formulas for the game metrics are presented in Section 5.2.1.

5.2.1 Evaluating Level Patterns

In order to inform the formulas which can evaluate game levels in a domain-independent (or at least domain-minimal) fashion, the closest analogue is the study of Bjork and Holopainen in general game design patterns [19]. Although this study includes hundreds of patterns, some of the more high-level patterns which translate well to level design are those of *symmetry*, *area control* and *exploration*. Symmetry is "a common feature in games to ensure that players have equal opportunities" and instantiates patterns such as player balance or team balance[1]. Area control "can give access to otherwise unavailable actions and can make the use of actions and tactics easier" and usually incorporates control over the game's resources. Finally, exploration is the "goal of learning the layout of the game world, or locating specific parts or objects in it" and assumes that some information about the level is initially imperfect or uncertain.

[1] To avoid confusion between the gameplay symmetry of [19] and visual symmetry, this thesis uses the term *balance* instead.



(a) (b) (c) (d)

Fig. 5.2: Sample quality evaluations for the map sketch in Fig. 5.2a with S_M (stars), S_N (triangles) and impassable tiles (black). Measuring safety of M tiles with regards to N tiles, the purple star is much closer to the blue triangle than to the red triangle, and thus has a high safety value. The remaining stars have small safety values since they are equally close (green star) or equally faraway (brown star) from both triangles. Measuring the area control of N tiles, Fig. 5.2b displays safe tiles for the red triangle (A_1 in red) and for the blue triangle (A_2 in blue). Measuring the exploration of N tiles, Fig. 5.2d displays exploration from the red triangle to the blue triangle ($E_{1\to2}$) and Fig. 5.2d displays exploration from the blue triangle to the red triangle ($E_{2\to1}$).

In order to be meaningful, area control and exploration require a set of two or more tiles identified as *reference tiles* (S_N); reference tiles usually have a special purpose in the game, such as player bases. For control measures, each reference tile can "own" a number of nearby tiles, if it is closer to these tiles than all other reference tiles. The safety of any tile t to a reference tile i is measured by eq. (5.1); the closest reference tile i has a positive safety value, while remaining reference tiles have a safety value of zero. For exploration measures, it is assumed that if a large part of the map must be covered in order to discover one reference tile when starting from another reference tile, then the exploration value is high. This map coverage is simulated by a flood fill algorithm, and the exploration required from a reference tile i to all other reference tiles is shown in eq. (5.2).

$$s_{t,i}(S_N) = \min_{\substack{1\le j\le N \\ j\neq i}} \{max\{0, \frac{d_{t,j} - d_{t,i}}{d_{t,j} + d_{t,i}}\}\} \tag{5.1}$$

$$E_i(S_N) = \frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq i}}^{N} \frac{E_{i\to j}}{P} \tag{5.2}$$

where N is the number of elements in set S_N; $d_{t,i}$ is the distance from tile t to element i; P is the number of passable tiles on the map; and $E_{i\to j}$ is the map coverage when a four-direction flood fill is applied starting from element i and stopping once element j has been found (see Fig. 5.2).

According to Bjork and Holopainen [19], control encompasses both control of areas and control of strategic resources: the former includes all passable tiles and the latter includes only tiles which provide bonuses or serve a particular function (usually special tiles). While both evaluations of control require the definition of reference tiles, the safety of strategic resources also requires the definition of tiles constituting strategic resources (named *target tiles* or S_M). The mathematical formulation of strategic resource control (f_s), area control (f_a) and exploration

(f_e) are shown below:

$$f_s(S_N, S_M) = \frac{1}{M} \sum_{k=1}^{M} \max_{1 \leq i \leq N} \{s_{k,i}\} \tag{5.3}$$

$$f_a(S_N) = \frac{1}{P} \sum_{i=1}^{N} A_i \tag{5.4}$$

$$f_e(S_N) = \frac{1}{N} \sum_{i=1}^{N} E_i \tag{5.5}$$

where M and N is the number of elements in sets S_M and S_N respectively; $s_{k,i}$ is the safety metric of element k of S_M to element i of S_N, i.e. $s_{k,i}(S_N)$ from eq. (5.1); A_i is the map coverage of safe tiles for element i (see Fig. 5.2b); P is the number of passable tiles in the map and E_i is the exploration metric from eq. (5.2) for element i. A tile t is safe for element i if $s_{t,i} < C_s$; the constant $C_s = 0.35$ throughout this thesis, as it creates a good ratio of contested areas in most maps.

The evaluations of f_s, f_a and f_e either average or aggregate among reference tiles. To ensure that reference tiles are symmetrical in terms of these evaluations, e.g. that the exploration value of one reference tile is not much larger than the others', the evaluations of *balance* are introduced. The mathematical formulation of strategic resource control balance (b_s), area control balance (b_a) and exploration balance (b_e) are shown below:

$$b_s(S_N, S_M) = 1 - \frac{1}{MN(N-1)} \sum_{k=1}^{M} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} |s_{k,i} - s_{k,j}| \tag{5.6}$$

$$b_a(S_N) = 1 - \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{|A_i - A_j|}{max\{A_i, A_j\}} \tag{5.7}$$

$$b_e(S_N) = 1 - \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{|E_i - E_j|}{max\{E_i, E_j\}} \tag{5.8}$$

5.2.2 Playability constraints

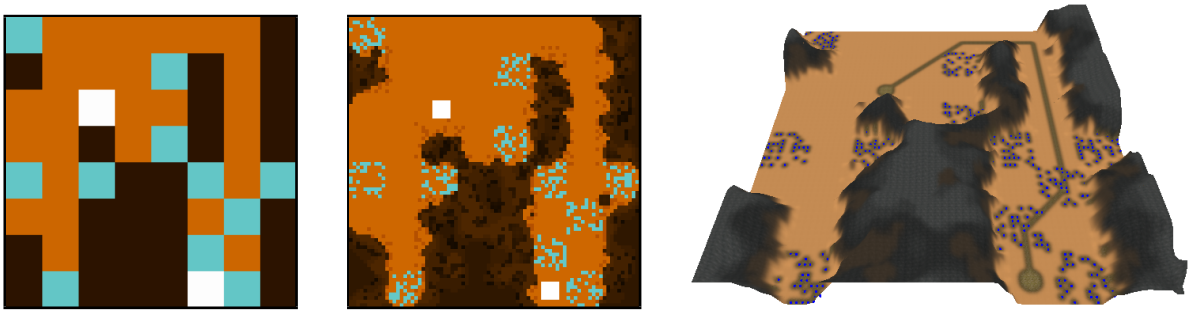
Each type of level comes with specific playability constraints; such constraints may be quite elaborate as demonstrated by [223, 220] although for the purposes of Sentient Sketchbook only connectivity constraints are considered. A map sketch is unplayable if any of the special tiles (tiles with a specific game purpose e.g. bases and resources in strategy games or team spawn areas in first person shooter levels) is not connected with the remaining special tiles via a passable path. Certain gameplay properties evaluated via the level evaluations described in Section 5.2.1 have additional requirements apart from the connectivity of the special tiles considered. Evaluations of $f_a(S_N)$, $f_e(S_N)$, $b_a(S_N)$ and $b_e(S_N)$ require at least two tiles of type S_N, and similarly $f_s(S_N, S_M)$ and $b_s(S_N, S_M)$ require at least two tiles of type S_N and one tile of type S_M.

5.3 A concrete example: map sketches for strategy games

One of the most straightforward map sketches, and the first to be implemented and tested, was the sketch of a strategy game level. Strategy games are in their vast majority antagonistic: two players vie for supremacy, either by destroying each other's headquarters as in *Warcraft II* (Blizzard 1995) or in being the first to amass the necessary resources to win the game in e.g. the Space Race of *Civilization IV* (Firaxis 2005). Strategy games usually have a method for motivating players to control large parts of the map; this is achieved via the use of resources. Resources may be necessary to support a larger economy or army, as is the case of *Warlock 2: the Exiled* (Paradox 2014) or may allow the construction of special units in *Civilization IV*. In certain games, resources can be collected by sending workers from the player's base as in *Age of Empires* (Microsoft 1997), in which case the workers must be protected throughout their journey. In other games, a resource can be collected by simply claiming it in *Heroes III* (New World Computing 1999), by building a city near it in *Civilization IV* or by building a structure on top of it as in *Dawn of War* (Relic 2004). Resources may be finite and can be depleted as in *Starcraft* (Blizzard 1998) or can be infinite and provide a benefit every turn as in *Heroes III*.

Although strategy game levels can contain many different types of resources, each with different properties, in the abstraction that Sentient Sketchbook works on the special tiles of strategy game map sketches consist of player bases (B), and resources (R). Fig. 5.3a displays a map sketch of a strategy game level, while Fig. 5.3b and 5.3c shows the complete level it can create. As described in 5.2.2, in order for a strategy game level to be playable it must allow each base to reach all enemy bases, while all resources must be accessible from every base. Following typical strategic gameplay, each player is expected to start at a base tile (chosen at random) and collect resources in order to build units; such units must reach the enemy players' bases and destroy them.

Granted the popularity of competitions using real-time strategy games such as *Starcraft*, the high degree of antagonism places player balance as the most significant property of high-quality levels. In addition, easily controlled space near each player largely determines the difficulty of the map, while readily accessible resources allow players to build up their forces



(a) Map Sketch (b) Final Level (2D) (c) Final Level (3D)

Fig. 5.3: An example map sketch for a strategy game, with bases (B) in white, resources (R) in cyan, and impassable tiles in dark brown (Fig. 5.3a). Using the constructive algorithms described in Section 5.4.3, this map sketch can be converted to a final map in 2D (Fig. 5.3b) or in 3D (Fig. 5.3c)

before attacking their enemies. When resources are predominantly in contested areas, players must rush to secure them with the few forces they can initially muster, creating a faster and more aggressive gameplay. Finally, the location of the players' bases also affects the pace and challenge of the map since enemy bases which are difficult to find can afford the enemy player time to build up their forces without being harassed.

Translating the desirable properties of strategy game maps to evaluation functions of Section 5.2.1 is straightforward. Area control around bases amounts to $f_{saf} = f_a(S_B)$, with its corresponding evaluation of balance $b_{saf} = b_a(S_B)$. Easy access to resources from bases amounts to $f_{res} = f_s(S_B, S_R)$, with its corresponding evaluation of balance $b_{res} = b_s(S_B, S_R)$. Discovery of enemy bases is measured by $f_{exp} = f_e(S_B)$ and its corresponding evaluation of balance $b_{exp} = b_e(S_B)$. Since many of the experiments in this thesis use strategy game map sketches, the f_{res}, f_{saf}, f_{exp}, b_{res}, b_{saf} and b_{exp} notation will be used in several of the following Chapters.

5.4 Tool Layout

A graphical interface has been developed to allow a user to manually edit a map sketch for strategy game levels (see Figure 5.4). The map editor allows the designer to place tiles on the map; while the designer edits the map, the tool tests it for *playability*, offers alternative views pertaining to navigational properties of the user's sketch and updates the evaluations of game-dependent level quality.

5.4.1 Displayed Evaluations

The *fitness dimensions* presented in eq. (5.3)–(5.8) are recalculated after every user interaction and displayed on-screen. If a fitness dimension can not be evaluated because of unreachable special tiles or an insufficient number of reference/target tiles (see Section 5.2.2 for details), then that fitness dimension displays "N/A" to indicate infeasibility.

In addition to these fitness dimensions, several other *metrics* are computed and displayed via the map editor, mostly pertaining to map navigation. The displayed metrics include the number of bases and resources on the map, the shortest paths' length among bases[2], the percentage of used space and the number of chokepoints, dead ends and open areas. *Used Space* consists of passable tiles which are on a shortest path between any two bases or any base and any resource. *Chokepoints* are tiles with two neighboring (passable) tiles, *dead ends* are tiles with one neighboring tile and *open areas* are tiles with the maximum number of neighboring tiles.

5.4.2 Alternative Views

In addition to the numerical display of the fitness dimensions, there is a set of optional displays for the user's sketch which visualize different navigational properties. Fig. 5.5 provides

[2]Specifically the metrics display the shortest path length between the closest bases, the shortest path length between the furthest bases, and the average path length among all bases.

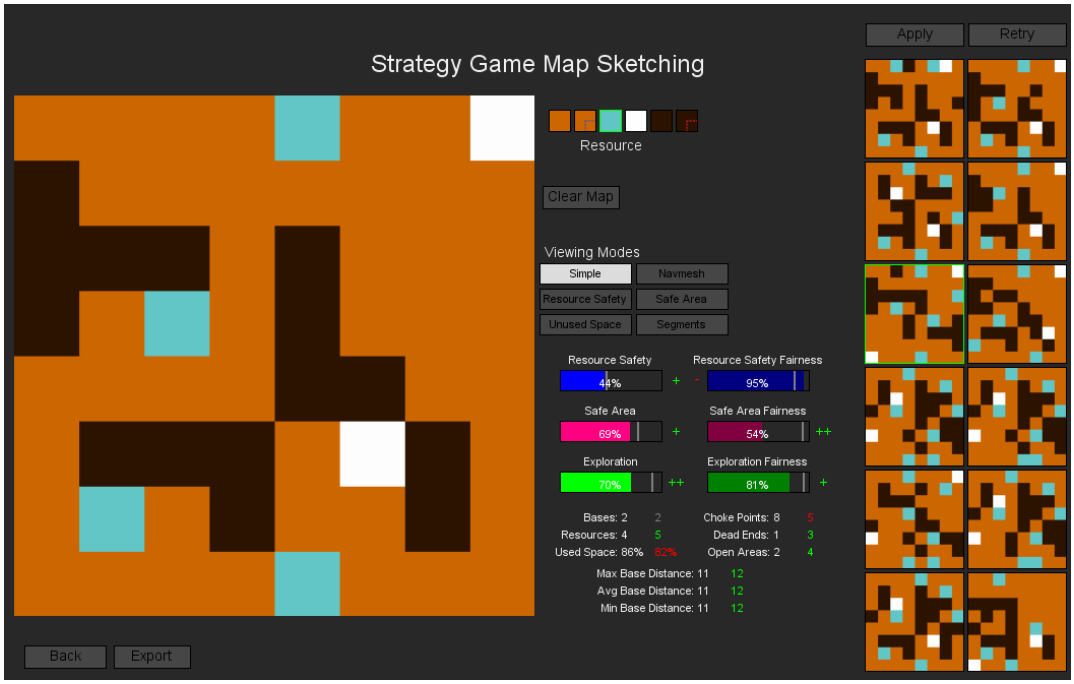
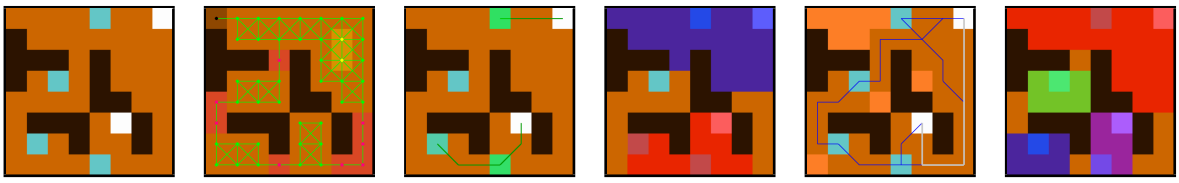


Fig. 5.4: The Sentient Sketchbook tool during a design session of a strategy game level. To the left is the sketch editor, far to the right are the automatically generated map suggestions; between these elements is the tile palette, the map display menu and an overview of map's fitness dimensions and metrics.

examples of these views for strategy game maps. The navigational grid (navmesh) of Fig. 5.5b is ubiquitous for all top-down game levels and provides visual indications of tiles constituting open areas, chokepoints and dead-ends. Other, specialized visualizations such as the safety of resources in Fig. 5.5c or the safe areas of bases in Fig. 5.5d are specific both to the domain of strategy games and to the presence of a fitness dimension for f_{res} and f_{saf} respectively.

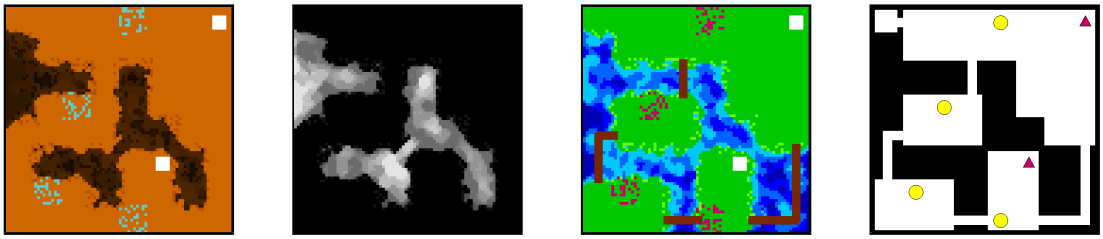
5.4.3 From Sketches to Final Maps

At any time, the designer can switch to the final map view which displays the complete map on which an actual game can be played; Figure 5.6 provides examples of how the strategy game sketch of Fig. 5.5a can be transformed into a full strategy game level or even another type of game level (see Fig. 5.6d). The visualizations of the final map show how map sketches can be interpreted in different ways and how they can be applied to other game genres, while other visualizations of final maps for different genres will be shown in Chapter 7. These final maps are generated in a constructive manner, via rule-based systems which may also include a certain degree of randomness as is the case with the initial seeds of cellular automata of maps in Fig. 5.6a–5.6c. The randomness of the constructive algorithms creates an organic-looking map, but is carefully designed in a way that retains all the properties (chokepoints, paths) of the coarse map sketch. Manual editing, evaluation and evolution are all done on the sketch level, making computations such as pathfinding easier and reducing the effort required to design a complete map.



(a) Default (b) Navmesh (c) Resource safety (d) Safe areas (e) Unused space (f) Segments

Fig. 5.5: Different map displays on a small two-player map sketch describing a strategy game level: *Default* displays passable tiles (light brown), impassable tiles (dark brown), resources (cyan) and player bases (white). *Navmesh* displays the passable pathways on the map and the location of choke points (red), dead ends (black) and open areas (yellow); *Resource safety* displays the safety value of each resource (in shades of green) and connects bases to their safe resources. *Safe areas* shows the tiles close to each base (here in red and blue) which are considered safe. *Unused space* shows all shortest paths between bases and resources, and highlights leftover unused space (orange). *Segments* shows the passable areas (in different colors) which are surrounded solely by chokepoints.



(a) Default (b) Heightmap (c) Waterways (d) Dungeon

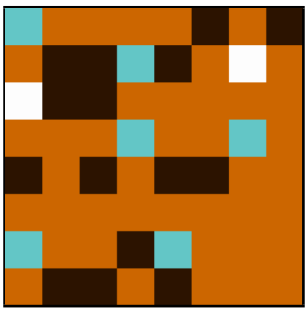
Fig. 5.6: Visualizations of the final map, which offers a higher-detail view of the map sketch in Figure 5.5. The *Default* visualization adds detail to resource tiles and impassable tiles, while *Heightmap* elaborates the impassable regions of *Default*. *Waterways* treats impassable tiles and chokepoints as water and replaces chokepoints with bridges, while *Dungeon* divides the passable segments and dead ends into rooms and treats chokepoints as corridors.

5.5 Generating Map Suggestions

In order for Sentient Sketchbook to approximate the role of a colleague, it should not only evaluate the user's current design but also provide new and unexpected alternatives to it. Through these *suggestions*, the tool intends to challenge the user's current design focus and provide unforeseen alternatives to achieving the designer's goals. The map suggestions are updated while the user edits the map, using the current map's appearance as their starting point. Currently map suggestions are generated via genetic algorithms (GAs) performing constrained optimization to maximize either the maps' scores in the fitness dimensions described in Section 5.2.1 or to maximize the maps' diversity. Six genetic algorithms, running on separate threads, optimize maps in a single fitness dimension each as elaborated in Section 5.5.4; the best evolved individual of each GA is included in the map suggestions. A seventh genetic algorithm evolves maps via novelty search as elaborated in Section 5.5.5; the six most different evolved maps in this GA are also included in the map suggestions. Every genetic algorithm runs for 10 generations, and optimizes 10 individuals which initially consist of mutations of

1	0	0	0	0	4	0	4
0	4	4	1	4	0	2	0
2	4	4	0	0	0	0	0
0	0	0	1	0	0	1	0
4	0	4	0	4	4	0	0
0	0	0	0	0	0	0	0
1	0	0	4	1	0	0	0
0	4	4	0	4	0	0	0

(a) Genotype (1D integer array).



(b) Phenotype (map sketch).

Fig. 5.7: Any map sketch can be represented genotypically as an integer array. In this instance, the strategy game map sketch of Fig. 5.7b is represented in the genotype of Fig. 5.7a, with impassable tiles stored as 4, bases stored as 2, resources stored as 1 and empty tiles stored as 0.

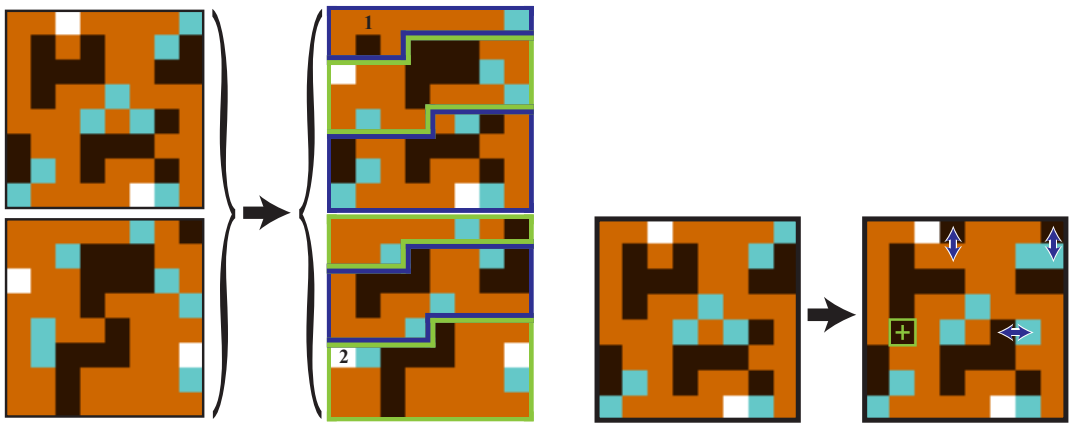
the authored map sketch. The 12 map suggestions (six evolved via objective-based search and six evolved via novelty search) are displayed, as thumbnails, on the edge of the tool's window; maps identical with the user's map or with each other are omitted. The user can at any time select a map suggestion and replace their current sketch with it. While a suggestion is selected, its scores in the fitness dimensions and its other metrics are displayed along the relevant scores in the current sketch, clearly indicating which scores increase or decrease (see Fig. 5.4).

5.5.1 Map Sketch Representation

In order to evolve map suggestions, each map sketch is directly encoded in a genotype as an array of integers with an integer denoting whether the tile is impassable, empty or a specific type of special tile. An example of a genotype and its corresponding phenotype can be seen in Fig. 5.7a and 5.7b respectively. The direct encoding used has the least representational bias, while the small size of map sketches does not hinder its evolutionary optimization.

5.5.2 Genetic Operators

In all of the experiments included in this thesis, evolution is carried out via fitness-proportionate roulette-wheel selection regardless of the population it is in (feasible or infeasible). The same parent may be selected more than once and thus generate multiple offspring, which is necessary when the number of offspring is different than the number of parents (e.g. when applying the offspring boost). In each population, the best individual is carried to the next generation while remaining individuals are replaced by new ones. New individuals can be created via a 2-point crossover of two parents (see Fig. 5.8a) or via mutation (see Fig. 5.8b). Mutation alters between 5% to 20% of the maps' tiles (determined randomly): each tile has an equal chance of being swapped with a randomly chosen adjacent one, or transformed from passable to impassable and vice versa (special tiles are not transformed). The mutation scheme is specialized to the task of level design and the direct representation of map sketches: by altering a controlled number of tiles, suggestions are expected to both appear different without being



(a) Example of 2-point crossover.

(b) Example of mutation.

Fig. 5.8: Examples of genetic operators applied on sketches of strategy game levels. Due to the direct representation, crossover creates two offspring with parts of the first map (light outline) and the second map (dark outline). Crossover may result in offspring with more bases or resources than their parents; assuming a designer constraint for maps with exactly two bases, the repair mechanism must be applied as the offspring have 3 bases and 1 base respectively. For the top map the repair mechanism removes a base chosen randomly between the three, and the base at location 1 is changed to passable tile. For the bottom map the repair mechanism adds a base on a randomly chosen passable tile, and the tile at location 2 is changed to base. Mutation changes a few tiles on the map: in Fig. 5.8b that amounts to 6% of the map. Mutated tiles may be shifted with an adjacent one (dark arrows) or transformed from passable to impassable or vice versa (+ sign).

unrecognizable. Moreover, since mutation does not alter the number of special tiles (although it can alter the number of passable tiles), the maps are less likely to become infeasible due to too few or too many special tiles — as will be discussed in Section 5.5.3.

5.5.3 Constraint Satisfaction

All maps are tested for playability constraints laid out in Section 5.2.2. Two minimal criteria exist: (a) there must be a maximum and a minimum number of special tiles which fulfill all the included fitness dimensions' constraints, and (b) all special tiles must be connected via passable paths.

The number of special tiles allowed is ultimately a designer decision, but is also dependent on the size of the map: small maps can only have a few special tiles, while larger maps might have sufficient open areas for a larger number of bases and therefore players. Moreover, the special tiles must also have a minimal value if they are used to evaluate the fitness evaluations of Section 5.2.1; the tool assumes that the designer takes such constraints into account when designated the maximum and minimum bounds of such special tiles. The current mutation scheme does not change the number of special tiles (e.g. bases or resources for strategy game levels) on the map, and thus mutating a gene with a feasible number of special tiles is guaranteed generate a gene with a feasible number of special tiles. When applying crossover, however, there is a likelihood that the offspring maps will have more or less special tiles than their individual parents; in such cases, a *repair* mechanism alters the offsprings' genotype to satisfy constraints on the number of special tiles. The repair mechanism adds a missing special

tile to infeasible individuals with missing special tiles (of that type), or removes extraneous special tiles from infeasible individuals with excess special tiles (of that type) — without any penalties to their fitness. The process is stochastic, as missing special tiles are added to randomly selected passable tiles while excess special tiles are chosen at random and removed (see Fig. 5.8a). Although this repair mechanism increases the unpredictability of the crossover operator, it is not as destructive as creating two infeasible offspring.

When two special tiles are not connected, there is no straightforward way of repairing the map; therefore, the connectivity of special tiles is the hard constraint warranting the two-population approaches. Feasible individuals have all special tiles connected to each other via passable paths; infeasible individuals have at least one disconnected path. The infeasible population evolves towards minimizing the distance from feasibility; the distance from feasibility d_{inf} for a set S_I, which normally includes all special tiles on a map, is:

$$d_{inf}(S_I) = \frac{2u_c}{I(I-1)} \tag{5.9}$$

where I the number of elements in set S_I and u_c the number of disconnected pairs of elements in set S_I.

5.5.4 Suggestions from Objective-Driven Search

Each of the six suggestions which are derived from objective-driven search is generated via its own thread running a FI-2pop GA, described in Section 4.1. The FI-2pop GA maintains an infeasible population evolving towards minimizing the distance from feasibility of eq. (5.9) as well as a feasible population which evolves to minimize a single objective among those in eq. (5.3)–(5.8). The sequence of suggestions is always the same, i.e. the first suggestion on the top left always corresponds to the fittest individual in the GA evolving towards the first objective. In the unlikely event, however, that a GA contains no feasible individuals by the end of the short evolutionary sprint then the suggestion is omitted and the next suggestion (generated on its own thread) takes its place. Thus, the spatial arrangement usually aids users to guess the properties of each suggestion although there is always a chance that fewer than 12 suggestions will be shown at any time.

5.5.5 Suggestions from Novelty Search

Apart from the six suggestions generated by six individual threads running objective-driven search as described in Section 5.5.4, a FINS algorithm runs on its own thread towards generating six diverse map suggestions. FINS is a variant of two-population novelty search which has been described in Section 4.2. Using d_{inf} of eq. (5.9) for its infeasible population, the feasible population of this FINS algorithm evolves towards optimizing the novelty score of eq. (3.3), i.e. the average difference between the evaluated individual and past or current solutions. Past solutions are stored in the novel archive; unless otherwise noted, the 5 most novel individuals of the feasible population are added to the novel archive in each generation. Unless otherwise noted, the FINS considers only the 20 closest neighbors to the evaluated individual when calculating its novelty score ($k = 20$ in eq. (3.3)).

Measuring the difference between two maps is no straightforward task and the choice of a distance metric for novelty search is likely to affect the appearance of generated individuals. Since users of the Sentient Sketchbook tool inspect the generated map suggestions visually, it is assumed for the purposes of this study that visual diversity between generated maps is the most desirable quality. The distance metric for *visual diversity*, shown as d_{vis} in Eq. 5.10 compares two maps on a tile-by-tile basis: the fewer matching tiles between maps, the higher the score of d_{vis}. This d_{vis} metric will be used to calculate the novelty score according to Eq. 3.3 for all experiments in this thesis.

$$d_{vis}(i,j) = \frac{1}{WH} \sum_{x=1}^{W} \sum_{y=1}^{H} D_{x,y}(i,j) \tag{5.10}$$

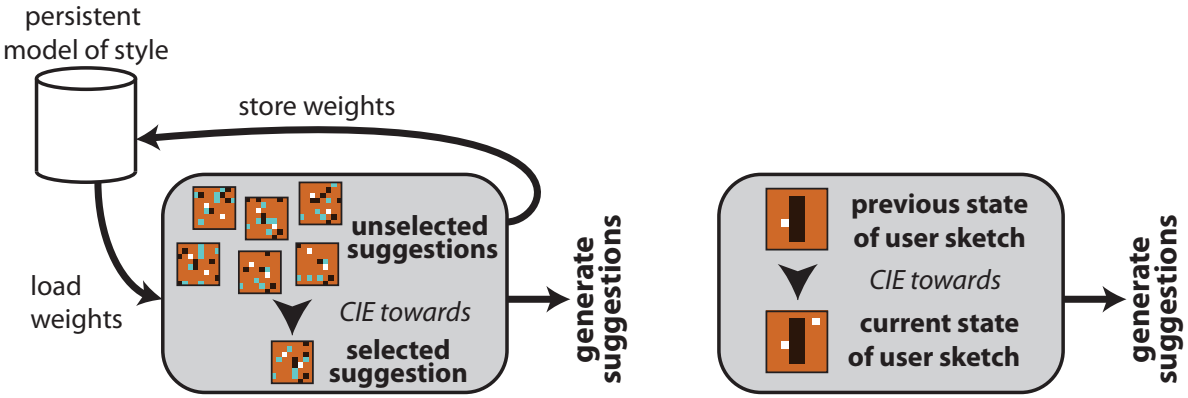
where W and H is the width and height of the map respectively and $D_{x,y}(i,j) = 0$ if at coordinates (x,y) the tile of map i is of the same type as that of map j and $D_{x,y}(i,j) = 1$ if it is of a different type.

5.6 Designer Modeling Extensions

During the user evaluations of Sentient Sketchbook for strategy game level design, which is described in Chapter 6, computer-generated suggestions were not used in several cases. Based on the observations in Section 6.2 and inspired by the hypotheses of designer modeling described in Section 2.5, a number of designer models were implemented in order to address cases where suggestions did not seem appropriate to the designer. Models of process and, to a lesser degree, preference strive to make the suggestions more relevant to what the user is currently doing, while a model of symmetry goals strives to make the appearance of shown suggestions more in tune with the visual goals of the human creation. Each of these models will be described in its own Section below.

5.6.1 Accommodating a Designer's Style

A straightforward way of personalizing content to a designer is by biasing dimensions which are deemed important to the user based on their past choices of suggestions. This model of style is equivalent to choice-based interactive evolution described in Section 4.3.1 and assumes that each user has an overarching style which favors certain strategic properties over others, e.g. levels with winding paths and scattered resources. Since Sentient Sketchbook presents multiple suggestions per step, the user's selection of one suggestion over others is assumed to contain information on their overall style. The model of style is represented as a weighted sum of all fitness dimensions, where the weights are adapted to capture those strategic qualities prevalent (either higher or lower) in the selected suggestion compared to unselected ones. These weights can be negative, in which case they drive objective-driven search towards maps with e.g. fewer safe resources. Weights are adjusted via the simple but straightforward weight update rule of eq. (5.11): if the fitness score f_n (in dimension n) of the selected suggestion is higher than the averaged fitness score in dimension n of the unselected suggestions, then the weight w_n increases; vice versa, if the fitness score of the selected suggestion is lower, then w_n decreases.



(a) Modeling Style (b) Modeling Process

Fig. 5.9: Modeling style and process, with one sequence of weight updates (via CIE) shown inside the gray box. The model of style only learns when a user selects a suggestion, and it stores the weights into a persistent database. The model of process learns every time users edit their sketches but does not store the updated weights; after every user action, the model always starts learning from weights of 0.

$$w_n^{t+1} = w_n^t + \alpha(f_{n_S} - \bar{f}_{n_U}) \tag{5.11}$$

where α is a weight update step (0.01 for this case), f_{n_S} is the selected suggestion's score for fitness n and $\bar{f}_{n_U}$ is the average score for fitness n across all unselected suggestions.

Weight updates are performed for a maximum number of epochs ($3 \cdot 10^5$ in this case) or until the selected map has a higher score (calculated as the weighted sum of fitness scores for all dimensions) than all other maps. Once adaptation is complete, the weights are normalized so that the sum of their absolute values is 1. The designer's style, as captured by the weighted sum, can be directly used as a fitness function for the feasible population of the FI-2pop GA which creates Sentient Sketchbook's suggestions.

5.6.2 Extracting the Designer's Process

Modeling a designer's overall style can be useful for long periods of interactions with extensive use of suggestions. Modeling the designer's process, on the other hand, is more situational and focuses on what the user is *currently* doing. The proposed model takes into account the designer's current point in the process (i.e. their current sketch) and their sketch prior to the last applied action; such an action could be painting a set of impassable tiles, adding or removing a base or resource, or replacing a sketch with a suggestion. The model assumes that the user implicitly prefers their current version of the sketch to the previous one; this is true for most cases since few user actions (such as erasing tiles) can be considered a "step backwards" in the design process.

As with the model of style, the model of process largely follows the paradigm of choice-based interactive evolution in the sense that a fitness score is derived from a weighted sum of strategic qualities, where the weights are adapted according to the user's last action (and its initial state

(a) Map (b) V_t (c) V_b (d) H_l (e) H_r (f) P_t (g) P_b (h) P_l (i) P_r

Fig. 5.10: The effects of different symmetry types and map halves when creating symmetrical maps from the base sketch of Fig. 5.10a. Vertical reflection is shown as V_t (reflecting the top half) and V_b (reflecting the bottom half); horizontal reflection is shown as H_l (reflecting the left half) and H_r (reflecting the right half). Point symmetry centered on the map's midpoint is achieved by rotating by 180^o the top, bottom, left or right half of the map (P_t, P_b, P_l, P_r respectively).

and end state). The scores in different fitness dimensions of the current and previous sketch are used to adapt the weights of the model of process, using eq. (5.11) where f_{n_S} is the score of the current sketch in dimension n and $\bar{f}_{n_U}$ is the score of the previous sketch (since the score is singular, it is not averaged). However, the model adapts its weights via eq. (5.11) for a minimum of $5{\cdot}10^4$ epochs, even when the current sketch has a higher score than the previous; this places increased importance to those dimensions that are persistently prevalent. The model of process can use weights adapted from the previous action to bias the model of the next action (as a form of "memory"), similarly to the model of style. However, the extensions described here assume that the highly situational nature of the creative process necessitates a *tabula rasa* approach by setting each step's initial weights at 0; in this fashion, the model has no memory of previous actions and is not biased by assumptions of strategic quality except those behind the user's latest action.

5.6.3 Fulfilling Goals of Symmetry

While models of style and process focus on the level qualities of the generated suggestions, a popular pattern in competitive multiplayer game levels as well as in the interaction data of Sentient Sketchbook in Chapter 6 is the prevalence of symmetries in the designers' creations. Earlier work documented in Section 10.2 included symmetries as fitness functions and adapted a preference model towards more symmetrical levels. Instead, the extensions proposed here assume that symmetry is not an explicit designer preference but an intended goal of the design process. The model of symmetry goals, as it will be identified, assumes that once users introduce a certain symmetry to their level (even in early steps) they are expected to follow that symmetry throughout the process, even when focusing on different strategic qualities such as chokepoint or resource placement. As visual symmetry is assumed to be orthogonal to strategic quality (e.g. a map can be balanced even if it does not appear symmetrical), the search process can target fitness dimensions of strategic quality while the mapping from genotype to phenotype is constrained to create only symmetric maps. This potentially allows the search to be biased by models of style or process while the appearance of the artifacts is biased towards the target visual "feel".

The model of symmetry goals introduces several restrictions to the mapping between genotype to phenotype, which ensure either reflective or point symmetry (see Fig. 5.10). Since these symmetries are along axes passing through the map's midpoint, the genotype contains data

for the tiles of one half of the map sketch (top, bottom, left or right); the phenotype uses that data with the necessary inversions to create both halves of the map sketch. Choosing which mapping to use for suggestions depends on the current symmetries of the user's sketch, if any such symmetries exist at all. In order to find the best mapping for symmetric maps as well as which half of the user's sketch should seed the population of suggestions, eight symmetric versions of the base sketch are created (see Fig. 5.10) and evaluated on their similarity with the base sketch. The symmetry heuristic (sym) is calculated via eq. (5.12) and assigns equal importance to each tile type regardless of the number of tiles of this type in the map sketch, as long as they exist. If the best scoring symmetry is above a minimum threshold ($sym > 0.75$ in this case), the suggestions offered as alternatives to the current sketch will use that mapping while their initial population will consist of mutations of the half of the map designated by the symmetry type. If no symmetry is above the threshold, the genotype describes the entire map sketch as in the default setting of Sentient Sketchbook.

$$sym = \frac{1}{N_T} \sum_{n=1}^{N_T} \frac{n_\cup - n_\cap}{n_\cup} \tag{5.12}$$

where $n_\cup$ is the number of tiles of type n which are in either the base sketch or its symmetric version; $n_\cap$ is the number of tiles of type n which are at the same position in both the base sketch and its symmetric version; N_T is the subset of non-empty tile types which are present in the base sketch. For instance, if a strategy game map if there are no impassable tiles in the base sketch, then $sym = \frac{1}{2}\left(\frac{B_\cup - B_\cap}{B_\cup} + \frac{R_\cup - R_\cap}{R_\cup}\right)$; B and R are bases and resources, respectively.

5.7 Summary

This Chapter provided both the design principles and implementation details of Sentient Sketchbook, which attempts to enhance the designer's creative process via algorithmically generated suggestions. Sentient Sketchbook operates on an abstraction of a game level which is easy both for a human user to quickly draft and change but also for a computational creator to evaluate and generate. By using this abstraction, evolutionary search of permutations to the user's designs is possible without significant computational overhead; this allows suggestions to be evolved essentially in real-time, providing constant feedback to the user. Experiments with simulated users and proof-of-concept tests for these designer modeling extensions are presented in Chapter 9. The response of human users and their interactions with Sentient Sketchbook are covered in Chapter 6. The evolutionary processes of Sentient Sketchbook, and their ability to generate good and diverse suggestions, will be evaluated through methodical experimentation in Chapter 7 and 8 respectively. While not in place in past versions of Sentient Sketchbook, this Chapter has also introduced designer modeling extensions which can recognize and accommodate a designer's style, process and goals in the generated suggestions, either directly or indirectly.

The map sketches of Sentient Sketchbook allow for the coterminous creative process of a human and a computer; this makes Sentient Sketchbook ideal for rapid prototyping and concept development. In other instances of game levels or other artifacts, or in other phases of the design process, such abstractions may not be possible. Other methods for introducing computational initiative, in different degrees and forms, are discussed in Chapter 10.

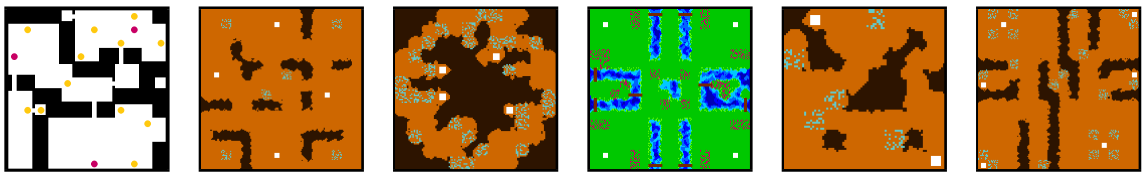
Chapter 6

Human Use of Map Sketches

As presented in Chapter 5, Sentient Sketchbook assists human designers in level creation via real-time map evaluation and playability checks, via a number of map displays and via generated suggestions. This Chapter addresses the usability of the tool from the perspective of the human user via a small-scale user study with 5 expert users consisting of independent game developers, game programmers and level designers; all of the users have a Master of Science in Games (focusing on technology or design). At the time of this user study, Sentient Sketchbook did not exhibit any of the designer modeling additions of Section 5.6, and the final map visualizations were less elaborate than those of Fig. 5.6; however, the heuristics of level quality and the algorithms for creating suggestions (FI-2pop GA and FINS) follow the same properties as described in Chapter 5. The tool was sent via e-mail to the participants, who were asked to interact with it as they saw fit on their spare time; feedback and interaction data was returned via e-mail. Each participant had several map design sessions; in each session a user creates a complete map starting from a blank canvas. Data is collected from 24 design sessions, with each participant contributing a minimum of 4 sessions. A general analysis of the data is presented with regards to user experience and usability in Section 6.1 and with regards to the use of suggestions in Section 6.2. Section 6.3 provides an in-depth analysis of design sessions with small maps.

6.1 General Interaction and User Feedback

Each user created between 4 and 6 final maps of varying map sizes (24 in total), a sample of which are displayed in Figure 6.1; all of the users' final maps can be observed in Appendix A. Users could choose between small maps (8 by 8 tiles), medium maps (12 by 12 tiles) and large maps (16 by 16 tiles). Participants favored medium-sized maps in 42% of design sessions; remaining 29% of maps were small, and 29% large. Most maps had 2 bases (50%) or 4 bases



(a) (b) (c) (d) (e) (f)

Fig. 6.1: A sample of the final maps created by five participants interacting with the Sentient Sketchbook; the range of map sizes and visualizations is shown.

(25%); all small maps had 2 bases. Unlike bases, resources vary — from 2 to 12 on Small maps, 6 to 27 on Medium maps and 6 to 40 on Large maps. Regarding map appearance, participants predominantly favored symmetrical features, with bases placed at the corners of the map as in Figure 6.1d or following symmetries on non-Cartesian axes as in Figure 6.1b.

The number of user actions taken (placing tiles, clearing the map or applying suggestions) in each design session range from 13 to 168 (58 on average). Observing the changes in fitness scores after each action, it appears that f_{saf} and b_{res} are consistent with the designer's process as user actions lead to increased scores in these fitness dimensions. Using the difference between positive and negative changes over the total number of fitness changes as the test statistic, obtained values (0.41 and 0.24, respectively) are significant, with $p < 10^{-6}$ for both fitness dimensions.

Overall, the expert users' reception of Sentient Sketchbook was positive, especially regarding the different map displays; a user stated that "I used the alternative map displays quite often [...] especially the navmesh [...] and I checked at least once for each map the resource safety display to guarantee some safe resources for everyone". The fitness dimensions were also mostly accurate, although a user commented on the f_{saf} fitness that "I don't consider empty space valuable to a player, safe or not, I only care about the resource points within it". While one user initially commented on the suggestions' apparent lack of visual structure and symmetry, they noted that after a while "the tool starts pushing you to places you didn't really consider" such as "working with some high level abstract ideas for the maps [...] instead of doing something symmetrical and intricate". Directions for improvement were also provided, such as speeding up the generation of suggestions for large maps, being "able to lock some features" in the map suggestions and improving the detection of chokepoints, which were deemed "confusing". These suggestions have not been addressed yet, but should be considered for future versions of Sentient Sketchbook.

6.2 Degree of Use for Generated Suggestions

Regarding the suggestions generated by the computational creator, participants made use of them in 15 out of 24 design sessions. Based on qualitative observations of the survey's interaction data as well as user feedback, cases where suggestions did not prove useful fell under four categories:

(a) instances where a user had preplanned a map layout before starting their design session; for instance, a user wanted to "re-create the map from the first Warlords game".

(b) instances where the strict visual symmetries of human maps were not retained in the generated suggestions; as a user puts it, "when I do something very symmetrical, it feels like the generator just does a more messy version of the same map".

(c) instances where suggestions did not retain the number of bases of the user's sketch. Since adding or removing bases (and therefore players) would need a thorough re-evaluation of the designer's current ideas, such suggestions were not preferred.

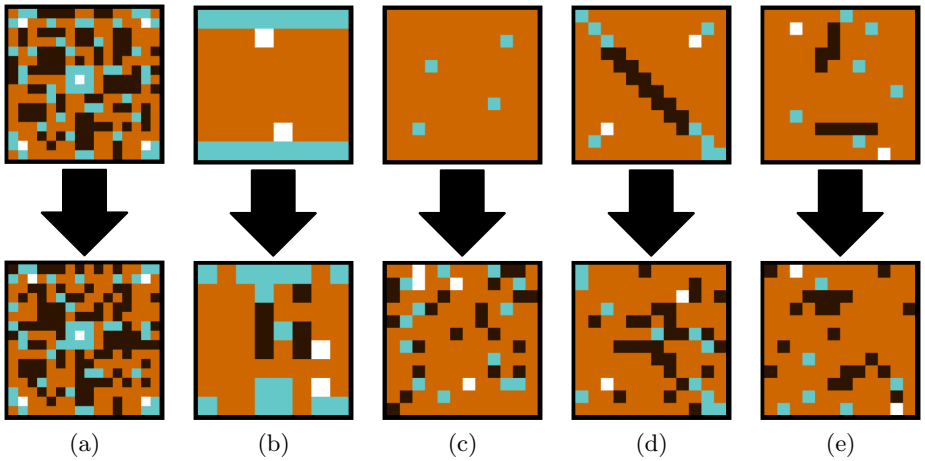


Fig. 6.2: A sample of user maps before (top) and after (bottom) a suggestion was selected, showcasing different instances where suggestions were deemed useful.

(d) instances where suggestions appeared disconnected to what the user was focusing on at that time; one user reported that "the suggestions were interesting, but it was not entirely clear how they related to what I was doing".

While there is a clear way of amending instances of type (c) and — to an extent — (b), instances of type (a) are much less likely to benefit from mixed-initiative processes (since computational input is ignored), while instances of type (d) require an elaborate model of the designer's goals, processes and preferences [140]. Addressing the instances of type (b) and (d) was a direct motivation for introducing designer models of process, preference and symmetry goals in Section 5.6.

Users often selected one map suggestion per session (in 12 sessions). In two sessions of different users, map suggestions were selected 4 times per session; since the two sessions belong to different users, it is worthwhile to investigate the reasons behind this behavior. In the first case, the user created an elaborate map (see Figure 6.2a), and then sequentially applied a map suggestion and saved the new map, thus creating four variations of their original map as a cheap shortcut for generating more content. In the second case, the user created a very simple symmetrical map (see Figure 6.2b) with all resources on two map edges, and successively used the map suggestions to add impassable regions and randomize tile placement. Overall, the selection of suggestions followed two usage patterns. In 3 sessions, map suggestions were selected early in the design process to quickly generate a map "draft" which was then edited by the designer (see Figure 6.2c); such design sessions were short, since the suggestion was already playable and did not need significant changes. In 10 sessions, map suggestions were applied as one of the last steps before saving the final map; this last step either aimed at breaking the simple authored patterns and create more organic maps (see Figure 6.2d) or to increase the score in one or more fitness dimensions (see Figure 6.2e). Further elaboration on the patterns of suggestion use for small maps is provided in Section 6.3.3.

The type of map suggestions selected by users are shown in Table 6.1. Despite the fact that novelty search generated as many suggestions as all the objective-driven GAs together, its

	# sessions	suggestions selected	suggestion's objective						
			f_{res}	f_{saf}	f_{exp}	b_{res}	b_{saf}	b_{exp}	NS
Overall	24	22	3	3	5	2	2	1	6

Table 6.1: The objectives, including novelty (NS), towards which the five users' selected suggestions were optimized.

suggestions are selected less frequently than those generated to optimize specific objectives. Novelty search was preferred by users requesting large changes in the map, usually in early stages of the design process when user-created maps were still uninteresting; e.g. Figures 6.2b, 6.2c and 6.2d feature suggestions generated via novelty search. In late stages of the design process, when the authored maps were almost final, suggestions were used to fine-tune the map and were selected based on their improvements to the fitness dimensions. Surprisingly, suggestions targeting balance were not chosen often; a likely reason is that user-created maps were often optimal in most balance dimensions because of the strict map symmetries of human designs. Suggestions which increased the f_{exp} score, on the other hand, were more preferable as they create long, winding paths which would require significant effort to create manually.

6.3 Quality of Use for Generated Suggestions

The impact of mixed-initiative design in the creative process will be studied more methodically on designer sessions with small maps, since those sessions are more concise in terms of discrete user actions taken from empty map to final map. During the user study detailed both above and in [142], seven design sessions on small maps were undertaken by multiple expert designers.

6.3.1 Qualitative Observation of Creation Paths

The creation path of each map is shown on Fig. 6.4; every map instance, before and after the user's action, is displayed sequentially from the session's start to its finish. With the exception of session 1, the patterns in user actions in the sessions shown indicate a preference towards symmetry, not only on the final map but also on the process; that is shown often in the initial placement of bases (sessions 4 and 6), resources (session 2) or impassable regions (sessions 3 and 5). In sessions 1, 2, 4 and 5, no computer-generated suggestions are chosen to replace the user's design; the designer's tendency towards symmetry is never broken (excluding session 1, which did not include symmetry at any point in the process). On the other hand, in sessions 3 and 6 the designer used a suggestion to change their current design once (at steps 7 and 24 respectively), while in session 7 the designer used the suggestions multiple times in a row (steps 19 to 22), changing their design thoroughly. In session 6, the suggestion caused a minor change in the map structure: while the diagonal symmetry of the map is broken, the suggestion does not seem to result in a better map or inspire the user to continue working on it. In session 3, the suggestion caused all necessary game elements to be added to the map (i.e. two bases and multiple resources), acting as a form of shortcut to speed up the creation process; the designer then proceeds to manually "correct" the generated suggestion by creating multiple paths between bases and balancing the placement of resources around each base. Finally, session 7 is interesting in the fact that the user created a very symmetrical initial map and

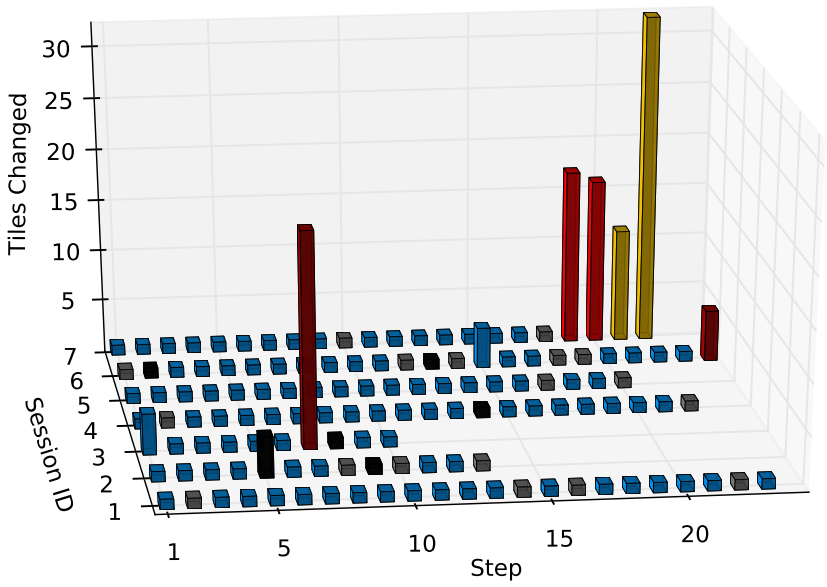


Fig. 6.3: Tiles changed on each step for the design sessions shown in Figure 6.4. Steps tagged as milestones by 2 users are shown in gray, and by 3 users in black. Steps with suggestions are shown in yellow (not tagged by multiple users), red (tagged by 2 users) and dark red (tagged by 3 users).

then proceeded to explore other possibilities via the suggestions; steps 19 and 22 saw the use of suggestions targeting novelty, which lead to big changes in the map, while steps 20 and 21 saw the use of suggestions targeting specific map qualities, which caused more subtle changes which can be considered "finetuning".

6.3.2 Quantitative Evaluation of Creation Paths

While the qualitative evaluation of the design process is quite informative, the question of quantitatively evaluating the creation path (and its creativity) remains to be answered. A mathematically defined heuristic could potentially be used to evaluate the process; in the domain of tile-based map sketches, the number of tiles changing within a step could perhaps be a good abstraction of how humans qualitatively measure map change. Fig. 6.3 displays the tiles changed from one step (i.e. one user action). As the interface of Sentient Sketchbook mostly supports "painting" a single tile, most user actions result in one tile changed; all steps of sessions 1, 4 and 5 see one tile changed per step. The designer can also paint a larger passable or impassable area, leading to a handful of tiles changing (step 5 of session 2, step 1 of session 3 and step 15 of session 6). However, only the replacement of the user's sketch with a computer-generated suggestion causes large changes in the number of tiles. For instance, 21 tiles change at step 7 of session 3, when the user chooses a computer-generated suggestion that adds bases and resources; despite the fact that the suggestion was not evolved towards visual diversity, the large change is due to the playability constraint that computer-generated maps must possess multiple bases and resources. In session 6, the generated suggestion at step 24 does not change many of the map's tiles, as is directly visible in Fig. 6.4. In session 7, generated suggestions evolved via novelty search change the map substantially (steps 19

and 22) while those generated towards optimizing a map quality less so (steps 20 and 21). The fact that novelty search makes more significant changes to maps is not surprising, since it explicitly targets tile difference in the novelty score of the genetic algorithm.

6.3.3 Human Audience

Another evaluation which combines both the subjective properties of qualitative human evaluation and quantitative, data-driven evaluations comes from the feedback of a human audience regarding the creation process. In this context, the expert users participating in the presented user survey were asked to tag which of the steps displayed in Fig. 6.4 constituted *milestones* in the creation process — i.e. enablers of lateral path change. All survey participants were asked to tag all creation paths (excluding theirs) for small maps. The user's responses were unsurprisingly varied: some designers tagged numerous steps as milestones (as high as 8.1 milestones on average per session) while others identified milestones sparsely (1.4 milestones on average per session). To the same effect, different designers arguably selected milestones based on different underlying criteria and may ascribe different meanings to the term "milestone". This study will investigate steps classified as milestones by two or more users, excluding the map's creator. Both Fig. 6.4 and Fig. 6.3 highlight which steps were classified as milestones by two or three designers (no milestones had the consensus of all four designers). While many of these milestones are common-sensical to a casual, human observer, it is interesting to note that they are not often detected by the tile change heuristic of Fig. 6.3. The steps which place the first or second base (e.g. step 2 of session 4, step 10 of session 7) or the steps which turn an asymmetrical map into a symmetrical one (e.g. step 14 of session 4, step 20 of session 5) are often identified as milestones, yet they involve a single tile change and are not identified as significant by the heuristic. However, the designer's analogical reasoning behind the pure visual representation of the strategy game level identifies that bases are significant for gameplay and that symmetrical maps result in "fair" games. Designers are thus not tagging milestones solely for their potential as visual diagrammatic lateral paths, but also as analogical diagrammatic lateral paths. Although several steps were tagged as milestones by multiple users, suggestions are particularly prevalent among them, considering their rarity. Out of the 6 computer suggestions in the sessions investigated, 4 were tagged as milestones by two or more users, out of a total of 27 milestones on 134 steps, while 2 suggestions were tagged as milestones by three users out of a total of 8 milestones on 134 steps.

It is important to note that identifying a computer generated suggestion is not particularly difficult for casual observers and users of Sentient Sketchbook alike; the tile change heuristic can also detect suggestions with a low margin of error. The fact that generated maps are not symmetrical is a tell-tale sign of a computer generated suggestion, as well as the fact that multiple tiles change in a single step. While in cases where designers were focusing on symmetrical maps these generated asymmetrical patterns did not always resonate with them, in the context of MI-CC the fact that generated suggestions do not appear similar to something a human designer would have created is not in any way detrimental. Instead, the asymmetries act as a stimulus necessary to disrupt the user's habitual pattern of thought and, to a degree, the ingrained attraction of humans to symmetry [2, 193]. By targeting map qualities such as balance, generated suggestions which are asymmetrical, yet of high-quality and balanced, disrupt preconceived notions of level designers that "fairness" between opponents in a strategy

game can only be achieved through symmetry in the levels. As can be expected, the foreignness of the generated suggestions is often rebelled against and suggestions are ignored, especially in cases where users are not necessarily open to non-linear diagrammatic lateral thinking. However, even the presentation of such suggestions while the user is editing the level can lead to a certain didactic experience in diagrammatic lateral thinking skills; as a participant of the user survey commented regarding their experience using Sentient Sketchbook, "I've started to like the tool a lot. Instead of doing something symmetrical and intricate, I started just working with some high level abstract ideas for the maps, like 'this one should be scattered' or 'in this one the players should be 3 vs. 1' or whatever. Suddenly the tool starts pushing you to places you didn't really consider."

6.4 Summary

This Chapter presented a small-scale user survey with industry experts interacting with Sentient Sketchbook. While the usability of the tool, the feedback of users, and the final artifacts created by the users were analyzed and reported, this Chapter focused on the impact of generated suggestions on the design process since those constitute the main proactive initiative of the computational creator. A quantitative evaluation of the number of suggestions chosen by users to replace their own designs — and the type of objective or novelty those were evolved towards — provided some quantitative evaluation of their usefulness, although the small number of users makes general conclusions from such an analysis impossible. A more in-depth analysis of the quality of use of suggestions within the users' creative paths provided insights on the effect of computational initiative and how it differs from other "creative milestones" identified by the same users. The following Chapters will evaluate how the algorithms that generated the suggestions of Sentient Sketchbook perform in different settings and types of game levels.

Fig. 6.4: Complete creation paths of seven design sessions on small maps, displaying all steps taken. Each step corresponds to a user's interaction, and causes the transition from the previous map to the next. Steps where a generated suggestion replaced the user's sketch are shown as red arrows. Steps tagged as milestones in the creation process by two designers are indicated with a white circle above the arrow; those tagged by three designers are indicated with a black circle.

Chapter 7

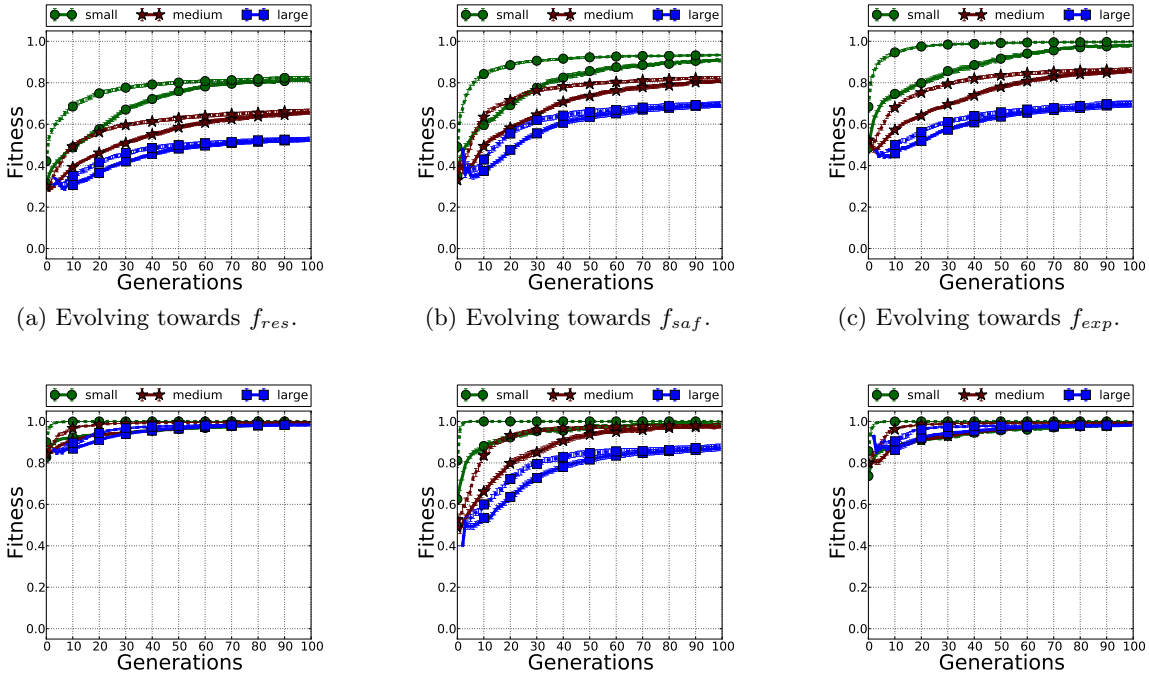
Constrained Optimization of Map Sketches

As a mixed-initiative tool, Sentient Sketchbook (described in Chapter 5) requires that the computational creator is able to generate suggestions for the human designer. Constrained optimization of specific gameplay objectives is used to generate a portion of these suggestions. The fitness dimensions which evaluate the quality of a map sketch were designed to be as generic and parametrizable as possible, so that more than one type of game level can be generated. This Chapter tests how constrained optimization performs in three different types of game levels: strategy game maps in Section 7.1, dungeons for roguelike games in Section 7.2.1 and first-person shooter arenas in Section 7.2.2. Such different game genres necessitate different special tiles (such as player bases, treasures, or weapon caches) but also different formulations of the quality evaluations from Section 5.2.1; by testing the FI-2pop GA on such diverse level types and observing the generated results, some conclusions on the generalizability of the evaluations can be drawn. Moreover, using strategy game levels as the most studied instance of map sketching within Sentient Sketchbook, the impact of different parameter setups such as the choice of genetic operators and the use of offspring boost is tested in Sections 7.1.3 and 7.1.4 respectively; comparisons within those sections test for significance via standard two-tailed t-tests (assuming unequal variances), with $p < 0.05$.

7.1 Strategy Game Levels

Section 5.3 has already presented the format of strategy game map sketches, which consist of impassable tiles, passable tiles, bases and resources. Moreover, the evaluations of resource safety (f_{res}) and its balance (b_{res}), base area safety (f_{saf}) and its balance (b_{saf}) and base exploration (f_{exp}) and its balance (b_{exp}) have been covered, along with the formulas used for calculating them. As presented in Chapter 6, these evaluation functions were deemed highly useful by expert game developers in a user study in which user-authored strategy maps were being evaluated. Combining all these evaluations will result in high-quality maps favoring slow, defensive gameplay.

To showcase the algorithms' behavior in different constrained search spaces, experiments in this Chapter are performed on small, medium and large strategy game maps. Small maps have 64 tiles (laid on a grid of 8 by 8 tiles), and must contain 2 bases and between 4 and 10 resources. The few bases and resources and small size makes the creation of feasible individuals likely, even by random generation: from 10^6 randomly generated maps, $3{\cdot}10^4$ are feasible. Large maps have 256 tiles (laid on a grid of 16 by 16 tiles), and must contain between 2 and 10 bases and between 4 and 30 resources. The large map size increases the likelihood that paths between bases and resources will be blocked, especially when either bases or resources are



(a) Evolving towards f_{res}. (b) Evolving towards f_{saf}. (c) Evolving towards f_{exp}.

(d) Evolving towards b_{res}. (e) Evolving towards b_{saf}. (f) Evolving towards b_{exp}.

Fig. 7.1: Evolving towards a single objective for strategy game levels. Values are averaged across 50 runs. The dotted line displays the population's maximum fitness and the solid line the population's average fitness.

numerous; from 10^6 randomly generated large maps, only 10 are feasible. Medium maps have 144 tiles (laid on a grid of 12 by 12 tiles), and must contain 4 bases and between 8 and 20 resources. The map size should place medium maps somewhere between small and large maps in terms of ease of satisfying constraints, although medium maps have a minimum of 12 tiles that must be connected (bases and resources) versus the minimum of 6 tiles for small and large maps: from 10^6 randomly generated medium maps, 852 are feasible.

7.1.1 Single Objective

Using a single fitness dimension as the objective function, the genetic algorithm aims to optimize a single gameplay feature; the generated maps are, thus, expected to be one-sided and lack the necessary features for competitive strategy play. Figure 7.1 shows the evolutionary progress of the maximum and average fitness in the population; displayed values are averaged from 50 independent runs, with the standard error shown as error bars.

Observing the progress of optimization across map sizes and fitness dimensions, we observe that the maximum fitness quickly stabilizes to a high score, usually within the first 20 generations. This behavior is most prominent in small maps, which are easier to optimize due to the smaller size of the genotypes as well as the small map size and fewer bases and resources. The larger map sizes face more challenges in finding an optimal value, to a degree due to the larger

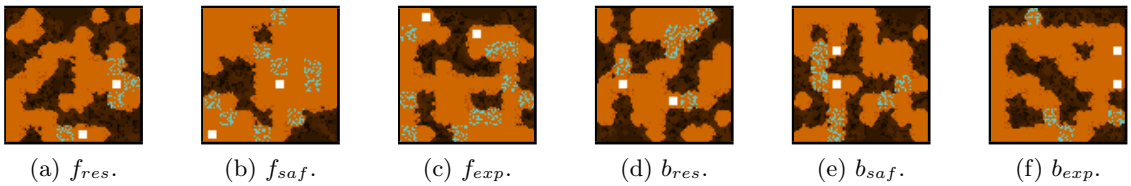
genotypes (large maps have four times the number of chromosomes in the genotype than small maps) but most importantly due to the highly constrained search spaces which they have to explore. With both medium and large maps, the chances of feasible parents creating feasible offspring is far lower than that of small maps, which makes optimization slower. Additionally, the initial population is very unlikely to contain even a single feasible individual in medium and especially in large maps; this causes the FI-2pop GA to explore infeasible space for several generations before a feasible individual is found so that the feasible population can begin its optimization. This can be gleaned from Fig. 7.1, where for large maps only a few of the runs have feasible individuals before the 10th generation and thus cause the apparent fluctuation of fitness values due to averaging of different numbers of runs per generation.

Observing the differences between fitness dimensions, we observe that optimal fitness scores for balance (b_{res}, b_{saf}, b_{exp}) are easily attainable, since high-scoring individuals exist even in the random initial populations. Thus, evolution quickly finds optimal solutions for these fitness dimensions within few generations, and there are few differences in fitness score between the best individual in the population and the average fitness score of the population. Somewhat of an exception is b_{saf}, which for larger map sizes is slower to find a good solution, and the maximum fitness score reached for large maps is not optimal. Fitness dimensions of f_{res}, f_{saf}, f_{exp}, on the other hand, are slower to optimize and reach lower scores than their balance counterparts. The lower scores are as much caused by the formulation of the fitness function as they are due to the optimization behavior, however. In particular, f_{res} appears as the hardest to optimize mainly due to how the safety metric is calculated: based on (5.1), $s_{t,i}$ cannot reach its optimal value (1.0), but approaches it if $d_{t,j} \gg d_{t,i}$ for all bases $j \neq i$. Since the resources can't overlap with bases and thus the minimal $d_{t,j}$ for any base j will be 1, even in the best circumstances $s_{t,i}$ and thus f_{res} will be considerably lower than 1.0. Similarly, f_{saf} cannot reach optimal values since there will be at least one tile in the map at equal distance from both bases (and thus not safe).

The highest scoring final individual among the 50 runs for each fitness dimension is shown in Fig. 7.2 for small, medium and large maps. The maps are represented as complete levels for better visual identification.

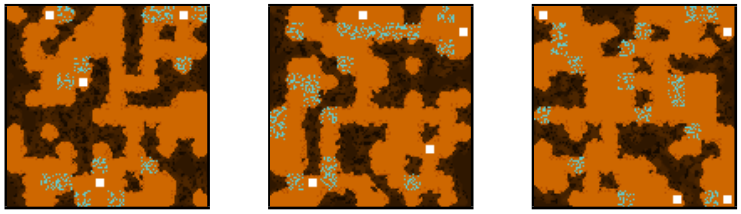
Observing the best individuals, it is obvious that optimal maps in balance dimensions (b_{res}, b_{saf}, b_{exp}) have bases relatively close to each other; this is most obvious in the large maps of Fig. 7.2p and 7.2q and the medium maps of Fig. 7.2l. The reason for this behavior is that maps with nearby bases have these bases equally close to the (same) resources (b_{res}), have almost no area around them that they can control since there is always at least one more base contending the area (b_{saf}) and it is equally easy to find each other (b_{exp}) as is the case in Fig. 7.2f. Essentially, the deviation among bases is low because $s_{t,i}$ and E_i are (equally) low for all bases i. With bases so close to each other, the gameplay of such maps would involve little strategic choice and would limit itself to who would start attacking their enemies first.

On the other hand, maps optimized for the fitness dimensions of f_{res}, f_{saf}, f_{exp} have bases which are usually far away from each other (despite exceptions such as Fig. 7.2m which has a relatively low score in f_{res} in the first place). The appearance of such optimal maps is due to the fact that both $s_{t,i}$ and E_i reward a high distance between bases, either implicitly ($s_{t,i}$) or explicitly (E_i). The best maps in f_{res} have resources adjacent to bases, which is evident in Fig. 7.2a. However, the number of resources adjacent to each base is obviously unbalanced: in the map of Fig. 7.2a one base has three adjacent resources while the other has only one

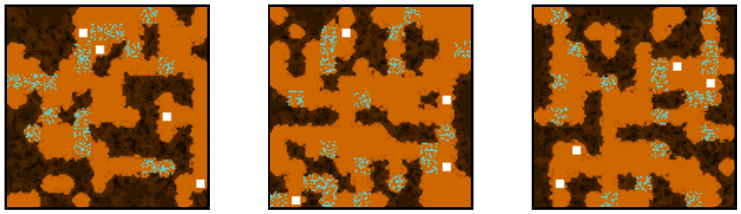


(a) f_{res}. (b) f_{saf}. (c) f_{exp}. (d) b_{res}. (e) b_{saf}. (f) b_{exp}.

Small Maps

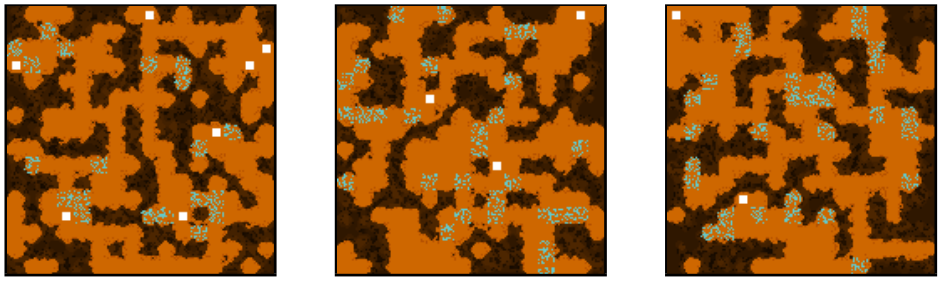


(g) f_{res}. (h) f_{saf}. (i) f_{exp}.

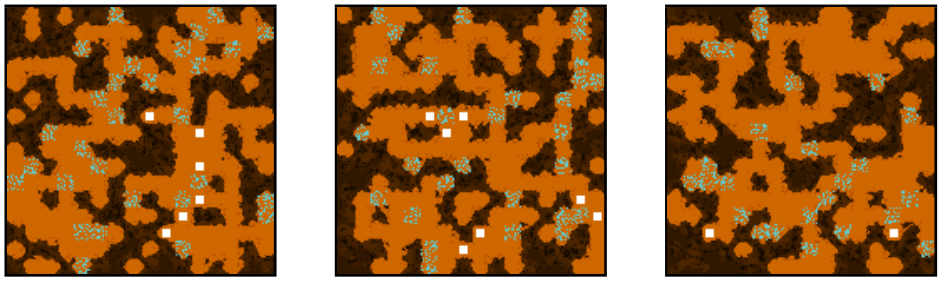


(j) b_{res}. (k) b_{saf}. (l) b_{exp}.

Medium Maps



(m) f_{res}. (n) f_{saf}. (o) f_{exp}.



(p) b_{res}. (q) b_{saf}. (r) b_{exp}.

Large Maps

Fig. 7.2: Best scoring final strategy game maps evolved for a single objective, across 50 runs.

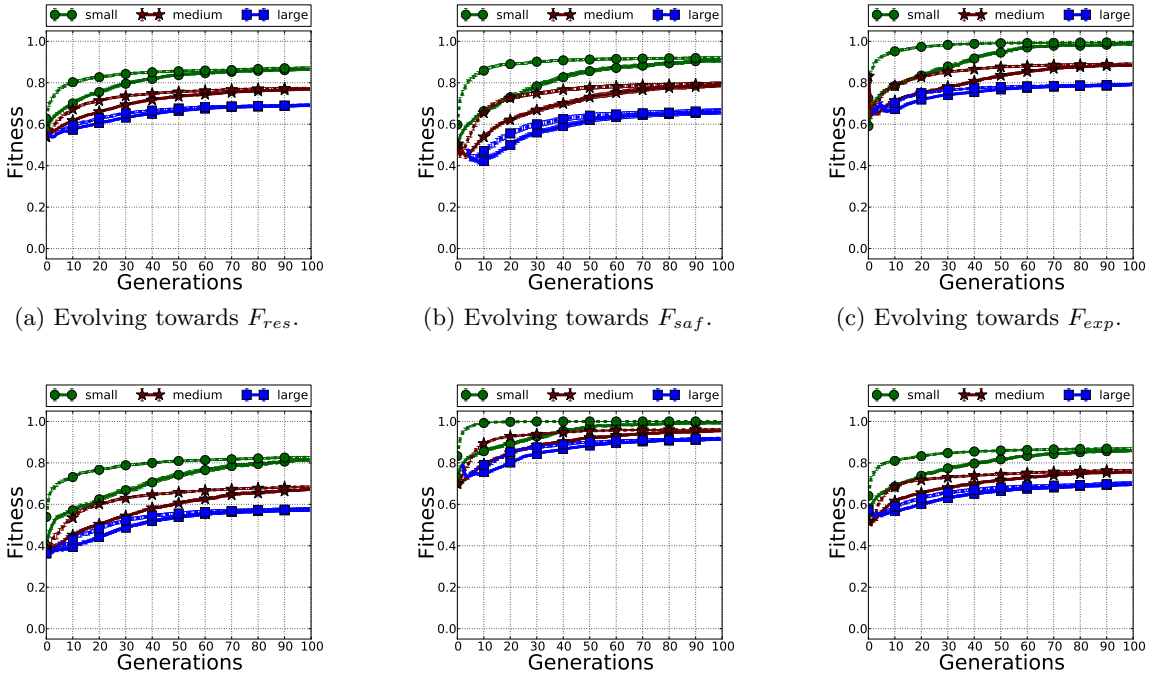
resource, and in Fig. 7.2m three bases have no nearby resources at all. The best maps in f_{saf} have bases neatly hidden away from other bases, such as the bottom left base in Fig. 7.2b or all bases in Fig. 7.2h; however, it is also likely that one base can control more area than others as is evident in Fig. 7.2b where the base in the middle controls most of the map. Finally, the best maps in f_{exp} have bases far away from each other (more so than maps optimized for other dimensions); however, since f_{exp} does not account for resources, optimal maps in this dimension may have few or unbalanced safe resources.

7.1.2 Multiple Objectives

Experiments in Section 7.1.1 showed that maps optimized for a single dimension usually have interesting traits, but lack the necessary features needed for competitive play. Combining multiple fitness dimensions into a weighted sum and using it as the objective function for the genetic algorithm is expected to generate more appropriate maps. While there are many sophisticated methods for handling multiobjective optimization [42], this thesis follows a straightforward — if somewhat naive — approach of aggregating objectives into a single fitness score; the potential of more elaborate approaches is discussed in Section 11.2.1. Experiments in this section will assess the combined optimization of two or more fitness dimensions; the most intuitive combinations are the following:

- $F_{res} = \frac{1}{2}f_{res} + \frac{1}{2}b_{res}$.
- $F_{saf} = \frac{1}{2}f_{saf} + \frac{1}{2}b_{saf}$.
- $F_{exp} = \frac{1}{2}f_{exp} + \frac{1}{2}b_{exp}$.
- $F_{bal} = \frac{1}{2}f_{saf} + \frac{1}{2}b_{saf}$.
- $F_{qual} = \frac{1}{3}f_{res} + \frac{1}{3}f_{saf} + \frac{1}{3}f_{exp}$.
- $F_{bal} = \frac{1}{3}f_{res} + \frac{1}{3}f_{saf} + \frac{1}{3}f_{exp}$.
- $F_{all} = \frac{1}{6}f_{res} + \frac{1}{6}f_{saf} + \frac{1}{6}f_{exp} + \frac{1}{6}b_{res} + \frac{1}{6}b_{saf} + \frac{1}{6}b_{exp}$.

Figure 7.3 displays the evolutionary progress of the maximum and average fitness in the population; displayed values are averaged from 50 independent runs, with the standard error shown as error bars. The optimization behavior is not much different than that for single objectives in Fig. 7.1, with maximum fitness stabilizing within a few generations and small maps reaching higher values than medium and large maps, in that order. Unsurprisingly, the best fitness scores reached in each combination are highly contingent on the best scores of the contributing fitness dimensions as those were found in Section 7.1.1: F_{res} has lower scores due to the contribution of f_{res} which had difficulties in reaching high scores in Fig. 7.1a, while F_{saf} is slower to optimize (at least in terms of average fitness) due to b_{saf} being similarly slower to optimize than other dimensions of balance in Fig. 7.1. By the same account, optimizing F_{qual} is more difficult than optimizing F_{bal} since each of the contributing fitness dimensions of the former (f_{res}, f_{saf}, f_{exp}) are slower to optimize and reach lower values than balance dimensions; yet another reason is that optimal maps in balance dimensions share the same features for all of b_{res}, b_{saf} and b_{exp} as will be showcased below.

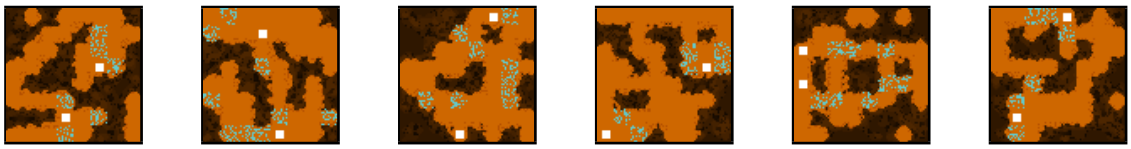


(a) Evolving towards F_{res}. (b) Evolving towards F_{saf}. (c) Evolving towards F_{exp}.

(d) Evolving towards F_{qual}. (e) Evolving towards F_{bal}. (f) Evolving towards F_{all}.

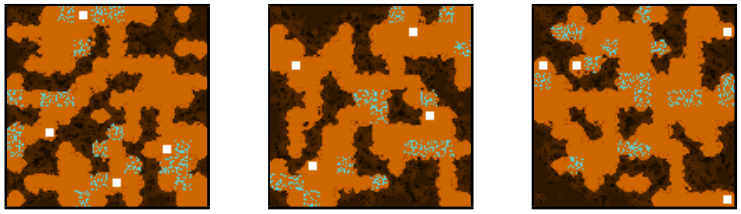
Fig. 7.3: Evolving towards multiple objectives aggregated into a single fitness for strategy game levels. Values are averaged across 50 runs. The dotted line displays the population's maximum fitness and the solid line the population's average fitness.

The highest scoring final individual among the 50 runs for the different combinations of fitnesses is shown in Fig. 7.4 for small, medium and large maps. As expected, optimizing for F_{bal} results in bases placed close to each other (see Fig. 7.4q) so that they are equally easy to discover each other and they have an equal but low amount of safe resources and controllable area; the same features were prominent in most maps optimized for each of the balance dimensions in Fig. 7.2. However, maps optimized for both a strategic quality and its balance are particularly well-formed. Maps optimized for F_{res} have bases far away from each other, with each base usually having an equal number of nearby resources: the map of Fig. 7.4a has 3 resources near each base while the map of Fig. 7.4g has 4 resources near each base (excluding the leftmost base which has 5 resources). Maps optimized for F_{saf} have neatly partitioned map segments via impassable tiles and chokepoints, which are easily controllable by one base: the division of the map into such segments is quite fair, as in the case of Fig. 7.4b where one base controls the top half of the map and the other controls the bottom half or Fig. 7.4n where one base controls the top right map quadrant and the other controls the bottom left map quadrant. F_{exp} similarly has bases far away from each other and hidden behind extensive impassable regions; the impact of b_{exp} is difficult to ascertain by visually inspecting the best maps, especially with multiple bases as in Fig. 7.4o where the bases on the top right and bottom right corners are more secluded and difficult to discover than the others. Finally, maps optimized for F_{all} have most of the significant properties of competitive strategy game levels, with equal distributions of resources, distant bases and easily controllable areas around each

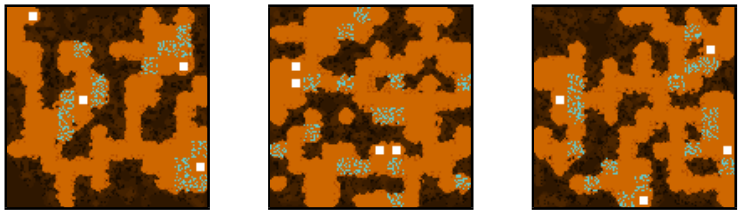


(a) F_{res}. (b) F_{saf}. (c) F_{exp}. (d) F_{qual}. (e) F_{bal}. (f) F_{all}.

Small Maps

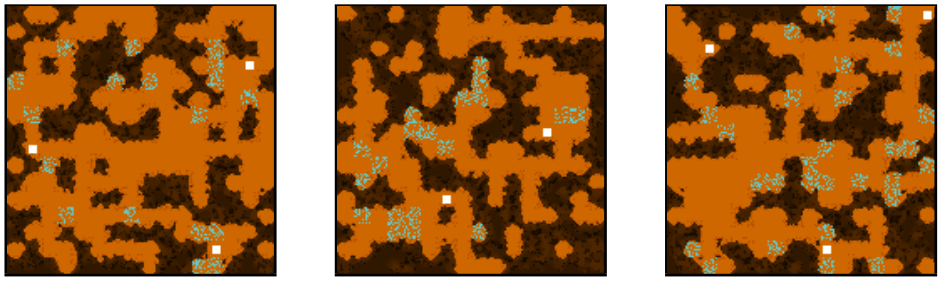


(g) F_{res}. (h) F_{saf}. (i) F_{exp}.

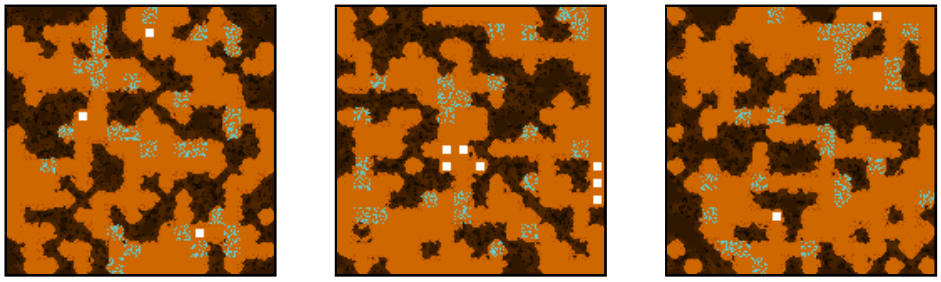


(j) F_{qual}. (k) F_{bal}. (l) F_{all}.

Medium Maps



(m) F_{res}. (n) F_{saf}. (o) F_{exp}.



(p) F_{qual}. (q) F_{bal}. (r) F_{all}.

Large Maps

Fig. 7.4: Best scoring final strategy game maps evolved for multiple objectives, across 50 runs.

Obj.	f_{res}	f_{saf}	f_{exp}	b_{res}	b_{saf}	b_{exp}
Small Maps						
Single	0.826 (0.005)	0.934 (0.003)	0.998 (0.001)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)
F_{res}	0.756 (0.008)	—	—	0.989 (0.002)	—	—
F_{saf}	—	0.842 (0.010)	—	—	1.000 (0.000)	—
F_{exp}	—	—	0.990 (0.003)	—	—	1.000 (0.000)
F_{qual}	0.755 (0.009)	0.844 (0.005)	0.882 (0.010)	—	—	—
F_{bal}	—	—	—	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)
F_{all}	0.655 (0.013)	0.779 (0.011)	0.832 (0.011)	0.977 (0.004)	0.999 (0.001)	0.977 (0.004)
Medium Maps						
Single	0.666 (0.007)	0.825 (0.006)	0.864 (0.008)	0.996 (0.001)	0.983 (0.006)	0.993 (0.001)
F_{res}	0.593 (0.009)	—	—	0.954 (0.005)	—	—
F_{saf}	—	0.620 (0.016)	—	—	0.973 (0.006)	—
F_{exp}	—	—	0.809 (0.010)	—	—	0.973 (0.003)
F_{qual}	0.602 (0.009)	0.769 (0.007)	0.683 (0.008)	—	—	—
F_{bal}	—	—	—	0.961 (0.004)	0.969 (0.006)	0.955 (0.004)
F_{all}	0.490 (0.012)	0.633 (0.012)	0.662 (0.009)	0.921 (0.007)	0.936 (0.009)	0.944 (0.003)
Large Maps						
Single	0.529 (0.009)	0.697 (0.011)	0.699 (0.014)	0.984 (0.002)	0.879 (0.011)	0.984 (0.001)
F_{res}	0.455 (0.008)	—	—	0.931 (0.006)	—	—
F_{saf}	—	0.492 (0.019)	—	—	0.837 (0.016)	—
F_{exp}	—	—	0.620 (0.010)	—	—	0.966 (0.002)
F_{qual}	0.509 (0.010)	0.655 (0.011)	0.566 (0.009)	—	—	—
F_{bal}	—	—	—	0.936 (0.005)	0.869 (0.012)	0.953 (0.005)
F_{all}	0.428 (0.009)	0.551 (0.014)	0.593 (0.013)	0.901 (0.008)	0.784 (0.021)	0.953 (0.004)

Table 7.1: Contributing fitness dimensions' scores for the best final strategy game levels. Values displayed are averaged across 50 runs, with standard error shown in parentheses.

base; excluding several contested (unsafe) resources in the large map of Fig. 7.4r, each of the six fitness dimensions seems to contribute in creating a high-quality final best map.

In order to ascertain the impact of each contributing fitness dimension in the best maps, Table 7.1 displays the fitness scores of the best final maps. The map of each run with the highest score in the aggregated fitness is considered, averaging the score of each fitness dimension across the 50 runs of each experiment. The highest fitness scores of single objective experiments are included for comparative purposes. Unsurprisingly, the aggregation of multiple fitness dimensions into a weighted sum results in lower scores in each dimension than when optimizing a single dimension; the drop is more prominent for dimensions of strategic quality (f_{res}, f_{saf}, f_{exp}) than dimensions of balance, which remain near-optimal even when all fitness dimensions are optimized simultaneously (F_{all}).

When optimizing multiple objectives, it was expected that the aggregation via weighted sum of multiple fitness dimensions into a single fitness score would hinder the ability of the FI-2pop GA to optimize all dimensions simultaneously. Due to the fact that the fitness dimensions are largely not conflicting, however, even with this arguably naive approach the FI-2pop GA has shown potential at optimizing two or more objectives to a good degree — although optimization was slower than for single objectives. The fitness dimensions of strategy game levels could be a reason for this relatively robust behavior: f_{res}, f_{saf} and f_{exp} all favor maps with distant bases, and although b_{res}, b_{saf} and b_{exp} may create maps with nearby bases

	Avg. Fitness		Max Fitness		Feasible Individuals	
	Rec.	Mutation	Rec.	Mutation	Rec.	Mutation
Small Maps						
f_{res}	**0.812 (0.006)**	0.592 (0.008)	0.826 (0.005)	**0.850 (0.004)**	**65.8 (2.1)**	41.8 (0.7)
f_{saf}	**0.907 (0.007)**	0.653 (0.006)	0.934 (0.003)	**0.943 (0.002)**	**80.2 (1.9)**	40.2 (0.8)
f_{exp}	**0.981 (0.005)**	0.834 (0.007)	0.998 (0.001)	**1.000 (0.000)**	**77.0 (1.8)**	40.0 (0.7)
b_{res}	**0.985 (0.004)**	0.947 (0.002)	1.000 (0.000)	1.000 (0.000)	**69.4 (1.8)**	42.4 (0.7)
b_{saf}	**0.989 (0.002)**	0.895 (0.007)	1.000 (0.000)	1.000 (0.000)	**71.3 (1.7)**	43.2 (0.7)
b_{exp}	**0.984 (0.005)**	0.872 (0.005)	1.000 (0.000)	1.000 (0.000)	**69.7 (1.8)**	42.9 (0.8)
F_{all}	**0.858 (0.005)**	0.723 (0.003)	0.870 (0.004)	**0.880 (0.002)**	**75.7 (1.8)**	40.4 (0.6)
Medium Maps						
f_{res}	**0.656 (0.008)**	0.519 (0.007)	0.666 (0.007)	**0.690 (0.007)**	**59.7 (1.8)**	25.4 (0.7)
f_{saf}	**0.807 (0.007)**	0.605 (0.007)	0.825 (0.006)	**0.861 (0.004)**	**61.6 (1.7)**	25.2 (0.7)
f_{exp}	**0.855 (0.008)**	0.680 (0.007)	0.864 (0.008)	0.858 (0.007)	**68.3 (2.0)**	23.3 (0.7)
b_{res}	**0.991 (0.002)**	0.935 (0.002)	0.996 (0.001)	0.998 (0.000)	**69.3 (1.9)**	29.2 (0.8)
b_{saf}	**0.975 (0.007)**	0.740 (0.013)	0.983 (0.006)	**0.998 (0.001)**	**65.9 (1.8)**	29.2 (0.8)
b_{exp}	**0.988 (0.002)**	0.894 (0.003)	0.993 (0.001)	**0.996 (0.000)**	**66.7 (2.0)**	27.8 (0.7)
F_{all}	**0.755 (0.004)**	0.672 (0.004)	0.764 (0.004)	**0.774 (0.003)**	**65.3 (1.9)**	26.9 (0.8)
Large Maps						
f_{res}	**0.525 (0.009)**	0.451 (0.006)	0.529 (0.009)	**0.568 (0.005)**	**55.4 (1.2)**	12.8 (0.7)
f_{saf}	**0.692 (0.011)**	0.515 (0.007)	0.697 (0.011)	0.698 (0.005)	**60.9 (1.9)**	12.6 (0.6)
f_{exp}	**0.696 (0.014)**	0.546 (0.008)	**0.699 (0.014)**	0.648 (0.006)	**62.8 (1.7)**	13.6 (0.5)
b_{res}	**0.982 (0.002)**	0.920 (0.003)	**0.984 (0.002)**	0.975 (0.001)	**63.4 (1.7)**	16.4 (0.6)
b_{saf}	**0.870 (0.012)**	0.668 (0.008)	0.879 (0.011)	0.888 (0.008)	**63.0 (1.8)**	14.6 (0.7)
b_{exp}	**0.982 (0.002)**	0.918 (0.004)	0.984 (0.001)	0.986 (0.001)	**62.9 (1.8)**	16.6 (0.7)
F_{all}	**0.697 (0.007)**	0.625 (0.004)	**0.702 (0.008)**	0.678 (0.004)	**62.0 (1.8)**	13.7 (0.6)

Table 7.2: Performance metrics for single objective and multiobjective optimization via 2-point crossover and 1% chance of mutation (Rec. column) and only mutation (Mutation column) for strategy game levels. Results are averaged across 50 runs, with standard error in parentheses. Significantly higher values for each metric are displayed in bold.

when optimized by themselves (see Fig. 7.2), optimizing a balance and a fitness dimension simultaneously showed that there is little tradeoff between e.g. b_{res} and f_{res}. With different objectives (e.g. minimizing f_{saf} while maximizing f_{exp}) or in other game genres, objectives may become conflicting which causes aggregated fitness methods to underperform; in those cases multi-objective optimization [42] would be necessary. The issues of the FI-2pop GA with multiple objectives are elaborated in Section 11.1.1 and improvements are suggested in Section 11.2.1.

7.1.3 Impact of Genetic Operators

In all experiments described in this Chapter as well as within Sentient Sketchbook, map sketches are evolved by recombining two parents, with a 1% chance of mutating their offspring. In order to ascertain the effect of mutation on the process of optimization, this Section will compare the recombination scheme (with 1% chance of offspring mutation) with a mutation-only scheme where one parent is selected (via fitness-proportionate roulette-wheel selection) and mutated to produce a single offspring; the same parent can be selected multiple times, and thus can create multiple offspring.

Table 7.2 displays the average fitness of the population, the fitness of the best feasible individual in the population, and the number of feasible individuals after 100 generations. Results are from single-objective optimization processes, as well as for the multiobjective optimization of F_{all} which aggregates all fitnesses into a weighted sum. Observing the maximum fitness of the population, mutation seems to achieve higher values than recombination for small and medium maps; however, the differences between scores are not very pronounced as they are in the second decimal of the fitness score. For large maps, the differences in maximum fitness between mutation and recombination are even less pronounced since in some cases recombination reaches higher maximum fitness. On the other hand, the average fitness of the population when mutation is applied is much lower than when recombination is applied. Coupled with the findings regarding maximum fitness, it can be surmised that evolution via recombination runs the risk of premature convergence, which drives most of the population towards the same solution. Observing the number of feasible individuals, the final feasible population is smaller with mutation than with recombination regardless of map size. From the results, it is obvious that applying mutation to a feasible individual is more likely to render it infeasible than when recombining it with another individual (which is guaranteed to be feasible); this is likely due to the fact that mutation can add impassable tiles to a map, which can block a previously passable path. This finding again points to the fact that mutation has a chance of avoiding premature convergence but at the same time can be particularly destructive and generate many infeasible individuals. This finding is obviously regarding the mutation operator used for Sentient Sketchbook, which has a high chance of adding (or removing) impassable tiles; removing this particular tile conversion mechanism or changing its probability could make mutation less destructive. With the current mutation in place, the small computational budget and small population of Sentient Sketchbook's real-time suggestions generators make recombination preferable for its larger ratio of feasible individuals.

7.1.4 Impact of Offspring Boost

In all experiments described in this chapter as well as within Sentient Sketchbook, map sketches are evolved using the offspring boost mechanism in place. In order to ascertain the effect of the offspring boost on the process of optimization, this Section will compare the final artifacts as well as the optimization progress of runs with and and without the offspring boost. Recombination with 1% mutation is applied on both experiments, all three map sizes are tested, and resulting data is averaged from 50 runs with and without the offspring boost.

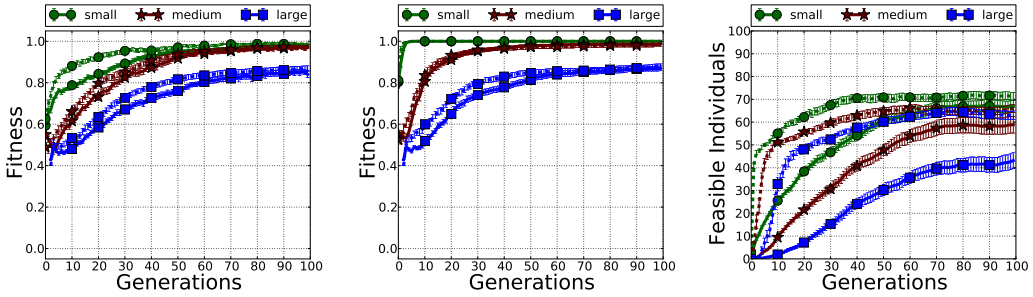
Table 7.3 displays the average fitness of the population, the fitness of the best feasible individual in the population after 100 generations, and the number of feasible individuals after 100 generations. Results are from single-objective optimization processes, as well as the F_{all} multiobjective optimization, which aggregates all fitnesses into a weighted sum. Observing Table 7.3, the offspring boost does not seem to significantly increase the number of feasible individuals in the final population except in the highly constrained search spaces of large maps. This behavior is surprising, considering the fact that the offspring boost directly targets the increase of feasible offspring. In small and medium maps, the search of the infeasible population towards minimizing infeasibility seems to eventually lead to feasible solutions even without the offspring boost. Moreover, after 100 generations there is not much difference in terms of average or maximum fitness between algorithms with or without the offspring boost.

	Avg. Fitness		Max Fitness		Feasible Individuals	
	with Boost	no Boost	with Boost	no Boost	with Boost	no Boost
Small Maps						
f_{res}	0.812 (0.006)	0.809 (0.007)	0.826 (0.005)	0.820 (0.005)	**65.8 (2.1)**	56.4 (2.5)
f_{saf}	0.907 (0.007)	0.918 (0.004)	0.934 (0.003)	0.932 (0.003)	80.2 (1.9)	79.6 (1.8)
f_{exp}	0.981 (0.005)	0.973 (0.006)	0.998 (0.001)	0.999 (0.001)	77.0 (1.8)	77.2 (1.9)
b_{res}	0.985 (0.004)	0.984 (0.003)	1.000 (0.000)	1.000 (0.000)	69.4 (1.8)	63.5 (2.4)
b_{saf}	**0.989 (0.002)**	0.985 (0.004)	1.000 (0.000)	1.000 (0.000)	71.3 (1.7)	66.9 (2.6)
b_{exp}	0.984 (0.005)	0.986 (0.004)	1.000 (0.000)	1.000 (0.000)	69.7 (1.8)	71.6 (2.0)
F_{all}	0.858 (0.005)	0.860 (0.003)	0.870 (0.004)	0.870 (0.003)	75.7 (1.8)	74.4 (1.8)
Medium Maps						
f_{res}	0.656 (0.008)	0.636 (0.010)	0.666 (0.007)	0.653 (0.008)	**59.7 (1.8)**	47.2 (3.1)
f_{saf}	0.807 (0.007)	0.799 (0.008)	0.825 (0.006)	0.817 (0.007)	61.6 (1.7)	56.9 (2.8)
f_{exp}	0.855 (0.008)	0.851 (0.006)	0.864 (0.008)	0.868 (0.007)	**68.3 (2.0)**	58.4 (3.1)
b_{res}	0.991 (0.002)	0.990 (0.002)	0.996 (0.001)	0.996 (0.000)	69.3 (1.9)	68.1 (2.8)
b_{saf}	0.975 (0.007)	0.970 (0.006)	0.983 (0.006)	0.988 (0.004)	65.9 (1.8)	58.9 (3.6)
b_{exp}	0.988 (0.002)	0.986 (0.002)	0.993 (0.001)	0.995 (0.001)	66.7 (2.0)	64.5 (2.3)
F_{all}	0.755 (0.004)	0.757 (0.004)	0.764 (0.004)	0.763 (0.004)	65.3 (1.9)	58.9 (3.0)
Large Maps						
f_{res}	0.525 (0.009)	0.523 (0.010)	0.529 (0.009)	0.529 (0.011)	**55.4 (1.2)**	41.6 (4.0)
f_{saf}	0.692 (0.011)	0.681 (0.010)	0.697 (0.011)	0.688 (0.010)	**60.9 (1.9)**	46.2 (3.6)
f_{exp}	0.696 (0.014)	0.726 (0.016)	0.699 (0.014)	0.738 (0.017)	**62.8 (1.7)**	54.8 (3.2)
b_{res}	0.982 (0.002)	0.979 (0.003)	0.984 (0.002)	0.983 (0.002)	**63.4 (1.7)**	45.8 (3.5)
b_{saf}	0.870 (0.012)	0.840 (0.015)	0.879 (0.011)	0.869 (0.012)	**63.0 (1.8)**	43.1 (3.4)
b_{exp}	**0.982 (0.002)**	0.971 (0.003)	0.984 (0.001)	0.982 (0.001)	**62.9 (1.8)**	41.2 (3.3)
F_{all}	0.697 (0.007)	0.697 (0.005)	0.702 (0.008)	0.701 (0.005)	**62.0 (1.8)**	51.4 (2.8)

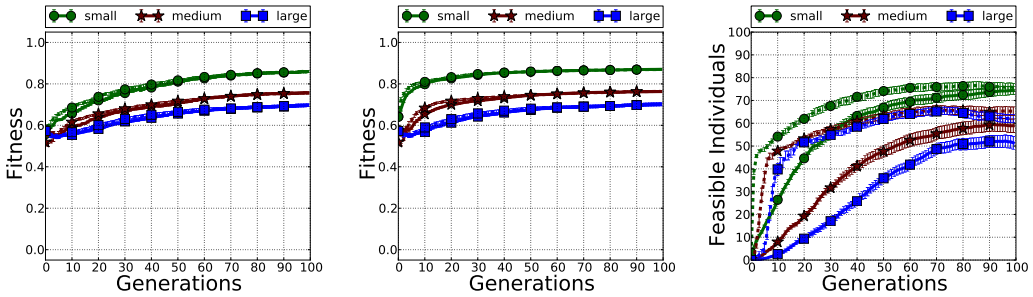
Table 7.3: Performance metrics with and without the offspring boost mechanism, for single objective and multi-objective optimization of strategy game levels. Results are averaged across 50 runs, with standard error in parentheses. Significantly higher values for each metric are displayed in bold.

Use of the offspring boost leads to slightly higher average fitnesses, although this is significant only in 2 cases; there is no significant difference in maximum fitness whatsoever, and some experiments seem to favor the offspring boost while others do not (e.g. the maximum fitness of medium maps).

While the final results of an evolutionary run after 100 generations of optimization do not seem particularly different with the introduction of the offspring boost, the impact of the offspring boost may be more obvious during the progress of evolution. Fig. 7.5 shows the two most representative experiments among those in Table 7.3, showing the progress of optimization without the offspring boost (solid line) compared with optimization with the offspring boost (dotted line). It is obvious that for both experiments (b_{saf} and F_{all}) the number of feasible individuals increases dramatically with the offspring boost in the early stages of evolution. For small maps, the number of feasible individuals reaches 50% of the total population within the first 2-3 generations with the offspring boost; without the offspring boost, the 50% threshold for feasible individuals is reached after 35 generations for b_{saf} (Fig. 7.5c) and after 25 generations for F_{all} (Fig. 7.5c). Since the offspring boost does not affect when the first feasible individual is discovered, large maps may require 10 generations before the first feasible large map sketch is discovered; after that point, however, the offspring boost immediately increases the size of the feasible population. Offspring of feasible individuals are still less likely to be feasible with



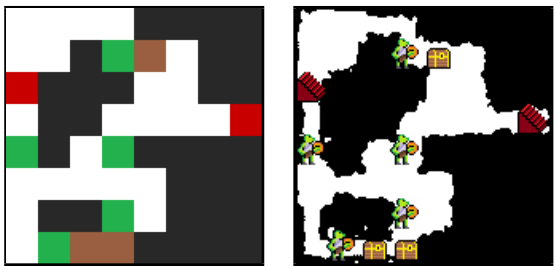
(a) Avg. Fitness for b_{saf}. (b) Max. Fitness for b_{saf}. (c) Feasible individuals for b_{saf}.



(d) Avg. Fitness for F_{all}. (e) Max. Fitness for F_{all}. (f) Feasible individuals for F_{all}.

Fig. 7.5: Evolving towards b_{saf} (Fig. 7.5a–7.5c) or F_{all} (Fig. 7.5d–7.5f) with the offspring boost (dotted line) and without the offspring boost (solid line). Values are averaged across 50 runs.

large maps than with small maps, which explains the lower number of feasible individuals for large maps compared to small maps, even with the offspring boost. Despite the obvious differences in terms of feasible individuals between runs with offspring boost and without it, the differences are not that obvious in terms of fitness optimization progress. For F_{all}, Fig. 7.5e and 7.5d show only a minor increase in terms of maximum and average fitness respectively due to the offspring boost, and only in the earliest generations; even in cases where this increase is significant, it seems minor compared to the obvious differences in feasible populations. For b_{saf}, the offspring boost seems to speed up the optimization progress since the average fitness of Fig. 7.5a and to a lesser extent the maximum fitness of Fig. 7.5b is higher for the first 30-40 generations. As the FI-2pop GA manages to amass a sufficient feasible population after the first 50 generations even without the offspring boost, the fitness differences between the two methods disappears (as shown in Table 7.3). It should be noted, however, that experiments with b_{saf} had the most obvious difference in terms of behavior; remaining experiments were closer to F_{all} or, at best, less obvious than b_{saf}. Despite the experiments reported here not showing a profound impact from offspring boost, its addition is very beneficial for the short evolutionary sprints of Sentient Sketchbook where both the total population and the number of generations is much smaller. Moreover, experiments in two-population novelty search in Chapter 8 will demonstrate how the feasible offspring boost affects search in more challenging settings.



(a) Sketch (b) Final Dungeon

Fig. 7.6: An example dungeon sketch and the dungeon level it creates; with impassable tiles in black, enemies (E) in green (goblins in the final level), treasure (T) in brown (chests in the final level), and level entrances/exits (X) in red (staircases in the final level).

7.2 Other types of Game Levels

Due to the map sketches' abstractions and the generic, parametrizable evaluations of their quality, many different types of game levels can be represented as map sketches. Assuming the view in the map sketch remains top-down, the only difference between map sketches for different game types is in the choice of special tiles and in the level qualities favored during evolution. This Section will elaborate on levels for two popular game genres: single-player roguelike dungeons and multiplayer arena-type first person shooter arenas.

7.2.1 Roguelike Dungeons

Following the success of *Rogue* (Toy and Wichman 1980), dungeons for single-player games (roguelikes) are a popular domain of procedural content generation. Map sketches can be used to represent a minimal abstraction of a roguelike dungeon, and evolve towards appropriate objective functions of roguelike quality. The special tiles in roguelike dungeons consist of dungeon exits (X), enemies (E) and treasures (T); an example map sketch and the final dungeon it creates is shown in Fig. 7.6. Following the play logic of roguelikes, a player changes dungeon levels once they step on an exit tile; it is assumed that the player starts the current level at one of the exit tiles. Within this dungeon level, a player encounters and battles monsters on enemy tiles and collects rewards (weapons, spells, or gold) on treasure tiles.

At its most basic, the purpose of a roguelike dungeon is for the player to travel from the entrance to the exit; placing the exit tile in a hard to reach location extends the gameplay time of the level. While traveling to the exit, the player encounters monsters and collects treasures; in order to make gameplay challenging, treasure is usually offered as a reward for vanquishing monsters. Treasures are therefore usually near a monster acting as its "guardian". Assuming that all treasures and monsters are equally valuable or powerful respectively, it would be useful for each monster to provide equal rewards in terms of treasure. A popular design paradigm for dungeons, exemplified in *Diablo* (Blizzard 1996), is the absence of monsters near the entrance of each level, allowing some breathing room for a player to prepare for the coming challenges. Finally, assuming that monsters are powerful enough to challenge a player, having the player battle two monsters simultaneously is over the intended challenge level of the dungeon. Evenly

distributing the monsters throughout the dungeon, with no monster being close to another, averts the danger of a player having to fight two monsters at once.

These desirable properties of roguelike dungeons can be transformed into evaluation functions with minor reinterpretations. Since entrance and exit are both represented as exit tiles (X), a winding path from the entrance to the exit can be evaluated by $f_e(S_X)$. The placement of treasures close to monsters can be evaluated by $f_s(S_E, S_T)$, while the placement of monsters away from each other and from the entrance can be achieved by $f_a(S_X \cup S_E)$. The evaluations of balance (b_s, b_a, b_e) are not as significant in the single-player dungeons as in the competitive multiplayer strategy maps. However, having a balanced distribution of treasures near each monster as per $b_s(S_E, S_T)$ couples challenge and reward more closely, while balance in the space "allotted" to each monster and exit tile $b_a(S_X \cup S_E)$ better distributes these special tiles on the map. Finally, the balance in the exploration from the entrance to the exit and vice versa as per $b_e(S_X)$ seems the least useful of the proposed evaluations; however, since the entrance can be in any tile within S_X, preemptively ensuring that all of them are equally difficult to find guarantees a winding path from entrance to exit.

Results

To test how constrained optimization performs in different constrained spaces, different dungeon sizes will be evolved in this set of experiments. Small dungeons consist of 66 tiles (arranged in 6 rows of 11 tiles each) and must contain 2 exit tiles (one entrance, one exit), between 3 and 5 monster tiles and between 2 and 4 treasure tiles. Similar to small strategy game levels, the small map and genotype size is expected to easily satisfy the playability constraints. Medium dungeons consist of 144 tiles (12 by 12) and must contain 2 exit tiles, between 5 and 10 monster tiles and between 4 and 8 treasure tiles. The medium dungeon is more than double the size of the small dungeon, and similarly may contain double the amount of monsters and treasures than the small dungeon; due to the requirement that monsters and treasures are connected with the exit tiles, medium maps are much more likely to be infeasible when randomly generated. Finally, large dungeons consist of 16 by 16 tiles and must contain 2 exit tiles, between 10 and 20 monster tiles and between 8 and 16 treasure tiles. As with large strategy game maps, the large number of special tiles is expected to hinder both the satisfaction of playability constraints and the progress of optimization.

A few indicative combinations of fitnesses will be explored:

- $F_{treas} = \frac{1}{2} f_s(S_E, S_T) + \frac{1}{2} b_s(S_E, S_T)$.
- $F_{monst} = \frac{1}{2} f_a(S_X \cup S_E) + \frac{1}{2} b_a(S_X \cup S_E)$.
- $F_{exit} = \frac{1}{2} f_e(S_X) + \frac{1}{2} b_e(S_X)$.
- $F_{all} = \frac{1}{3} F_{treas} + \frac{1}{3} F_{monst} + \frac{1}{3} F_{exit}$.

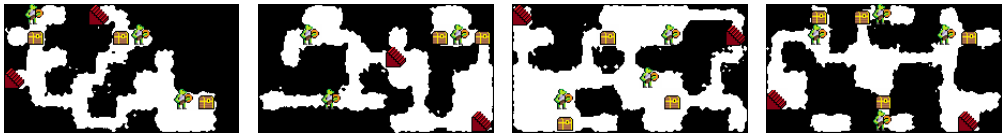
Table 7.4 displays the contributing fitness dimensions of the single best final dungeon for each experiment. Results are averaged across 50 runs, with standard error shown in parentheses. Although usually the balance dimension outperforms the quality dimension (e.g. $f_s(S_E, S_T)$

	$f_s(S_E, S_T)$	$f_a(S_X \cup S_E)$	$f_e(S_X)$	$b_s(S_E, S_T)$	$b_a(S_X \cup S_E)$	$b_e(S_X)$
Small Maps						
F_{treas}	0.683 (0.014)	—	—	0.972 (0.005)	—	—
F_{monst}	—	0.514 (0.021)	—	—	0.936 (0.014)	—
F_{exit}	—	—	0.981 (0.004)	—	—	0.999 (0.001)
F_{all}	0.533 (0.020)	0.547 (0.019)	0.931 (0.007)	0.818 (0.018)	0.858 (0.015)	0.984 (0.004)
Medium Maps						
F_{treas}	0.668 (0.010)	—	—	0.936 (0.006)	—	—
F_{monst}	—	0.485 (0.019)	—	—	0.889 (0.013)	—
F_{exit}	—	—	0.983 (0.003)	—	—	0.999 (0.000)
F_{all}	0.421 (0.015)	0.464 (0.016)	0.939 (0.006)	0.752 (0.011)	0.716 (0.018)	0.985 (0.003)
Large Maps						
F_{treas}	0.504 (0.009)	—	—	0.853 (0.007)	—	—
F_{monst}	—	0.416 (0.017)	—	—	0.791 (0.015)	—
F_{exit}	—	—	0.966 (0.004)	—	—	0.999 (0.000)
F_{all}	0.338 (0.009)	0.409 (0.011)	0.941 (0.006)	0.769 (0.007)	0.608 (0.010)	0.991 (0.001)

Table 7.4: Contributing fitness dimensions' scores for the best final dungeons. Values displayed are averaged across 50 runs, with standard error shown in parentheses.

and $b_s(S_E, S_T)$), it is surprising to see that $b_s(S_E, S_T)$ and $b_a(S_X \cup S_E)$ have scores that are far from the optimal (1.000) even in small maps. For strategy game levels, the b_a and b_s heuristics were quick to attain optimal values in such fitness dimensions; this change in behavior is likely due to the dungeons' heuristics dependence on monsters, which are more numerous than bases in strategy games (especially in larger maps).

The highest scoring final individual among the 50 runs for each fitness dimension is shown in Fig. 7.7 for small, medium and large dungeons. The dungeons are represented as complete levels for better visual identification. Unsurprisingly, each map evolved towards F_{treas}, F_{monst}, F_{exit} possesses the patterns favored by the fitness dimensions included in its quality evaluation. Dungeons evolved for F_{treas} often have a number of monsters equal to the number of rewards and each reward adjacent to a single monster (see Fig. 7.7a and Fig. 7.7e); since exits are not accounted in the evaluation of F_{treas}, exits can be placed far from each other as in Fig. 7.7e or nearby as in Fig. 7.7i. Dungeons evolved for F_{monst} avoid placing monsters near each other (or near any exits), which often results in a single monster placed in a dungeon "room" or "corridor", as can be seen in Fig. 7.7f. Dungeons evolved for F_{exit} place exits on the ends of long corridors, ensuring that most of the level will need to be traversed (along with monsters encountered and treasures found) before the player can progress to the next level; obviously, since only exits are considered, monsters and treasures will be dispersed randomly in the level, resulting in numerous nearby monsters as in Fig. 7.7g or unguarded treasures as in Fig. 7.7k. When evolving towards all dungeon qualities (F_{all}), the generated levels possess many desirable features, such as evenly dispersed monsters guarding nearby treasures as in Fig. 7.7d. However, aggregating all fitness dimensions in a single weighted sum proves challenging for medium and large maps, where there are several monsters clumped in a group as well as unguarded treasures (Fig. 7.7l). The low fitness scores for both $f_s(S_E, S_T)$ and $f_a(S_X \cup S_E)$ explain the low quality of medium and large dungeons in terms of those features; on the other hand, the dungeon exits are always placed on faraway locations such as at the end of winding corridors as in Fig. 7.7l.

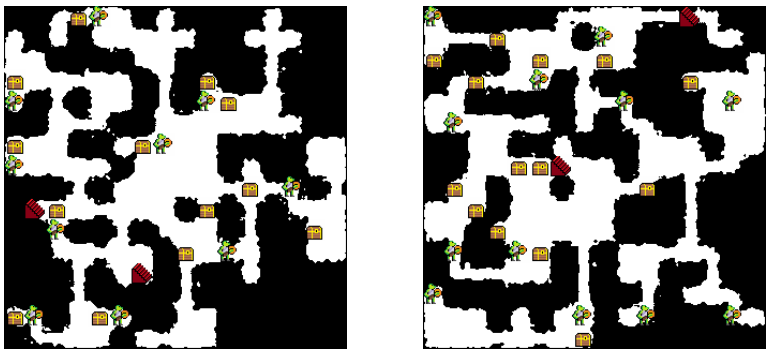


(a) $F_{treas.}$ (b) $F_{monst.}$ (c) $F_{exit.}$ (d) $F_{all.}$

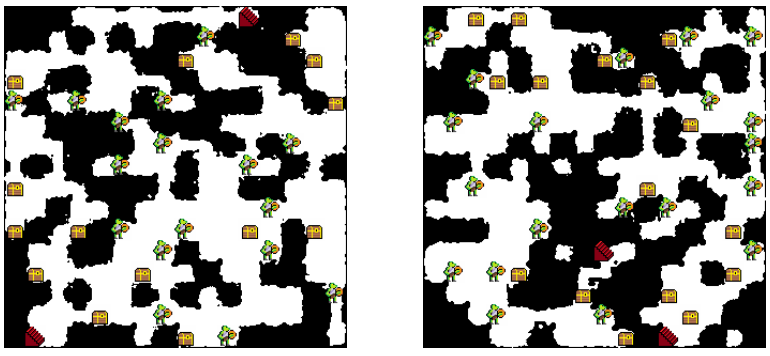
Small Maps

(e) $F_{treas.}$ (f) $F_{monst.}$ (g) $F_{exit.}$ (h) $F_{all.}$

Medium Maps



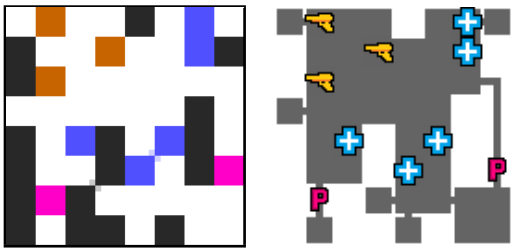
(i) $F_{treas.}$ (j) $F_{monst.}$



(k) $F_{exit.}$ (l) $F_{all.}$

Large Maps

Fig. 7.7: Best scoring final dungeons evolved for multiple objectives, across 50 runs.



(a) Sketch (b) Final FPS Map

Fig. 7.8: An example FPS map sketch and the final level it creates, with weapons (W) in orange, healing packs (H) in blue, and team spawn points (P) in purple.

.

7.2.2 First Person Shooter Arenas

Competitive multiplayer first-person shooters (FPS) such as *Team Fortress 2* (Valve 2007) can be evaluated as strategy maps (replacing resources with weapons and ammo); co-operative first-person shooters such as *Left 4 Dead* (Valve 2008) can be evaluated as dungeons (replacing monsters with bosses and rewards with ammo and healing). In this Section, map sketches will be used to represent competitive, arena-like FPS levels, and their special tiles consist of weapons (W), healing packs (H) and team spawn points (P). Copying traditional gameplay of games such as *Quake III Arena* (id Software 1999), members of the same team always spawn on the same team spawn point and, using either their own weapons or the more powerful weapons spread around the level, attempt to kill as many opponents as possible. Wounded players can restore some of their health by running over healing packs; dead players respawn at their team's spawn point.

It is highly unlikely for a commercially successful FPS level to let players of opposite teams spawn next to each other (unless it's a specific game mode); therefore, good FPS levels should have team spawn points far away from each other. On the other hand, since usually weapons strewn around the level are more powerful than the "default" player weapons, such powerful weapons tend to be placed in obscure, difficult to reach locations. This is not a ubiquitous design, since for instance *Unreal Tournament 2004* (Epic Games 2004) distributes most weapons evenly across the level; should one consider only high-powered weapons such as sniper rifles or rocket launchers, however, such weapons tend to be placed far from player spawn points. On the other end of the spectrum, healing packs tend to be placed in locations where designers expect the fighting to be the thickest, such as in chokepoints leading to the enemies' spawn point. This design pattern is motivated by the designers' goal of providing non-stop combat action by allowing players in heavy firefights to heal back the damage accrued and thus stay in the fight longer.

These desirable properties of FPS can be transformed into evaluation functions with minor reinterpretations. Placing enemy teams' spawn points faraway from each other can be achieved in a straightforward way with $f_e(S_P)$. Placing weapons in remote locations can be achieved with $f_e(S_W \cup S_P)$, which rewards weapons far away from team spawn points as well as from each other. Finally, the placement of health packs in the level is evaluated by $f_a(S_H \cup S_P)$; by distributing health packs evenly across the level (except near the team spawn points since

newly spawned players have full hitpoints and thus do not require healing) the computational designer makes no assumptions about where the fighting will be the thickest; the fairness of the health pack distribution is ensured via $b_a(S_H \cup S_P)$. Like in strategy games, the highly competitive nature of arena-like FPS levels places a lot of importance on the evaluations of balance. Thus, it is required that all players can discover an enemy spawn point at the same ease ($b_e(S_P)$) and can acquire a powerful weapon spending the same effort ($b_e(S_P \cup S_W)$).

Results

To test how constrained optimization performs in different constrained spaces, different FPS level sizes will be evolved in this set of experiments. Small FPS levels are 8 by 8 tiles and must contain 2 team spawn points, between 4 and 6 healing packs and 3 weapons. Similar to small strategy game levels, the small map and genotype size is expected to easily satisfy the playability constraints. Medium FPS levels are 8 by 16 tiles and must contain 2 team spawn points, between 6 and 10 healing packs and between 4 and 6 weapons. Finally, large FPS levels are 16 by 16 tiles and must contain 2 team spawn points, between 10 and 15 healing packs and 6 weapons. As with large strategy game maps, the large number of special tiles is expected to hinder both the satisfaction of playability constraints and the progress of optimization.

A few of the most indicative combinations of fitnesses will be explored:

- $F_{hp} = \frac{1}{2} f_a(S_H \cup S_P) + \frac{1}{2} b_a(S_H \cup S_P)$.
- $F_{spawn} = \frac{1}{2} f_e(S_P) + \frac{1}{2} b_e(S_P)$.
- $F_{weap} = \frac{1}{2} f_e(S_W \cup S_P) + \frac{1}{2} b_e(S_W \cup S_P)$.
- $F_{all} = \frac{1}{3} F_{hp} + \frac{1}{3} F_{spawn} + \frac{1}{3} F_{weap}$.

Table 7.5 displays the contributing fitness dimensions of the single best final FPS level for each experiment. Results are averaged across 50 runs, with standard error shown in parentheses. Unsurprisingly, the fitness dimension of $f_e(S_P)$ manages to reach higher values than $f_a(S_H \cup S_P)$ and $f_e(S_W \cup S_P)$, since it considers only a small subset of the special tiles considered by the latter fitness dimensions. Since healthpacks are much more numerous than the other special tiles, it is more difficult to distribute them evenly on the level, which explains the low score for $f_a(S_H \cup S_P)$; even $b_a(S_H \cup S_P)$ has lower scores than the other balance dimensions $b_e(S_P)$ and $b_e(S_W \cup S_P)$. When exploration takes into account both weapons and team spawn points, it is more challenging for evolution to find locations for such tiles which are difficult to reach from all other similar tiles; the scores in $f_e(S_W \cup S_P)$ are considerably lower than those of $f_e(S_P)$ despite the fact that they are calculated using the same f_e formula of eq. (5.5).

The highest scoring final individual among the 50 runs for each fitness dimension is shown in Fig. 7.9 for small and medium FPS levels and in Fig. 7.10 for large FPS levels. Results are represented as complete levels for better visual identification. The best FPS levels evolved for F_{hp}, F_{spawn} and F_{weap} possess the expected patterns. In levels evolved for F_{hp}, healthpacks are evenly distributed in the level (with two healthpacks rarely in the same "room") as can be seen in Fig. 7.10a. In levels evolved for F_{spawn}, the team spawn points are placed in corners

	$f_a(S_H \cup S_P)$	$f_e(S_P)$	$f_e(S_W \cup S_P)$	$b_a(S_H \cup S_P)$	$b_e(S_P)$	$b_e(S_W \cup S_P)$
Small Maps						
F_{hp}	0.523 (0.021)	—	—	0.933 (0.014)	—	—
F_{spawn}	—	0.988 (0.004)	—	—	1.000 (0.000)	—
F_{weap}	—	—	0.849 (0.005)	—	—	1.000 (0.000)
F_{all}	0.445 (0.019)	0.943 (0.007)	0.764 (0.006)	0.928 (0.014)	0.988 (0.003)	1.000 (0.000)
Medium Maps						
F_{hp}	0.476 (0.021)	—	—	0.840 (0.016)	—	—
F_{spawn}	—	0.955 (0.006)	—	—	0.998 (0.001)	—
F_{weap}	—	—	0.748 (0.005)	—	—	1.000 (0.000)
F_{all}	0.436 (0.018)	0.903 (0.010)	0.634 (0.008)	0.785 (0.017)	0.986 (0.003)	1.000 (0.000)
Large Maps						
F_{hp}	0.459 (0.013)	—	—	0.769 (0.015)	—	—
F_{spawn}	—	0.938 (0.006)	—	—	0.999 (0.000)	—
F_{weap}	—	—	0.742 (0.005)	—	—	1.000 (0.000)
F_{all}	0.410 (0.016)	0.898 (0.009)	0.651 (0.009)	0.636 (0.018)	0.987 (0.002)	1.000 (0.000)

Table 7.5: Contributing fitness dimensions' scores for the best final FPS levels. Values displayed are averaged across 50 runs, with standard error shown in parentheses.

of the map (such as the far top and far bottom in Fig. 7.9f). Although not explicitly targeted by $f_e(S_P)$, the maps of Fig. 7.9b, 7.9f, 7.10b all have more than one pathway from one team spawn point to the other (likely due to $b_e(S_P)$); this makes for more interesting strategies than simply defending the single chokepoint of the level and can allow sneaking past enemies to reach their spawn point (which would be effective in capture-the-flag matches). Since F_{spawn} doesn't account for weapons and health packs, such FPS levels may be unbalanced in terms of resources readily available to each team. In levels evolved for F_{weap}, weapons tend to be in locations which are hard to reach from team spawn points, e.g. in Fig. 7.9c, but larger maps do not follow this pattern perfectly as weapons may be close to each other (see Fig. 7.9g) or near the team spawn points (see Fig. 7.10c). Maps evolved towards F_{all} have interesting properties such as a central arena in Fig. 7.9d into which team spawn points lead to and from which players can access the side corridors containing health packs and weapons. The medium FPS level of Fig. 7.9h has team spawn points in different halves of the level (top and bottom), with a single chokepoint connecting them in the form of a narrow corridor; each side of this corridor has an even distribution of weapons and health, although one weapon per team is relatively close to that team's spawn point. Finally, the large FPS level of Fig. 7.10d has the two spawn points (despite first appearances) very far away from each other, at the ends of long, winding corridors. The topmost team's spawn point has access to two weapons while the other team has immediate access to only one, but the level is large enough to provide enough areas for fighting as well as hiding and there is not as clear a chokepoint as that of Fig. 7.9h.

7.3 General Findings

The experiments reported in this Chapter show that the feasible-infeasible two-population GA is capable of optimizing a very diverse set of map sketches, from small dungeons to large strategy game levels. By testing the optimization process on three different types of levels with distinct gameplay, the heuristics of level quality were shown to be able to capture many

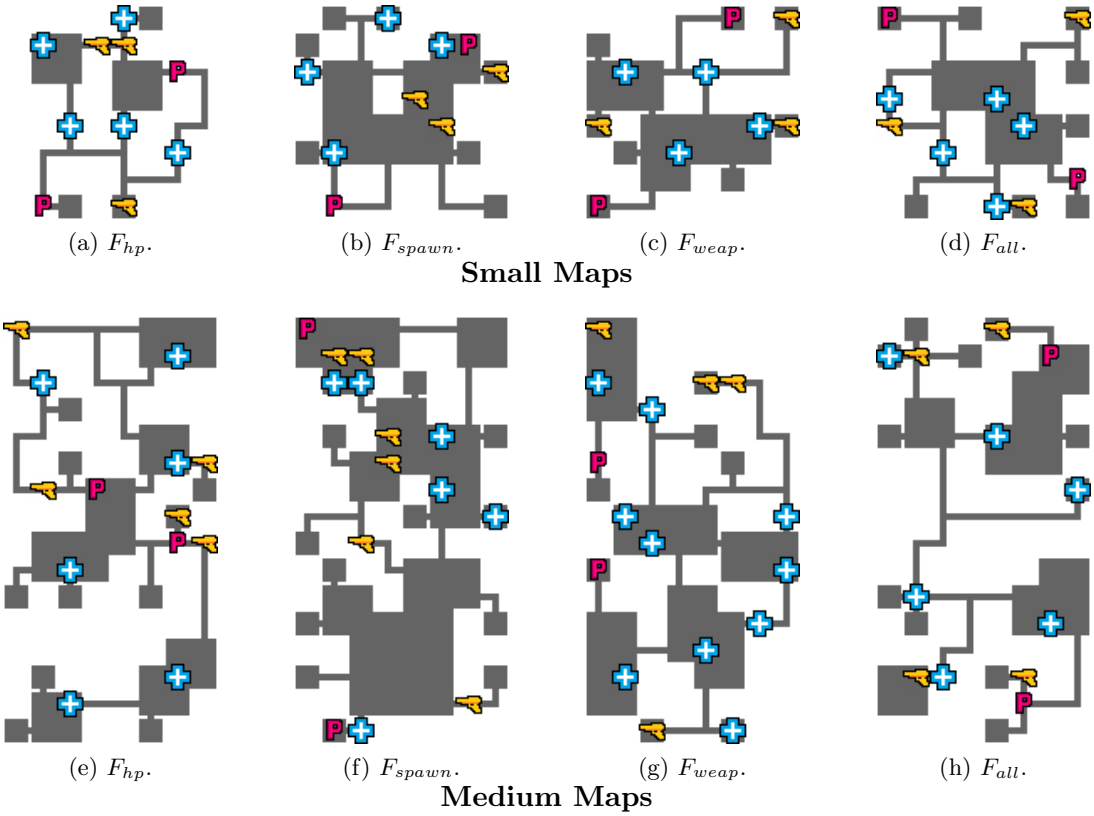
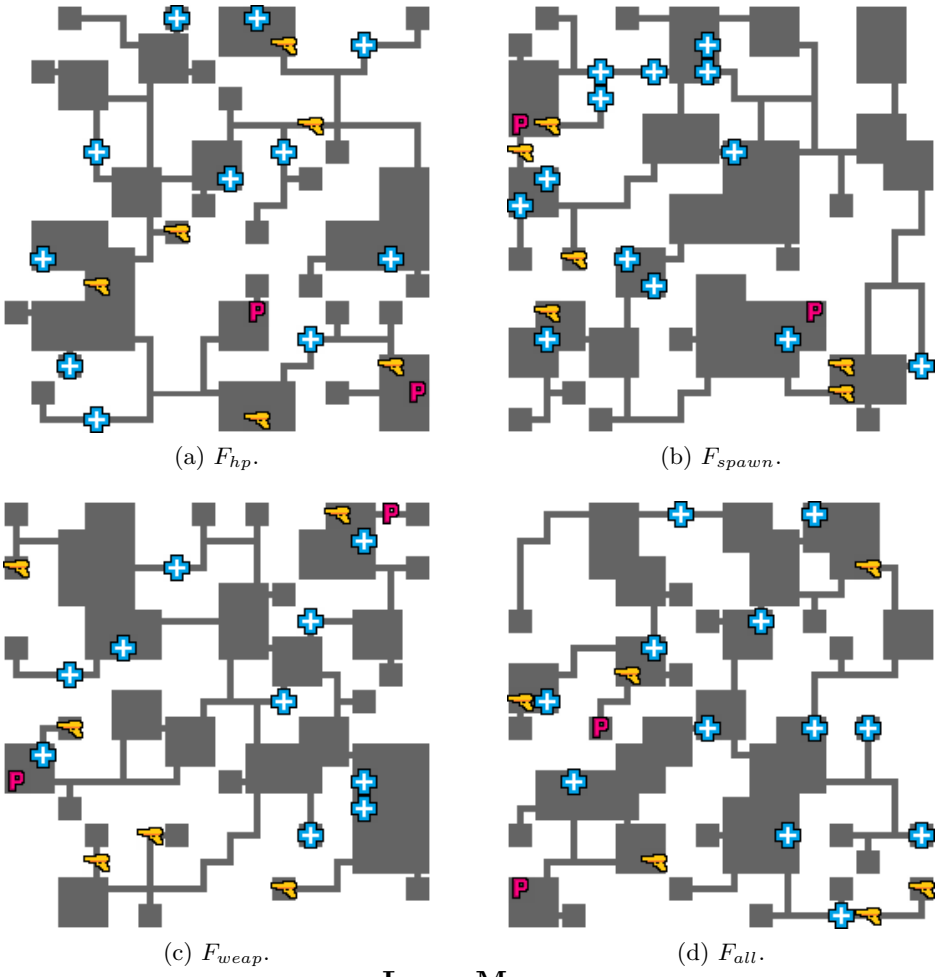


Fig. 7.9: Best scoring final small and medium FPS levels evolved for multiple objectives, across 50 runs.

of the desired properties of most types of levels. Regarding the optimization behavior of these heuristics, it largely depends on the type and size of the level, the number of special tiles (and special tile types) and the number of heuristics being evolved simultaneously. The paragraphs below will address these findings.

Regarding the optimization progress, experiments with strategy game levels showed that when the FI-2pop GA evolves according a single objective, evolution usually discovers good individuals within a few generations. Evolution past that point is not particularly fruitful and the feasible population tends to converge towards the best individual. This is particularly useful for Sentient Sketchbook, which has strict limitations on the number of generations it can afford if its suggestions are generated in quasi-real-time. Optimization becomes slower when multiple objectives are aggregated into a weighted sum; while dimensions of balance are easier to optimize and have more impact in such aggregated fitness scores, no fitness seems to dominate the optimization process since the fitness dimensions are not conflicting. While perhaps not perfect, map sketches for various genres evolved towards two or more objectives feature most of the desirable patterns of game levels of that genre.

As expected, larger map sketches are more likely to be infeasible due to the longer paths connecting different special tiles; these paths are more likely to be blocked by mutating a passable tile into impassable or by an unfortunate crossover. As evidenced by experiments



(a) F_{hp}. (b) F_{spawn}.

(c) F_{weap}. (d) F_{all}.

Large Maps

Fig. 7.10: Best scoring final large FPS levels evolved for multiple objectives, across 50 runs.

with strategy game levels, mutation operators (at least when applied exclusively) are more likely to cause such blocked paths and thus to result in fewer feasible individuals. Apart from the longer paths, however, larger map sketches also contain more special tiles and thus level quality via the formulations of Section 5.2.1 combines more variables. When many special tiles are included as reference tiles in the fitness formulations of eq. (5.3)–(5.8), the effect of averaging makes comparisons between map sketches' fitness scores less informative; for instance, a low exploration value between two bases is straightforward to understand, while a low exploration value between six bases may lead to some bases being far away from each other while the remaining bases are clumped in the middle of the map. The heuristics of level quality are designed to work on low-resolution map sketches with few special tiles, and seem to perform well in small maps; the large maps investigated in this set of experiments are likely the upper limit in terms of map complexity that can be meaningfully supported by the current quality metrics and representation.

7.4 Summary

This Chapter has documented a broad range of experiments on the constrained optimization of map sketches. When evolving different types of game levels, the generalizability of the evaluations of level quality from 5.2.1 was tested. When optimizing single or multiple objectives, the efficiency of the FI-2pop GA at discovering good content which fulfill all the requirements of playability and quality was tested. Finally, the impact of the offspring boost addition to the FI-2pop GA was tested along with the impact of recombination and mutation in the search process. Chapter 8 follows a similar process, testing how constrained novelty search methods perform under different constrained spaces and using different operators. Since the two-population novelty search methods are a variation of novelty search and an alternative to minimal criteria novelty search, the next Chapter focuses more on comparisons between these methods. Unlike experiments in this Chapter, novelty search (and its heuristic for map difference) is not affected by the level type being evolved and thus all experiments in Chapter 8 are performed on strategy game levels.

Chapter 8

Constrained Novelty Search of Map Sketches

Chapter 7 explored different combinations of fitness functions, domains and constrained spaces in order to assess the optimization behavior of the FI-2pop GA. Since all experiments in that Chapter used the FI-2pop GA, few comparisons were necessary except when testing the impact of genetic operators or the impact of the offspring boost. Since this thesis introduces two two-population novelty search algorithms, feasible-infeasible novelty search and feasible-infeasible dual novelty search as described in Chapter 4, comparisons between the two algorithms but also with existing methods of novelty search are necessary to assess their performance. Moreover, while genetic optimization in Chapter 7 follows a "standard" protocol for evaluating optimization methods, novelty search and its detachment from objectives makes the design of an experimental protocol challenging. As such, this Chapter describes the experimental protocol chosen in Section 8.1 before commencing extensive experiments with different methods of novelty search for map sketches. The choice of game levels (e.g. dungeons or first-person shooter arenas) does not affect the tile-based distance d_{vis} from eq. (5.10) used for novelty search; as the most studied genre in the lifetime of Sentient Sketchbook, strategy game levels are chosen as the map sketches being evolved.

8.1 Hypotheses and Experimental Protocol

In order to assess the behavior of constrained novelty search methods, a number of experiments were conducted and reported below; these experiments compare between minimal criteria novelty search (MCNS), feasible-infeasible novelty search (FINS), feasible-infeasible dual novelty search (FI2NS) and traditional, unconstrained novelty search (NS). The introduction of two-population novelty search aims to test three hypotheses (*H1*, *H2*, *H3*) regarding existing methods; a fourth hypothesis (*H4*) is made for the addition of the offspring boost mechanism to the two-population methods:

H1: FINS can find feasible individuals faster than FI2NS, NS and MCNS in highly constrained search spaces, where a feasible individual is hard to discover and not likely to exist in the initial population. In such search spaces MCNS performs random search and both NS and FI2NS search for novelty in infeasible space, while the infeasible population of FINS is guided by a measure of feasibility.

H2: FINS and FI2NS can find more feasible individuals than NS. As NS does not distinguish between feasible and infeasible individuals, it is likely to explore infeasible space when the

feasible space is small; this amounts to a few feasible individuals, limiting the efficiency of novelty search in that space.

H3: MCNS has lower diversity than all other methods, as it kills all infeasible individuals once a feasible individual is found. FINS, FI2NS and NS all retain infeasible individuals; such individuals are likely to eventually create feasible offspring and enhance the diversity of feasible individuals. On the other hand, MCNS kills infeasible offspring of feasible parents, thus forgetting their genetic information; this is likely to limit the effectiveness of novelty search near the border of feasible space and thus lower the diversity of feasible individuals.

H4: FINS and FI2NS become more efficient from the offspring boost mechanism in terms of the number and diversity of feasible individuals discovered. The feasible population of FINS and FI2NS must be large enough for search to be efficient; without the offspring boost, their feasible population may be too small.

To validate the above hypotheses, we derive four performance metrics (two for *H1* and one for each of *H2*, *H3*) to be used in the experiments of this thesis: (*H1*a) the number of evolutionary runs (out of 50) where at least one feasible individual is discovered (n); (*H1*b) the first generation in which a feasible individual is discovered (g); (*H2*) the number of feasible individuals (p); and (*H3*) the diversity of feasible individuals ($\bar{d}_{vis}$). *H4* can be tested via p and $\bar{d}_{vis}$, since its goal is higher diversity afforded by larger feasible populations. Diversity is measured by calculating the average distance d_{vis} from eq. (5.10) between all feasible individuals in the population (ignoring the novel archive). The values of p and $\bar{d}_{vis}$ are calculated after 100 generations, at which point the evolutionary run is terminated. Apart from n, performance metrics are averaged across 50 runs, with the standard deviation shown in parentheses. Significance throughout this Chapter is measured via two-tailed t-tests (assuming unequal variances), with $p < 0.05$; when comparing more than one method, the Bonferroni correction is applied [62]. Unless otherwise stated, experiments in this Chapter are performed on a total population of 100 individuals, which can be feasible or infeasible.

The algorithms' behavior in different constrained search spaces will be showcased via experiments on small, medium and large strategy game levels. The different map sizes are identical to those in Section 7.1, and range from a large feasible spaces for small maps to a highly constrained search space of large maps. Since the tile difference heuristic d_{vis} from eq. (5.10) is not dependent on the type of level or the type of special tiles, experiments with strategy game levels are sufficiently indicative of the behavior of novelty search in other types of game levels such as the First-person Shooters or roguelike dungeons of Chapter 7.

The algorithms behave differently depending on the search space and the genetic operators used. In order to test that, experiments documented in Section 8.2 compare the two-population approaches with unconstrained NS and MCNS and test the impact of the offspring boost mechanism. These experiments take place on a total population of 100 individuals, and start from a random initial population; to test how these parameters affect the behavior of the different methods, Section 8.3 tests population sizes both larger and smaller than 100 individuals, while Section 8.4 bypasses the challenge of discovering feasible individuals by injecting one or more feasible solutions into the initial population.

Map Size	Method	n	g	p	$\bar{d}_{vis}$
Small	NS	50	0.0 (0.2)	**14.7 (6.3)**	0.523 (0.079)
	MCNS	50	0.1 (0.4)	**96.7 (2.1)**	**0.402 (0.019)**
	FINS	50	0.0 (0.1)	**51.6 (5.4)**	**0.467 (0.016)**
	FI2NS	50	0.1 (0.4)	**48.9 (3.7)**	0.523 (0.010)
Medium	NS	50	7.3 (7.1)	**1.2 (1.4)**	0.192 (0.252)
	MCNS	43	13.4 (12.5)	**94.7 (2.9)**	0.299 (0.030)
	FINS	50	**2.6 (1.6)**	**50.1 (5.1)**	**0.418 (0.016)**
	FI2NS	50	8.7 (8.6)	**47.5 (3.6)**	**0.481 (0.012)**
Large	NS	34	53.7 (25.0)	**0.1 (0.3)**	**0.000 (0.000)**
	MCNS	7	43.3 (27.6)	**85.9 (28.2)**	**0.176 (0.077)**
	FINS	50	8.5 (3.9)	**48.9 (4.2)**	**0.361 (0.023)**
	FI2NS	30	60.5 (20.9)	**44.1 (4.5)**	**0.293 (0.119)**

Table 8.1: Performance metrics for different novelty search approaches used with 2-point crossover and 1% chance of mutation: number of runs (out of 50) where a feasible individual was discovered (n) and the first generation in which it occurred (g), as well as the final number of feasible individuals (p) and their average diversity ($\bar{d}_{vis}$) of each experiment. For metrics other than n, results are averaged across 50 runs with standard deviation in parentheses; significantly different values from all other methods for the same map size are displayed in bold, while (significantly) best values for each metric are underlined. Note that the best value is the lowest for g and the highest for p and $\bar{d}_{vis}$.

8.2 Impact of Genetic Operators

The choice of genetic operators can greatly affect the behavior of any genetic algorithm, even more so novelty search which relies on divergence to guide it. Similar to Section 7.1.3, this section compare how the different novelty search methods behave when the genetic algorithm uses 2-point crossover (with a 1% chance of mutation) and when it uses only mutation; as a shorthand, the former operators will be identified as *recombination* and the latter as *mutation*. Unless otherwise noted, FINS and FI2NS use the offspring boost mechanism; experiments in the second part of this section validate the *H4* hypothesis and support the use of the offspring boost in all experiments in this thesis.

Table 8.1 shows the performance metrics of different methods of novelty search evolving via the 2-point crossover of two parents and a mutation chance of 1% on the offspring. Comparing between the different map sizes, it is immediately obvious that the performance of different methods is greatly affected by the shape of the feasible space: in highly constrained spaces, minimal criteria novelty search (MCNS) and unconstrained novelty search (NS) appear to struggle to discover or retain feasible individuals, respectively. MCNS discovers a feasible individual in 43 out of 50 runs for medium maps and only in 7 out of 50 runs for large maps; as the initial population does not contain feasible individuals, MCNS sets the fitness of the entire population to 0 and performs random search until a feasible individual is found. For experiments with recombination, it is obvious that random search is not particularly efficient at finding parents whose offspring are feasible. Unconstrained novelty search, on the other hand, discovers feasible individuals more often than MCNS, but does not succeed at retaining them in the population: it was not uncommon for runs with NS to discover a single feasible individual in one generation, only to have it disappear in the next. As NS does not have an explicit (via a feasibility check) or implicit (via the distance metric) indication that the

Map Size	Method	n	g	p	$\bar{d}_{vis}$
Small	NS	50	0.0 (0.1)	**10.9 (4.0)**	0.599 (0.032)
	MCNS	50	0.1 (0.2)	**77.6 (6.3)**	**0.577 (0.017)**
	FINS	50	0.0 (0.2)	38.9 (4.5)	0.603 (0.007)
	FI2NS	50	0.0 (0.0)	38.5 (4.3)	0.607 (0.010)
Medium	NS	50	4.0 (2.8)	**1.7 (1.5)**	**0.392 (0.282)**
	MCNS	50	4.7 (2.9)	**55.2 (7.7)**	0.567 (0.011)
	FINS	50	**2.5 (1.6)**	**28.2 (4.3)**	0.577 (0.012)
	FI2NS	50	4.0 (2.8)	**25.4 (4.7)**	0.572 (0.015)
Large	NS	50	16.9 (9.0)	**0.2 (0.4)**	**0.000 (0.000)**
	MCNS	50	20.4 (11.4)	**31.8 (6.0)**	0.540 (0.018)
	FINS	50	**6.2 (1.9)**	16.1 (4.2)	0.544 (0.026)
	FI2NS	50	17.1 (7.8)	15.2 (4.2)	**0.523 (0.040)**

Table 8.2: Performance metrics for different novelty search approaches used only with mutation: number of runs (out of 50) where a feasible individual was discovered (n) and the first generation in which it occurred (g), as well as the final number of feasible individuals (p) and their average diversity ($\bar{d}_{vis}$) of each experiment. For metrics other than n, results are averaged across 50 runs with standard deviation in parentheses; significantly different values from all other methods for the same map size are displayed in bold, while (significantly) best values for each metric are underlined. Note that the best value is the lowest for g and the highest for p and $\bar{d}_{vis}$.

individual discovered is valuable, it was often discarded in favor of more visually different yet infeasible individuals. Both FINS and FI2NS have an explicit indication of feasible individuals' value, and once discovered these individuals are prompted to generate numerous offspring — equal to 50% of the total population due to the offspring boost mechanism. Unlike the random search of MCNS and the novelty search of FI2NS and NS, FINS attempts to minimize infeasible individuals' distance from feasibility, and this objective-driven search leads to the discovery of feasible individuals faster and more reliably (based on the low deviation of g) than the other methods in medium and large maps. Regarding the number of final feasible individuals, MCNS unsurprisingly has far more feasible individuals than other methods in all runs where a feasible individual was discovered, for all map sizes. Likely due to the fact that all infeasible individuals are killed, however, MCNS has a significantly lower diversity in the feasible population than all methods (excluding NS in medium or large maps which hardly contains any feasible individuals). In less constrained spaces such as with small maps, NS can discover a sufficient number of feasible individuals to measure their diversity; its final diversity is significantly higher than that of MCNS and FINS. For medium maps FI2NS achieves a significantly higher diversity than all other approaches, likely due to the fact that novelty search in the infeasible population discovers feasible individuals in distant or segmented areas of the search space. For large maps, the feasible population of FINS is significantly more diverse than other methods', likely due to the fact that FINS discovers individuals much earlier than other methods and thus performs novelty search on the feasible population for more generations.

Table 8.2 shows the performance metrics of different methods of novelty search evolving via the mutation of a single parent. Comparing the performance metrics with those of Table 8.1, it is clear that mutation, as a whole, tends to cause different novelty search methods to exhibit a similar behavior. Feasible individuals are discovered in all runs, even in large maps, and the discrepancies in g and $\bar{d}_{vis}$ between methods are largely subdued. All behaviors

(excluding NS in medium and large maps) reach similar values of final diversity, which are significantly higher than the values obtained by the same methods using recombination. This should not be surprising given the very nature of recombination as well as the tile-based distance metric used to measure diversity. Given an appropriate "geometric" recombination operator, evolutionary search can be seen a convex search as it searches inside the hypervolume specified by the individuals in the initial population [169]. In the current problem, the direct genotype to phenotype mapping makes it probable that the recombination operator is geometrical or almost geometrical in phenotype space (where diversity is measured). This observation could be a good explanation for the low observed diversity in experiments with prevalent recombination. More specifically, an individual created via crossover inherits parts of the map from either parent, resulting in a low distance d_{vis} between the offspring and either parent (see Fig. 5.8a). On the other hand, a map generated via mutation inherits parts of the map from a single parent, while the random mutation means that offspring of the same parent may still be dissimilar from each other (see Fig. 5.8b). Regarding the number of feasible individuals, the behavior of novelty search methods with mutation is similar to that of FI-2pop GA (Section 7.1.3) since all methods have fewer feasible individuals with mutation than with recombination, regardless of map size. However, mutation seems to enhance the likelihood of discovering feasible individuals; even the random search of MCNS discovers large feasible maps in all runs much faster than with recombination, although FINS still manages to discover such maps faster and more reliably. This is likely due to the fact that mutation can "clear" impassable regions which block paths more easily as it transforms impassable tiles into passable.

Results in this section point to very different behaviors in all performance metrics between experiments with mutation and experiments with recombination; such differences were not as pronounced for experiments with the FI-2pop GA in Section 7.1.3. To ascertain whether these differences are persistent across population sizes or initial populations, remaining experiments in this thesis will also be testing both sets of genetic operators.

Impact of the Offspring Boost

The offspring boost addition to two-population approaches was introduced in Section 4.1 as a way to increases the number of offspring created by the feasible population. The hypothesis *H4* is that the feasible population of FINS and FI2NS must be large enough for search to be efficient. While this was not validated for most instances of FI-2pop GAs in Section 7.1.4, the different search methods of FINS and FI2NS are more likely to cause feasible parents to create infeasible offspring and thus the offspring boost is likely more beneficial. To test *H4*, 50 runs of FINS and FI2NS without the offspring boost (denoted as FINS_{nb} and FI2NS_{nb} respectively) are compared with the 50 runs of FINS and FI2NS from Tables 8.1 and 8.2, which use the offspring boost. The results of these tests are shown in Table 8.3.

As expected, the offspring boost results in a significantly larger feasible population, since offspring of feasible parents are more likely to be feasible than offspring of infeasible parents. FI2NS benefits from the boost the most: without the boost, its final population is small for small maps and for medium or large maps it often consists of a single feasible individual. FI2NS_{nb} does not have a much larger number of feasible individuals than NS, although their difference is still significant. While FINS using recombination often has a sizable feasible

		Recombination		Mutation	
Map Size	Method	p	$\bar{d}_{vis}$	p	$\bar{d}_{vis}$
Small	FINS_{nb}	37.8 (10.6)	0.426 (0.025)	19.6 (5.6)	0.597 (0.013)
	FINS	51.6 (5.4)	0.467 (0.016)	38.9 (4.5)	0.603 (0.007)
	FI2NS_{nb}	20.4 (7.3)	0.511 (0.024)	12.4 (3.7)	0.605 (0.018)
	FI2NS	48.9 (3.7)	0.523 (0.010)	38.5 (4.3)	0.607 (0.010)
Medium	FINS_{nb}	28.1 (9.7)	0.337 (0.038)	6.4 (3.0)	0.556 (0.086)
	FINS	50.1 (5.1)	0.418 (0.016)	28.2 (4.3)	0.577 (0.012)
	FI2NS_{nb}	3.5 (2.6)	0.288 (0.195)	2.9 (1.2)	0.495 (0.197)
	FI2NS	47.5 (3.6)	0.481 (0.012)	25.4 (4.7)	0.572 (0.015)
Large	FINS_{nb}	23.4 (12.3)	0.239 (0.087)	2.8 (2.0)	0.410 (0.251)
	FINS	48.9 (4.2)	0.361 (0.023)	16.1 (4.2)	0.544 (0.026)
	FI2NS_{nb}	1.2 (0.7)	0.036 (0.128)	1.2 (0.5)	0.096 (0.218)
	FI2NS	44.1 (4.5)	0.293 (0.119)	15.2 (4.2)	0.523 (0.040)

Table 8.3: Performance metrics with and without (shown as $_{nb}$) the offspring boost mechanism, for FINS and FI2NS evolving via 2-point crossover and 1% chance of mutation (Recombination column) and only mutation (Mutation column). Since the offspring boost takes effect only after a feasible individual is discovered, n and g are not expected to differ from those in Tables 8.1 and 8.2 and are omitted. p and $\bar{d}_{vis}$ are compared between the same method (FINS or FI2NS) with and without the offspring boost; significantly better (higher) values are underlined. Results are averaged across 50 runs, with standard deviation in parentheses.

population even without the boost, when using mutation FINS similarly relies on the boost to reach an adequate population size. The larger feasible populations afforded by the boost allow both FINS and FI2NS to search for novelty more efficiently in the feasible space, usually resulting in significantly higher diversity values than the same methods without the boost. From the results of Table 8.3, the first part of hypothesis *H4* that the offspring boost results in more feasible individuals is validated in 12 out of 12 cases, regardless of map size, genetic operators, and two-population novelty search method. The second part of hypothesis *H4* that the offspring boost results in more diverse feasible individuals is validated for FINS in 5 out of 6 cases, and similarly for FI2NS in 5 out of 6 cases. Since *H4* is validated for the overwhelming majority of cases, every FINS and FI2NS method uses the offspring boost in the remaining experiments.

8.3 Impact of Population Size

Although depending on the search method and the optimization problem, most genetic algorithms benefit from larger populations for the additional parallel search that they afford; in theory, the same should hold true for constrained novelty search. A larger population size is more likely to create, even by chance, maps that are more different from each other and thus better steer novelty search towards higher diversity. Moreover, a larger population is more likely to discover a feasible individual in highly constrained spaces, regardless of the type of search.

While other experiments in this thesis de facto use a total population of 100 individuals, which includes both feasible and infeasible ones, Figure 8.1 displays how different performance metrics

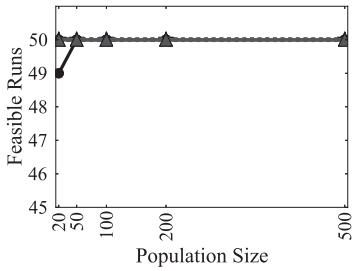
change with smaller or larger populations. Five different population sizes are tested: 20, 50, 100, 200 and 500 individuals. Several interesting conclusions can be drawn from Fig. 8.1, although they largely depend on the map size and the genetic operators used.

For all map sizes, the number of feasible runs increases as the size of the total population increases. This is hardly surprising since a larger population of infeasible individuals is faster at reaching the border of feasibility for FINS or exploring the infeasible space for FI2NS and NS. As MCNS lacks an informed heuristic for searching infeasible space, it benefits the most from the parallel random search afforded by large population sizes. The discovery of the first feasible individual is also considerably faster with larger population sizes (such as 200 or 500 individuals) for all methods, although FINS is the least affected. On the other hand, the ratio of final feasible individuals to the total population does not seem to be particularly sensitive to different population sizes. The average diversity of the final feasible individuals increases with larger populations of 200 or 500, which is hardly surprising since most forms of genetic search benefit from larger population sizes; it is interesting to note, however, that recombination benefits more from large population sizes than mutation.

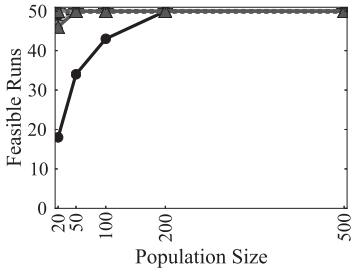
Small populations such as 20 or 50 individuals are quite detrimental for most methods. For small maps the discovery of feasible individuals is still fast and easy even for a population of 20, although the final feasible diversity is significantly lower compared to that for a population of 100. For medium maps, small populations do not particularly affect the discovery of feasible individuals for FINS, since it still consistently finds feasible individuals within the first 10 generations even for populations of 20. It does however affect the discovery of feasible individuals for FI2NS and NS, which discover individuals much later than in populations of 100, and especially MCNS which does not discover any feasible individuals in numerous runs when applying recombination. As with small maps, all methods fail to reach high values of feasible diversity for medium maps when evolving 20 or 50 individuals. For large maps, small populations are even more problematic, as even FINS struggles to discover feasible individuals with populations of 20. Other methods fare even worse, as MCNS, FI2NS and NS rarely discover feasible individuals with populations of 20; in fact, MCNS discovers a feasible large map only once when using recombination with populations of 20. Although it discovers feasible individuals rarely, MCNS can search for novelty in the feasible population more efficiently than other methods in very small populations; as other methods do not have a sufficiently large population in order to effectively explore the feasible space, MCNS achieves significantly higher diversity scores in populations of 20 for experiments with mutation.

The differences found in Section 8.2 between experiments with recombination and experiments with mutation seem to be persistent for the different population sizes tested. The number of runs with no feasible individuals is much larger for experiments with recombination than for experiments with mutation, especially with smaller population sizes. Similarly, experiments with mutation discover feasible individuals earlier, and the differences become even more pronounced with larger population sizes. Experiments with mutation consistently have fewer final feasible individuals than experiments with recombination, for populations both small and large. Regarding diversity, while experiments with mutation and recombination both benefit from larger population sizes, recombination struggles much more with small populations than mutation. Both experiments with mutation and experiments with recombination have a significantly lower final diversity with a population of 20 than with a population of 100, excluding MCNS and NS with recombination in large maps due to few feasible runs.

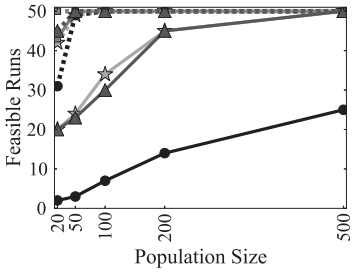
(a) Legend.



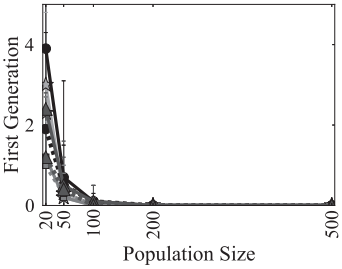
(b) Metric n for small maps.



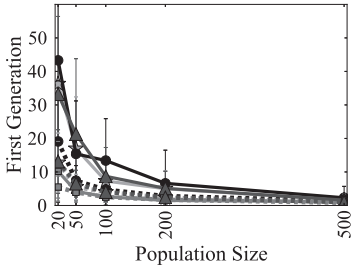
(c) Metric n for medium maps.



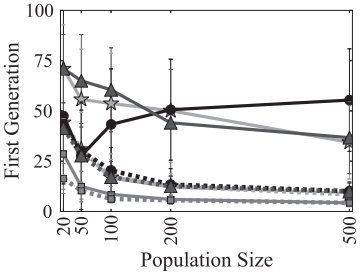
(d) Metric n for large maps.



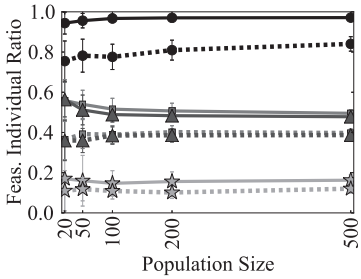
(e) Metric g for small maps.



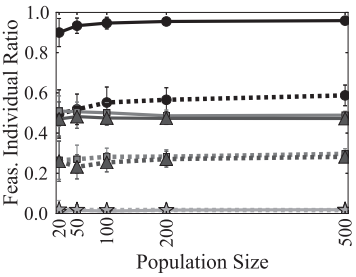
(f) Metric g for medium maps.



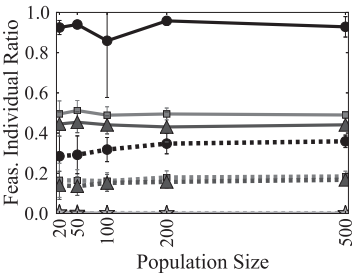
(g) Metric g for large maps.



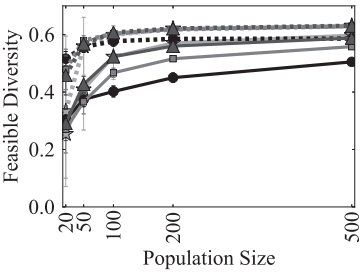
(h) Ratio of p for small maps.



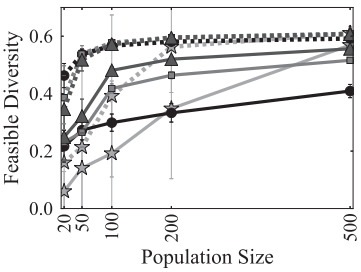
(i) Ratio of p for medium maps.



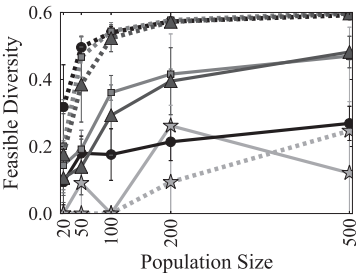
(j) Ratio of p for large maps.



(k) Metric $\bar{d}_{vis}$ for small maps.

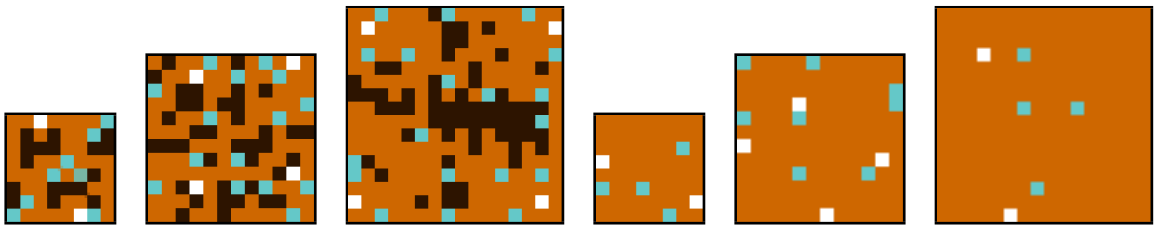


(l) Metric $\bar{d}_{vis}$ for medium maps.



(m) Metric $\bar{d}_{vis}$ for large maps.

Fig. 8.1: Comparisons of the performance metrics between different population sizes. The ratio of feasible individuals p is normalized to the population size used (a ratio of 1.0 represents the entire population size). Solid lines show experiments with recombination while dotted lines show experiments with mutation. Results are averaged across 50 runs, with errorbars denoting standard deviation.



(a) Small $seed_{one}$ (b) Medium $seed_{one}$ (c) Large $seed_{one}$ (d) Small sample $seed_{all}$ (e) Medium sample $seed_{all}$ (f) Large sample $seed_{all}$

Fig. 8.2: Types of initial seeds. Fig. 8.2a–8.2c show the feasible hand-crafted maps used to seed the initial population of the $seed_{one}$ experiments. Fig. 8.2d–8.2f show sample randomly generated feasible maps, with no impassable tiles, used to seed the initial population of the $seed_{all}$ experiments.

8.4 Impact of Initial Seeds

In highly constrained problems, such as the larger maps in the presented experiments, both unconstrained novelty search and MCNS are shown to perform poorly: unconstrained novelty search explores predominantly infeasible space, finding feasible individuals with difficulty but discarding them with ease, while MCNS performs random search which rarely discovers feasible individuals. For such cases, [131] suggest that the initial population should be "seeded" with feasible individuals to jump-start evolution. In order to test the performance of constrained and unconstrained novelty search with seeded initial populations, two experimental setups are devised: the $seed_{one}$ experiment, where a single feasible individual is injected into the initial population, and the $seed_{all}$ experiment, where every initial population consists of randomly generated individuals guaranteed to be feasible. For $seed_{one}$ experiments, the same hand-crafted individual (see Fig. 8.2) is injected into an otherwise randomly generated population for all runs and all methods of that map size. For $seed_{all}$ experiments, the random initial individuals are guaranteed to be feasible as they do not contain any impassable tiles that could block paths between bases and resources. These random individuals are generated by applying the repair algorithm on a blank map containing only passable tiles; resulting maps, as shown in Fig. 8.2, possess the minimum amount of allowed bases and resources. Tables 8.4 and 8.5 present the final number of feasible individuals and their diversity for experiments with initial populations of the two types of seeds.

In experiments using 2-point crossover and 1% chance of mutation, displayed in Table 8.4, the final number of feasible individuals is significantly larger for $seed_{all}$ initial populations than for $seed_{one}$ or unseeded initial populations from Table 8.1 for 9 out of 12 cases. This is mostly due to the fact that the $seed_{all}$ initial population contains no impassable tiles; as only mutation can add impassable tiles and the chance of mutation is low, even after 100 generations maps are unlikely to have numerous impassable tiles. This benefits NS the most, as it retains a sizable feasible population even for large maps when starting from a $seed_{all}$ initial population; this is not true for $seed_{one}$ or unseeded initial populations, where NS has very few, if any, feasible individuals for medium and large maps. Since recombination applied to a $seed_{all}$ initial population results in fewer impassable tiles in the final evolved maps, the final diversity of maps for $seed_{all}$ initial populations is usually lower than that of $seed_{one}$ initial populations or even

		$seed_{all}$ init. population		$seed_{one}$ init. population	
Map Size	Method	p	d_{vis}	p	d_{vis}
Small	NS	**30.4 (12.7)**	0.506 (0.012)	**16.2 (7.5)**	0.521 (0.029)
	MCNS	**97.5 (1.7)**	**0.403 (0.019)**	**97.1 (2.2)**	**0.403 (0.018)**
	FINS	**58.7 (12.1)**	**0.448 (0.032)**	**52.8 (5.9)**	**0.466 (0.014)**
	FI2NS	**49.1 (5.4)**	0.509 (0.012)	**49.2 (4.3)**	0.519 (0.012)
Medium	NS	**35.2 (11.7)**	**0.460 (0.010)**	**1.8 (2.3)**	0.291 (0.228)
	MCNS	**97.4 (2.0)**	**0.375 (0.013)**	**96.3 (2.0)**	0.343 (0.018)
	FINS	66.7 (14.5)	**0.404 (0.045)**	**50.6 (4.5)**	**0.417 (0.013)**
	FI2NS	63.1 (16.1)	**0.434 (0.063)**	**47.4 (3.9)**	**0.481 (0.013)**
Large	NS	**24.6 (9.5)**	0.399 (0.014)	**0.0 (0.1)**	0.000 (0.000)
	MCNS	**97.0 (1.9)**	**0.315 (0.014)**	**95.5 (2.5)**	**0.274 (0.020)**
	FINS	53.4 (8.6)	**0.343 (0.015)**	**49.4 (3.7)**	**0.360 (0.020)**
	FI2NS	52.3 (11.1)	0.391 (0.044)	**45.4 (3.6)**	**0.431 (0.011)**

Table 8.4: Performance metrics with seeded initial populations for different novelty search approaches used with 2-point crossover and 1% chance of mutation. Since the initial population is seeded with at least one feasible individual, $n = 50$ and $g = 0$ for all methods and are omitted. Results are averaged across 50 runs, with standard deviation in parentheses. Significantly different values in the performance metrics compared to all other methods for the same map size are displayed in bold, while (significantly) best (highest) values for each metric are underlined.

unseeded ones. For all map sizes, however, FI2NS and NS have significantly higher diversities than FINS and MCNS for $seed_{all}$ initial populations. For $seed_{one}$ initial populations, there are very few differences in the final number of feasible individuals compared to the unseeded initial populations of Table 8.1. However, since a feasible individual jump-starts evolution in highly constrained spaces where MCNS and FI2NS normally struggle to discover feasible individuals, these methods have more generations to perform novelty search in feasible space. This results in a significantly higher diversity for MCNS with a $seed_{one}$ initial population than with an unseeded one for medium and large maps, while FI2NS has a significantly higher diversity than MCNS and FINS even in large maps, where it underperformed with unseeded initial populations.

In experiments using mutation of a single parent, displayed in Table 8.5, the behavior of the different methods is overall not very different from those of Table 8.2. Unconstrained novelty search does not manage to maintain an adequate number of feasible individuals, even with a $seed_{all}$ initial population; this is likely due to the larger number of impassable tiles introduced via mutation which increases the likelihood of blocked paths between bases and resources. The feasible diversity scores are quite similar between methods applied on the same map size, excluding NS for medium and large maps. It is not surprising that MCNS, FINS and FI2NS have a significantly larger number of final feasible individuals for experiments with $seed_{all}$ initial populations than with unseeded populations. It is surprising however that for medium and large maps, the number of feasible individuals with $seed_{one}$ initial populations is significantly lower than that with unseeded ones in 3 out of 8 cases; this is likely due to the larger number of mutations applied to the already "crowded" hand-crafted maps, which more easily create disconnected paths.

		$seed_{all}$ init. population		$seed_{one}$ init. population	
Map Size	Method	p	$\bar{d}_{vis}$	p	$\bar{d}_{vis}$
Small	NS	**11.5 (4.8)**	0.555 (0.019)	**11.0 (4.3)**	0.602 (0.025)
	MCNS	**84.5 (5.1)**	**0.548 (0.006)**	**79.3 (6.1)**	**0.574 (0.019)**
	FINS	40.8 (4.0)	0.554 (0.007)	38.6 (4.1)	0.604 (0.006)
	FI2NS	41.9 (4.7)	0.558 (0.006)	38.8 (4.3)	0.603 (0.010)
Medium	NS	**1.6 (1.5)**	**0.324 (0.266)**	**1.6 (1.4)**	**0.358 (0.287)**
	MCNS	**60.6 (6.9)**	0.534 (0.006)	**50.5 (6.6)**	**0.555 (0.009)**
	FINS	31.6 (5.2)	0.539 (0.009)	26.9 (5.3)	**0.573 (0.013)**
	FI2NS	30.5 (5.4)	0.535 (0.010)	25.5 (4.5)	**0.564 (0.015)**
Large	NS	**1.3 (1.1)**	**0.272 (0.243)**	**0.2 (0.6)**	0.199 (0.313)
	MCNS	**71.6 (7.9)**	0.500 (0.005)	**26.6 (5.9)**	0.490 (0.019)
	FINS	34.4 (6.1)	0.498 (0.008)	14.4 (4.2)	0.472 (0.056)
	FI2NS	33.0 (6.8)	0.496 (0.008)	13.1 (3.5)	0.467 (0.052)

Table 8.5: Performance metrics with seeded initial populations for different novelty search approaches used with mutation only. Since the initial population is seeded with at least one feasible individual, $n = 50$ and $g = 0$ for all methods and are omitted. Results are averaged across 50 runs, with standard deviation in parentheses. Significantly different values in the performance metrics compared to all other methods for the same map size are displayed in bold, while (significantly) best (highest) values for each metric are underlined.

8.5 General Findings

The argument for two-population novelty search methods builds on four hypotheses:

According to *H1*, FINS can find feasible individuals faster than FI2NS, NS and MCNS in highly constrained search spaces. Sections 8.2 and 8.3 test that hypothesis in medium and large maps, which are unlikely to be feasible by chance. *H1* is validated in 14 out of 20 experiments for the highly constrained spaces of medium and large maps, as FINS discovers feasible individuals at significantly lower g values than other methods in these 14 cases, regardless of genetic operators used; significance was not achieved in the remaining cases primarily due the high standard deviation in g of other approaches for large maps using recombination. Granted that in 4 of the 6 cases where significant differences in g were not found, the difference in n between FINS and the other methods was overwhelming, *H1* can arguably be considered validated.

According to *H2*, FINS and FI2NS can find more feasible individuals than NS. With FINS and FI2NS using the offspring boost, *H2* was validated in all 42 cases examined, as NS had significantly fewer feasible individuals than all other methods for all experiments of Sections 8.2, 8.3 and 8.4. On the other hand, MCNS had significantly more feasible individuals than other methods in all but one of these 42 cases. Therefore, while *H2* is validated, FINS and FI2NS have fewer feasible solutions than MCNS.

According to *H3*, MCNS has lower diversity than all other methods. Since in most cases NS did not have any feasible individuals for map sizes other than small, we will test whether MCNS has a lower diversity than FINS or FI2NS. Testing experiments of Sections 8.2, 8.3 and 8.4, *H3* was validated in 28 out of 42 cases regarding FINS having a significantly higher diversity than MCNS, and again in 28 out of 42 cases regarding FI2NS having a significantly higher diversity than MCNS. The experiments which did not validate *H3* were mostly with

small populations of 20 and 50 individuals, where *H3* was validated only in 1 and 3 of 12 cases for FINS and FI2NS respectively. While not validated in its entirety, *H3* seems to hold in larger populations regardless of genetic operators or initial populations used.

According to *H4*, FINS and FI2NS are more efficient in terms of population size and diversity when the offspring boost mechanism is applied. Experiments in Section 8.2 validated *H4* in all 12 cases regarding boost significantly increasing the feasible population's size and in 10 out of 12 cases regarding boost significantly increasing the diversity of feasible solutions. These results largely validate *H4*.

As a more general conclusion from the experiments of Chapter 8, it is clear that choosing a novelty search method depends as much on the shape of the feasible space as on the intended outcomes of the experiment. MCNS is able to create many feasible individuals which are often not particularly diverse, since it discards infeasible offspring of feasible parents. In highly constrained spaces MCNS performs poorly as it cannot find a feasible individual using an unguided, random search. On the other hand, MCNS can explore large feasible spaces efficiently, requiring a smaller population than other methods. Unconstrained NS can create diverse feasible individuals but it has no notion of the value of feasible solutions, and thus it is likely to discard them in favor of infeasible ones. This leads to few feasible solutions even when constraints are easily satisfied, but is particularly detrimental in highly constrained spaces as NS rarely retains even a single feasible solution. FI2NS shares many of the properties of NS, including the way it searches the infeasible space; as it has a notion of the value of feasible individuals, however, it can retain them and increase their numbers due to the offspring boost mechanism. Without the offspring boost, FI2NS does not fare much better than NS in highly constrained spaces; with the offspring boost, however, FI2NS usually has a sizable population of feasible individuals with a diversity rivaling that of NS. FINS retains fewer feasible individuals than MCNS and their diversity is often lower than that of FI2NS or NS; however, FINS can discover feasible individuals easily in highly constrained spaces as its infeasible population searches towards the border of feasibility. This advantage of FINS over other methods can be somewhat mitigated by injecting feasible individuals in the initial population; in such cases, FI2NS and, under conditions, MCNS can be equally or better suited at finding diverse feasible solutions.

The choice of genetic operators similarly depends on the intended outcomes of the experiment, but also on the genotype to phenotype mapping and on the distance metric used. Experiments in this thesis have demonstrated that evolving solely via mutation leads to more diverse feasible individuals and a faster, more consistent discovery of feasible solutions in highly constrained spaces. However, as will be discussed in Section 11.1.2, the high diversity of experiments with mutation could be an artifact of the direct mapping between genotype and phenotype and of the tile-based distance metric used to calculate diversity.

8.6 Summary

This chapter compared the two-population novelty search methods proposed in Chapter 7 with existing methods of constrained or unconstrained novelty search. The experiments evaluated the behavior of the different methods with different sets of genetic operators, with and without the offspring boost mechanism, with different population sizes and with feasible individuals

seeding the initial population. Three different types of constrained spaces were explored, from small maps which were easily feasible to the highly constrained space of large maps. Having evaluated the performance of constrained optimization in Chapter 7 and that of novelty search in Chapter 8, the next Chapter evaluates the final AI module of Sentient Sketchbook: the designer models of process, preference and style.

Chapter 9

Adaptive Models for Human Use

The algorithms introduced for generating map sketch suggestions according to a measure of game level quality or a measure of difference were evaluated in a standardized experimental setup in Chapters 7 and 8 respectively. Evaluating the behavior of the designer modeling extensions for Sentient Sketchbook, described in Section 5.6, is not as straightforward. On one hand, the adaptive models are designed to recognize and accommodate human users, while on the other hand such evaluation is not rigorous and can give little insight on the actual impact of the models (even with a large number of testers). Controllable experiments are thus necessary for evaluating these adaptive models. To test the optimization behavior of choice-based interactive evolution in the model of designer style, an experiment with artificial users with predictable patterns for choosing suggestions is implemented in Section 9.1. Human process and symmetry goals on the other hand are not easily simulated via artificial agents, since the simulated users do not favor visual symmetries or draw directly on the canvas. The models of designer process and goals are evaluated in Section 9.2 based on interaction data from past design sessions with expert users presented in more detail in Chapter 6.

9.1 Testing Designer Models of Style with Artificial Agents

In order to test how the model of style can (a) learn from past choices and (b) influence future shown suggestions, a number of simulated uses of Sentient Sketchbook were performed using different models of style. The artificial agents that use the tool are assumed to create map sketches solely by selecting among generated suggestions in every step and replacing their user sketch with their favorite suggestion. Directly drawing tiles on the canvas is not allowed since the model of style does not learn from manual edits of the sketch. The simulation follows the process of Sentient Sketchbook closely: every time the user's sketch changes (in this case by being replaced by a generated suggestion), a new evolutionary sprint of 10 generations is performed starting from an initial population consisting of mutations of the user's current sketch. The algorithm has a population of 20 individuals (feasible or infeasible) and evolves via a FI-2pop GA using fitness-proportionate roulette wheel selection and two-point crossover with a 1% chance of mutation. The feasible population of Sentient Sketchbook's suggestions is guided by the adaptive model of designer style, represented as a weighted sum, while after the short evolutionary sprint the 6 fittest map sketches are shown to the user (although the shown sketches may be fewer if there are not enough feasible individuals in the population). The choice of 6 suggestions shown to the user is made under the assumption that the adaptive model of style replaces the current 6 suggestions which are individually evolved towards specific strategic qualities as described in Section 5.5.4.

The model of style is evaluated via artificial agents attempting to create small strategy map sketches as per experiments of Chapters 7 and 8; these small maps consist of a 8 by 8 tile grid, two bases and 4 to 10 resources. Experiments in Chapter 7 showed that other map sizes (medium and large strategy game maps) exhibit a similar optimization behavior, although their optimization is slower (both in terms of running time and convergence speed). Therefore, the behavior of small maps should be indicative of the general trend among different map sizes and types of game levels. To demonstrate as many different behaviors as possible, the artificial users select suggestions according to a single objective throughout the run, according to multiple objectives (which may change during the course of the design), or exhibit unexpected, subversive behaviors such as minimizing a strategic property.

The performance metric consists of the fitness score of the suggestions shown to the artificial user in every step (i.e. every time an artificial user selects a suggestion to replace its current sketch). Both the average fitness score and the range (maximum and minimum values) of all shown suggestions (up to 6) are considered. The values are averaged across 50 independent runs, and Figures display the standard error of the average fitness score of shown suggestions as error bars. Between steps, an evolutionary sprint of 10 generations is performed according to a specific (adapted or otherwise) measure of designer preference; for initialization, an empty map (without impassable tiles, bases or resources) is evolved for 10 generations to create the suggestions shown at step 1.

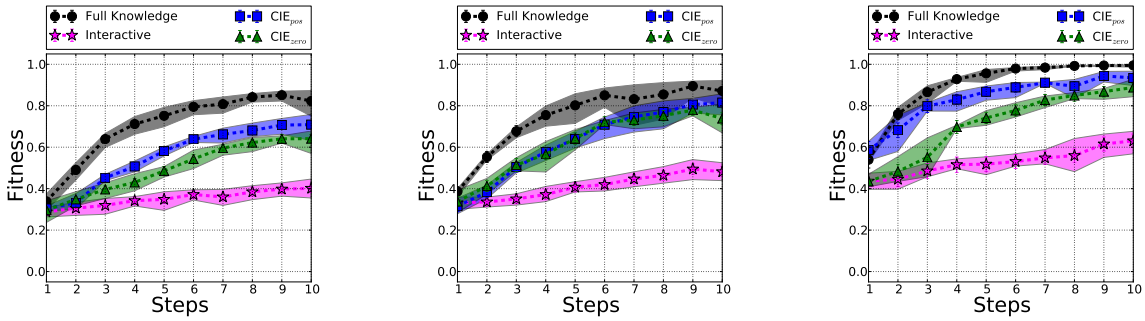
This set of experiments test the performance of two adaptive models of taste, with different initial states, along with a model closer to interactive evolution and a baseline which does not account for the artificial agents' choices. The different methods are described below:

- **CIE**$_{pos}$ uses the choice-based interactive evolution equation of 4.1 to increase or decrease the weights of fitness dimensions which are prominent (higher or lower) in the map sketch selected by the artificial user compared to the average of all unselected map sketches. This method initializes all weights with an equal positive amount adding up to 1 (in this case of 6 weights, all weights start from $\frac{1}{6}$); the method assumes that users prefer maps with high strategic qualities, and is in line with previous work on adaptive models of taste [138, 135].

- **CIE**$_{zero}$ similarly uses equation 4.1 to adapt its weights, but does not make any assumptions on the part of the users' taste and initializes all weights at 0.

- The **interactive** model evaluates feasible individuals according to their genotypical distance from the previous selected suggestion (which is assumed to have optimal fitness of 1). The interactive model does not show content for selection in every generation, and follows suggestions by Takagi for faster convergence in interactive evolution via distance-based prediction of unseen artifacts' fitness values [242].

- The **full knowledge** baseline uses the same fitness function as the agent to evaluate and evolve its feasible individuals; thus, it does not need to take account of the artificial agent's selections.



(a) Artificial agent choosing the highest f_{res}. The Y axis is the score of f_{res}.

(b) Artificial agent choosing the highest f_{saf}. The Y axis is the score of f_{saf}.

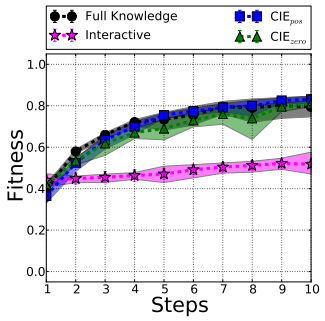
(c) Artificial agent choosing the highest f_{exp}. The Y axis is the score of f_{exp}.

Fig. 9.1: The average fitness (dotted line) and the fitness range (filled area) of shown suggestions in consecutive steps of simulated runs for agents with a single objective. Values are averaged across 50 runs.

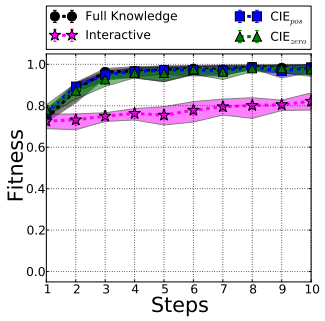
9.1.1 Artificial agents with a singular objective

The most straightforward human user that can be simulated is a single-minded one: the experiments in this Section incorporate artificial agents selecting according to one of the f_{res}, f_{saf}, f_{exp}, objectives for strategy game levels. The balance dimensions of b_{res}, b_{saf} and b_{exp} are not included in the report since they are much easier to optimize according to experiments in Section 7.1.1, and more importantly were not often selected as suggestions by human users according to Table 6.1. Figure 9.1 displays the average fitness of shown suggestions as well as their value ranges. Observing Fig. 9.1, the CIE$_{pos}$ model is quick to accommodate the user's taste and create maps which score high according to the user's fitness. Obviously, the full-knowledge model outperforms other methods as it has a single objective (compared to the multiple fitness dimensions of CIE methods) and does not need to learn from user choices. The CIE$_{zero}$ model often needs more steps to learn from user choices, leading to a slow start which eventually however also reaches high values. This is especially obvious in Fig. 9.1c, although for the other objectives CIE$_{zero}$ has a comparable performance to CIE$_{pos}$ (e.g.in Fig. 9.1b). Interactive evolution understandably performs poorly as it only rewards similarity with the previous selected individual, essentially limiting the maximum fitness it can reach to the fitness of the previous individual. If the previous selected individual is not particularly good according to the user's fitness, only the stochasticity of the genetic search can lead to improvements; the small step-wise fitness score increase (e.g.in Fig. 9.1a) shows that such improvements are minor and unlikely.

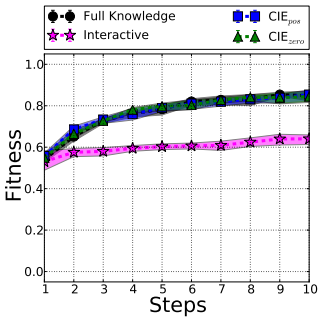
It should be noted that CIE$_{pos}$ and the full knowledge methods often start from a higher score at step 1 than CIE$_{zero}$ and the interactive method; this is most noticeable in Fig. 9.1c. This behavior is due to the fact that the suggestions of step 1 are evolved according to a weighted sum with positive weights and a single objective (respectively). By comparison, the CIE$_{zero}$ and interactive methods reward all individuals with 0, as they have initial weights of zero and no previous selected individual respectively; the random search before step 1 results in suggestions with lower scores than the other two methods.



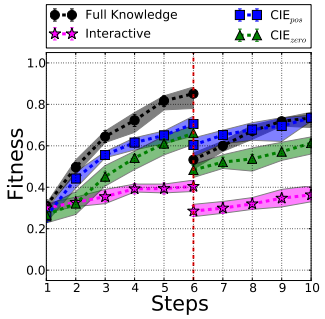
(a) Artificial agent choosing the highest F_{qual}, which is displayed on the Y axis.



(b) Artificial agent choosing the highest F_{bal}, which is displayed on the Y axis.



(c) Artificial agent choosing the highest F_{all}, which is displayed on the Y axis.



(d) Artificial agent choosing the highest f_{saf} for steps 1-5 and the highest f_{res} for steps 6-10. The Y axis is f_{saf} before step 6 and f_{res} after it.

Fig. 9.2: The average fitness (dotted line) and the fitness range (filled area) of shown suggestions in consecutive steps of simulated runs for agents with multiple objectives. Values are averaged across 50 runs.

9.1.2 Artificial agents with a multiple objectives

While the single-minded artificial agents of Section 9.1.1 are a controllable, straightforward way of evaluating the designer model of style, it is naive to assume that users of Sentient Sketchbook always select the suggestion based on the same objective. In this section, somewhat more complex artificial agents are used, as they select either based on a composite of objectives or change their mind with regards to their desired objectives midway through the session.

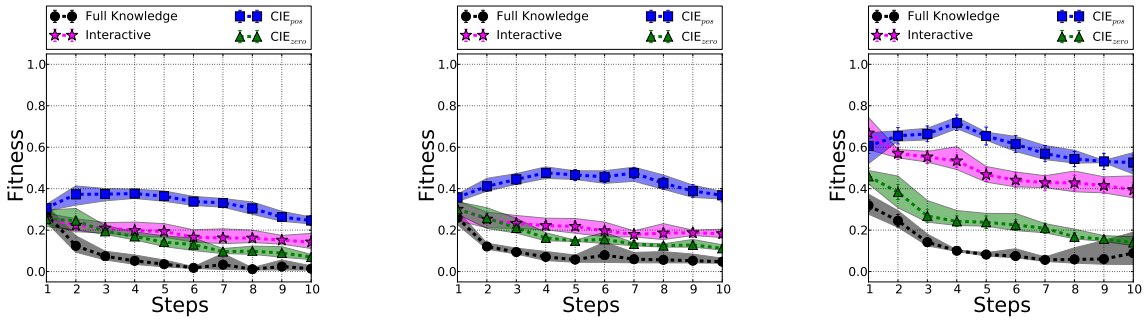
Figures 9.2a, 9.2b and 9.2c display the average fitness of shown suggestions as well as their value ranges when artificial agents select based on the composite objectives F_{qual}, F_{bal}, F_{all} respectively. Each of these fitness functions are described in Section 7.1.2 and aggregate multiple objectives into a sum, with F_{qual} incorporating f_{res}, f_{saf} and f_{exp}, F_{bal} incorporating b_{res}, b_{saf} and b_{exp} and F_{all} incorporating all of these dimensions. We observe a similar pattern in all three figures, that the full knowledge methods and both CIE methods have very similar performance in terms of average fitness of shown suggestions. This is somewhat surprising given the fact that the full knowledge method does not need to adapt to the user's taste as it

knows it beforehand, and also considering its obvious superiority in experiments with single objectives in Section 9.1. The fact that CIE$_{pos}$ performs at a similar level as the full knowledge model for F_{all} is expected, since the initial fitness function of CIE$_{pos}$ is a weighted sum with weights at $\frac{1}{6}$ which is identical to F_{all}. Apparently, however, even CIE$_{zero}$ manages to adapt its weights early on, even as early as the first iteration, and evolves appropriate content to its current agent. This behavior of both CIE methods can also be traced to the synergies between fitness dimensions which were also mentioned in Section 7.1.2. For instance, maps optimal for different balance dimensions exhibited similar patterns; therefore, if an adaptive model such as CIE$_{zero}$ increases the weight of a single balance dimension, it is likely that resulting artifacts will have high scores in the other balance dimensions (and thus in F_{all}) as well. Moreover, since the adaptive model may change its weights from iteration to iteration, finetuning them to the selected suggestion, it is even likely to be more appropriate than the static weighted sum of the full knowledge method which is susceptible to being dominated by certain fitnesses.

Another type of behavior which is more human-like is the artificial agent that "changes its mind". To simulate that, the artificial agent selects content based on a single objective for the first 5 steps, and at the 6th step changes its mind and selects content based on another objective for the remaining steps 6-10. Figure 9.2d captures the change in fitness scores for such an agent that selects according to f_{saf} in steps 1-5 and according to f_{res} in steps 6-10. The first 5 steps of this artificial agent follow the same patterns as those in Section 9.1.1: the full knowledge model outperforms CIE models which in turn outperform interactive evolution. When the artificial agent changes its selection criteria at the 6th step, the full knowledge model is shown to have a relatively low f_{res} score (the criterion for steps 6-10) due to the fact that evolution previously focused entirely on optimizing f_{saf} to the detriment of other strategic qualities. By comparison, the CIE$_{pos}$ model, which has positive weights for f_{saf} as well as f_{res} (at least initially), seems to optimize towards both objectives during the first steps and thus has individuals with relatively higher scores in f_{res} on step 6. CIE$_{zero}$ is not initially biased towards maximizing all strategic qualities, and thus adaptation is less likely to result in positive weights for f_{res} in steps 1-5, leading to a lower fitness in f_{res} by step 6, and a slower adaptation of its weight in steps 6-10. Finally, interactive evolution performs poorly both in steps 1-5 for f_{saf} and in steps 6-10 for f_{res} and results in overall worse maps with low scores in both fitnesses.

9.1.3 Subversive artificial agents

There is an implicit assumption in the design of artificial agents for Sections 9.1.1 and 9.1.2 that human users always desire maps that have the patterns of the level quality evaluations of Sentient Sketchbook. While experiments with human users in Chapter 6 largely support this assumption, as users mostly agreed that the heuristics of Sentient Sketchbook match what they look for in strategy game levels, the adaptive models' ability at capturing less predictable agents is an important benchmark. To test such "subversive" agents, artificial agents in this Section select the suggestion with the lowest score in a specific objective. As an example, maps with low f_{res} scores have resources equally far away from all bases, which makes gameplay favor rush strategies to grasp those contested resources as soon as possible; it is not impossible that such a map quality could be desirable, under circumstances, to a human designer. Note that this Section considers that the artificial agents are attempting to subvert one of the core



(a) Artificial agent choosing the lowest f_{res}. The Y axis is the score of f_{res}.

(b) Artificial agent choosing the lowest f_{saf}. The Y axis is the score of f_{saf}.

(c) Artificial agent choosing the lowest f_{exp}. The Y axis is the score of f_{exp}.

Fig. 9.3: The average fitness (dotted line) and the fitness range (filled area) of shown suggestions in consecutive steps of simulated runs for subversive agents. Values are averaged across 50 runs.

assumptions of Sentient Sketchbook and its suggestions, i.e. that human users would want their map sketches to be more balanced, with open spaces where players can expand etc. Another form of subversive behavior would be to select according to criteria not considered by the heuristics at all, e.g. visual symmetry or similarities with existing games; testing such individuals is more challenging as it is unclear what the results would demonstrate, as described in Section 11.1.3.

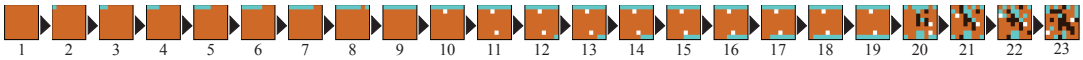
Figure 9.3 displays the average fitness of shown suggestions as well as their value ranges when artificial agents select based on the lowest score in the same objectives as Fig. 9.1. A surprising behavior for CIE$_{pos}$ is obvious in Fig. 9.3: CIE$_{pos}$ initially improves the fitness score of suggestions, despite the fact that the user desires the opposite. This behavior is likely due to the initial positive weights which are not adapted to a sufficient degree so that they become negative. Eventually, the fitness score of suggestions begins to drop, although the process is very slow. It is likely that the remaining fitness dimensions in the weighted sum have positive weights (at least in the first few steps) which may drive evolution towards directions not reflected in the user's taste and thus not shown in the Figure. On the other hand, the other methods have similar behaviors as in experiments of Section 9.1.1: the full knowledge model reaches the lowest (i.e. most desirable) scores, followed by CIE$_{zero}$ and interactive models in order of performance. The poor behavior of CIE$_{pos}$ can be traced back to the initial bias towards suggestions with high scores in the strategic qualities chosen; since this same bias worked well in cases where users selected according to the strategic qualities, there is no obvious answer regarding which version of CIE is more desirable.

9.2 Testing Designer Models of Process and Symmetry Goals with Past Design Sessions

In order to test how the models of process and symmetry goals perform, simulated runs with artificial users would not provide any insights since the artificial agent does not draw directly on the canvas (which is used for modeling process) and does not exhibit visual symmetries

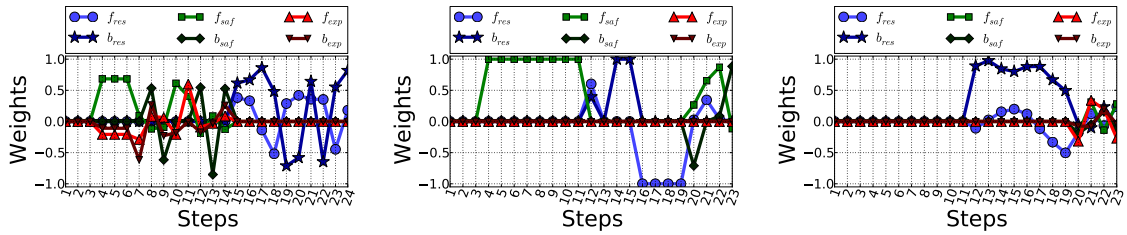
(a) Session 1

(b) Session 4



(c) Session 7

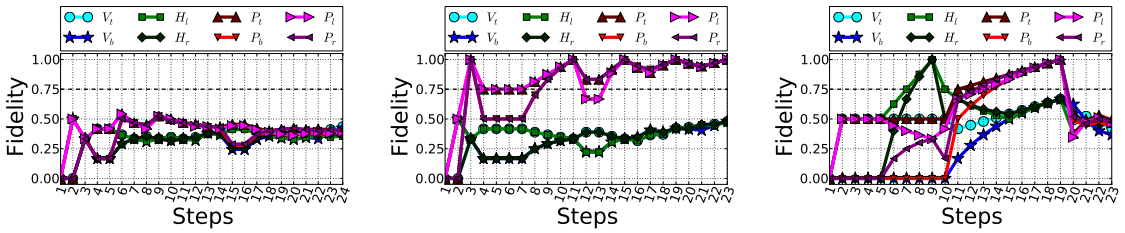
Fig. 9.4: The past design sessions of Fig. 6.4 which will be investigated in Section 9.2.



(a) Session 1 (b) Session 4 (c) Session 7

Fig. 9.5: Stepwise changes in the weights of the six fitness dimensions of strategic qualities for the creation paths of Fig. 9.4. The changes reflect the designer model of process which compares the sketch of the current step with that of the previous step.

(which is modeled as a designer goal). Instead, experiments for these two models will use design sessions from Chapter 6 for a proof-of-concept application of the proposed extensions. Focusing on small maps since the number of user actions is more manageable, the three most distinct creation paths of Fig. 6.4 will be considered; Fig. 9.4 displays the selected creation paths for better context. In session 1, the designer started by placing bases, then adding impassable regions to make discovery of enemy bases more challenging and finally added resources; both the final map and the intermediate steps seem rather haphazard and have no obvious symmetries. In session 4, the designer starts by placing bases as far away from each other as possible (on two corners of the map), then places impassable tiles, resources and more impassable tiles in that order; starting from the symmetrically placed bases and in most steps of the creation path, the symmetrical nature of the map is rather obvious to a casual human observer. In session 7, the designer starts by placing resources at the top of the map followed by a nearby base; consecutively, another base at the bottom of the map is added followed by resources symmetrical to the ones at the top. After step 19, the user is assumed to run out of ideas and successively replaces their sketch with computer-generated suggestions; steps 20-23 are all computer-generated suggestions, which explains why they lack the symmetries of the human-authored sketch.



(a) Session 1 (b) Session 4 (c) Session 7

Fig. 9.6: Stepwise changes of the *sym* score for different symmetry types for the creation paths of Fig. 9.4. The symmetry model with the highest score over the threshold of 0.75 (shown as a dotted line) would cause suggestions to have a perfect symmetry according to that model by changing the mapping from genotype to phenotype.

9.2.1 Adapting to Current Process

In terms of the model of process, the weights for all strategic qualities on each step of the creation path are shown in Fig. 9.5. For session 1, the weights of f_{exp}, b_{exp}, f_{saf}, b_{saf} change drastically in steps 3-15 while the user adds impassable tiles which increase the difficulty of reaching an enemy base (f_{exp}, b_{exp}) and create "pockets" of safe areas near each base (f_{saf}, b_{saf}). Since the weights of these four qualities change so drastically in steps 3-15, generated suggestions would evolve to maximize a strategic quality in one step only to attempt to minimize it in the next step (e.g. b_{saf} at step 12 and 13). Once the user begins placing resources after step 15, the weights of f_{res} and b_{res} are the only ones being adjusted (since remaining fitnesses do not depend on resources). These weights also change dramatically in steps 15-24, focusing from f_{res} to b_{res} and vice versa. For session 4, the model of process is much more stable; since the bases are initially placed as far away as possible, exploration is always at its optimal value and the weights of f_{exp} and b_{exp} never change throughout the process. Between steps 4 and 12, the user's addition of impassable tiles blocks off passable regions which are rendered safe for one of the bases, leading to the model of process focusing singularly (with a weight of 1) on f_{saf}. Steps 14 and 15 detect that the user places resources next to the bottom base balancing those of the top base, and increases the weight of b_{res} to 1. Once the designer places resources in the corners furthest from the bases, the model adjusts f_{res} to -1, detecting that the current focus is unsafe resources. Finally, when the designer adds impassable regions in steps 20-23, weights for f_{res}, b_{res}, f_{saf}, b_{saf} change drastically as the distances between bases change, in turn affecting all of those fitnesses. For session 7, the absence of two bases until step 11 makes the calculation of any fitness dimension impossible as they all hinge upon comparisons of distances between bases. While strategic qualities can not be evaluated, their weights are obviously not adjusted. Between steps 11 and 19 the user places resources and therefore the weights of f_{res} and b_{res} are adjusted: since the user adds resources near the base that didn't previously have any, the model detects that the user attempts to balance the map (b_{res}). Once the user begins making large changes to the sketch via suggestions at steps 20-23, the weights are adjusted drastically, although surprisingly the weights are rather close to each other and none of them reaches very high (or very low) values.

9.2.2 Evaluating Symmetry Goals

Concerning symmetry, Fig. 9.6 displays the *sym* values of different symmetry types. According to Fig. 9.4a, session 1 has low *sym* values for all symmetry types from start to finish, which can be verified with a casual observation of all steps in Fig. 9.4a. According to Fig. 9.4b for session 4, the point symmetry of the two bases at step 3 immediately results in a value of 1 for all point symmetries (P_t, P_b, P_r, P_l). While impassable tiles are not added in a symmetrical fashion in steps 4-7, the symmetry of bases pushes the *sym* score of P_l above the threshold of 0.75 and thus all suggestions during these steps would have point symmetry. Consecutive additions of impassable tiles complete the symmetry, and later additions of resources do not affect the best symmetry as much (since two of three tile types have perfect symmetry). From step 3 until the end of the session, there is at least one point symmetry with a *sym* score above the 0.75 threshold, so all suggestions from step 3 onward would exhibit point symmetry. On the other hand According to Fig. 9.4c for session 7, the model assumes a goal of horizontal symmetry during steps 7-10 due to resources on the left half being reflected on the right half. Once the user begins drawing the bottom row of resources at step 12, the model assumes a goal of point symmetry (or vertical symmetry to a lesser degree) as the bottom half begins to reflect the top half. While suggestions in steps 12-19 would exhibit point symmetry, once the user drastically changes their map via computer-generated alternatives at steps 20-23, no particular symmetries are found. This lack of symmetry in generated suggestions in the previous version of Sentient Sketchbook shown in Fig. 9.4 reveals the need for a designer model of symmetry goals, as the suggestions at steps 20-23 would have perfect point symmetry similar to the user's sketch at step 19.

9.3 General Findings

Although not yet tested with human designers, the experiments described in this Chapter show several promising properties for each of the designer models proposed. The designer model of style can find the key strategic qualities favored by a designer, even when it starts without any prior bias (CIE_{zero}). However, this model of style assumes that designers normally create maps with high scores on the six fitness dimensions of Sentient Sketchbook; subversive users showed that adaptive models may underperform when that assumption is not met and, presumably, would fare worse when user preferences are completely orthogonal to the prescripted fitness dimensions. Moreover, this designer model requires that users choose suggestions in order to learn their style, which may be problematic if those suggestions are not appealing in the first place. The model of process avoids this caveat by using the designer's manual edits as well as any chosen suggestions in order to provide focused, situational feedback. A potential caveat for the model of process is its dependency on base tiles; as shown in session 7, the model is not adapted (and provides random suggestions) while the user draws map sketches which do not contain bases. Design sessions where users only add bases in the end of their process, such as session 5 in Fig. 6.4, will not benefit at all from the designer model of process. Finally, the model of symmetry goals can identify the most common visual patterns in maps and dynamically switch between mappings from genotype to phenotype in order to ensure symmetries in generated suggestions. As the model of symmetry goals does not affect search, it can be used with any other designer model as well as with novelty search; its only possible

limitation is when map sketches have symmetries not included in any mapping, in which case the model would fail to detect them.

9.4 Summary

The experiments in this Chapter showed the potential of different methods of modeling a designer's preference, process or goals. Through simulations with artificial users or observations on past design sessions, Choice-Based Interactive Evolution proved able to recognize the patterns behind the user's choices — at least when such patterns somehow feature in the selection criteria of the user. The designer models of process and symmetry goals used previous designer sessions as a proof-of-concept experiment of their applicability and usefulness in recognizing the user's process and symmetry goals respectively; how these models accommodate the user will need to be tested via future user testing. This Chapter concludes the evaluation of Sentient Sketchbook, as the computational initiative has been tested on generative tasks including optimization, divergent search and personalization, as well as how it meshes with human creativity during design sessions. The next Chapter will enumerate other methods of integrating computational initiative, via variations of the algorithms presented in Chapter 4, in design tasks where human input is more or less prominent than in Sentient Sketchbook.

Chapter 10

Varying the Computational Initiative

The theory of mixed-initiative design and the methods introduced in Chapter 4 have been demonstrated in Sentient Sketchbook via extensive experiments documented in the previous Chapters. However, there are other ways of using the principles of mixed-initiative design, which provide more or less creative freedom to the human users as shown in Fig. 10.1. This chapter presents prototypes of other mixed-initiative tools which use the algorithms as well as the theories described in this thesis. Three different prototypes are described: Sentient World, which is an interface for iteratively refining virtual terrain described in Section 10.1, a prototype for evolving complete strategy game levels via rank-based interactive evolution in Section 10.2, and DeLeNoX in Section 10.3, which combines constrained novelty search with autoencoders able to change the characterization of distance. By varying the computational initiative, the need for human input also changes: in Sentient World human input is important both for initializing and for choosing from computer-generated output, in the rank-based interactive evolution experiment human input is lessened as it guides computer-generated strategy game levels towards desirable properties, while in DeLeNoX human input is replaced by computational feedback, with the computer acting both as a generator and as critic of its own work.

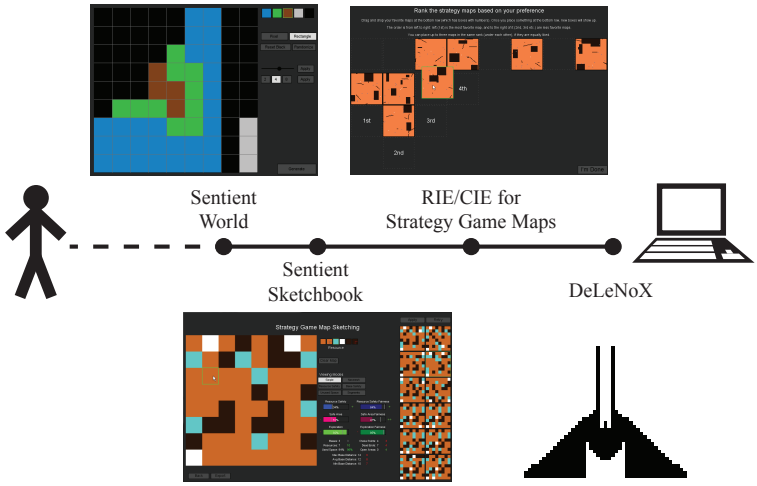


Fig. 10.1: A visualization of the different tools in this thesis, according to their balance between human and computational initiative. Obviously none of the tools have a "pure" human initiative, but Sentient World is closer to that as it needs it to create the initial draft. Sentient Sketchbook can autonomously generate content, but can also be "inspired" by the human sketch. Rank-based interactive evolution of complete strategy maps has a dominant computer initiative since the human merely guides evolution, while DeLeNoX requires no human input as it replaces the human critic with a computational critic.

10.1 Iterative Refining in Sentient World

Sentient Sketchbook allows the user to operate on map sketches which are transformed automatically by the computer into complete, detailed levels. This method allows for rapid prototyping; during such phases designers are most open to new ideas, so computer-generated suggestions may alter the designer's thought process. At other times, however, a designer needs to maintain creative control of their work. At first, this assertion may contradict the need for computer aid in the form of a mixed-initiative tool; however, based on the theory of sketching, the design process can be broken down into user-led sketching phases and computer-led refining phases. In Sentient Sketchbook, the refining phase from map sketch to final game level is purely computer-driven and obfuscated to the human user. Alternatively, the refining phase could be led by a computer and guided by a human user (via e.g. interactive evolution). An example use case follows with Sentient World, which allows users to control the initial seed of computer-generated terrain as well as progressively higher-detail versions of it. Other use cases could include evolutionary art tools where the human user provides a theme (either visual or semantic) and then the computer searches for elaborations of that theme in a fashion transparent to the artist, adding color or even sounds; early examples of such use cases are described in Section 11.2.5. Another example of mixed-initiative refining can be accomplished in engineering tasks, where the human designer begins with a set of minimal constraints on the desired solution (e.g. a machine part), observes the output of a constrained optimization run of the computational designer and — provided the current output is acceptable — progressively adds more constraints.

To address the requirement for designer control over generated content with minimal investment in human effort, *Sentient World* was developed for the creation of game maps and their terrain. Sentient World allows a designer to progressively add details to a rough sketch through the process of *iterative refining*. Iterative refining is accomplished by artificial neural networks (ANNs) trained via gradient search to conform to low-resolution sketches submitted by the designer; the infinite resolution potential of ANNs is then used to create higher-resolution maps which are submitted back to the designer to accept, reject, or edit. The iterative refining process is enhanced by neuroevolutionary novelty search which increases diversity in the pool of networks. Results obtained on a number of test maps show that the coupling of gradient and novelty search introduces divergent content without a significant computational overhead.

Sentient World attempts to combine novelty with gradient search via backpropagation in order to increase the representational power of ANNs without the uncontrollability often attributed to stochastic search algorithms. Additionally, Sentient World introduces the concept of iterative refining, where a human and a computer collaboratively add details to a rough concept sketch. While not yet tested with human designers, iterative refining is shown to secure the authorial control of human users as it largely conforms to initial designer sketches.

10.1.1 Methodology of Sentient World

The Sentient World tool is geared towards the iterative refining of maps, illustrated in Fig. 10.2. A user manually draws a low-resolution map; the height data from this map are used to train a number of neural networks previously optimized towards novelty via neuroevolution. Once

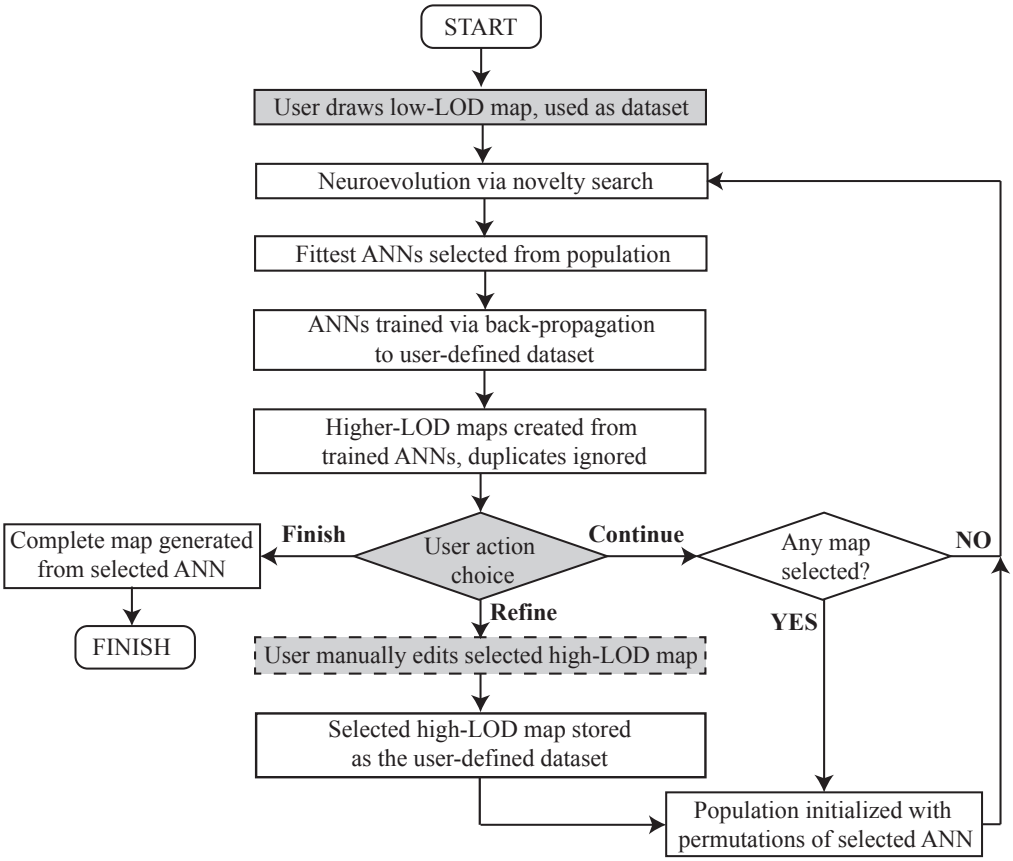
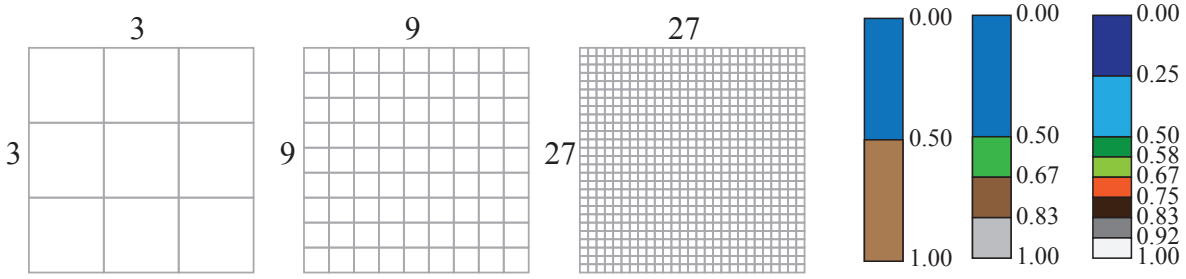


Fig. 10.2: An outline of the design process via the Sentient World tool, showing the different options for interaction (Finish, Continue and Refine) available to the user. Gray boxes represent user actions.

the networks' training (via gradient-search) is completed on the user-provided data, each ANN generates a map of higher detail which is presented to the user. The user can accept or manually edit the detailed maps, and resubmit them for further refining; the process terminates once the designer is content with their final map. The number of maps presented to the user is limited to eight in this study — despite the fact that evolution runs on a larger population — in order to reduce both the training time of ANNs and the cognitive load on the designer when inspecting the detailed maps.

Representation

The maps generated by Sentient World consist of a number of tiles, with each tile designating a specific *height zone*. The number of tiles (also termed *resolution*) and the number of height zones are interconnected and determine the *level of detail* (LOD) of the sketch (Fig. 10.3). A map sketch of any LOD can be encoded by a multi-layer ANN using a sigmoid activation function for all its nodes. The map is represented by an ANN in the following fashion: the normalized x, y coordinates of each tile's midpoint (red points in Fig. 10.4b) are used as input of the ANN, with the output being the tile's height value h. The output h belongs to a height zone i if $h \in [l_i, u_i)$, where l_i the zone's *lower bound* and u_i its *upper bound* (see Fig. 10.3b).



(a) Resolution LODs (b) Height zone LODs

Fig. 10.3: The Levels of Detail (LOD) used for the map sketches in Sentient World (from left to right: LOD of 1, 2 and 3), in terms of grid resolution and height zones. Numbers in Fig. 10.3a refer to the number of tiles available in each row and column for that LOD, while numbers in Fig. 10.3b refer to the lower and upper bounds (l_i, u_i) of each corresponding height zone.

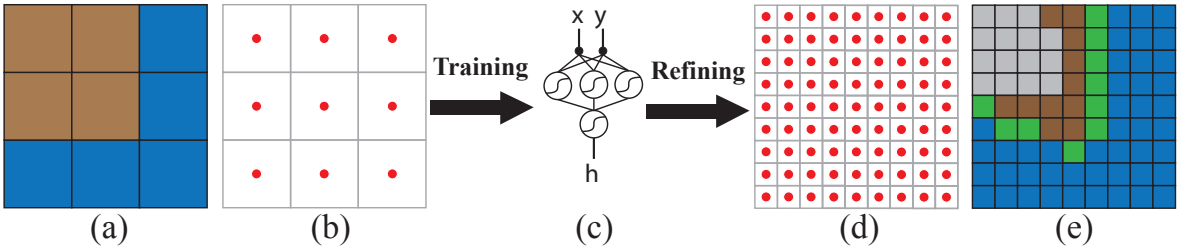


Fig. 10.4: An example of the iterative refining process. The initial sketch at LOD=1 drawn by the user (a) is used to create a dataset (b) using the height zone average (0.25 for water, 0.75 for land) at each tile's midpoint (red points). The dataset is used to train an ANN (c). Once training is complete, the ANN outputs the height values from the coordinates of a more detailed map (d) which are encoded into height zones of LOD=2, generating the refined map (e).

Iterative Refining

The key contribution of Sentient World is the process of iterative refining which allows the user, through interaction with the tool, to add an increasing number of details to a rough sketch. The process of iterative refining is currently accomplished through the training of multiple ANNs to conform with the rough sketch provided by the user. In order to increase diversity in the refined sketches and increase the ANNs' predicting abilities, a short evolutionary run optimizes these networks towards novelty and larger topologies.

ANN training: Iterative refining is accomplished through the training of multiple ANNs to approximate the patterns of the user-provided low-resolution sketch. In order to train these ANNs, the user sketch is converted to a dataset of input-output pairs. Inputs are the normalized x, y coordinates of the tile's midpoint (red points in Fig. 10.4b) and the desired output d is the tile's height zone average ($d = \frac{l_i+u_i}{2}$; where i is the tile's height zone and l_i, u_i are the zone's lower and upper bounds, respectively). The error e of the network, for actual output a and desired output d, is calculated as $e = \frac{1}{2}(d-a)^2$. Each network is trained via backpropagation [200] to minimize errors of the entire dataset, and training terminates either once all output values are within the desirable height zones or after 10^5 epochs. Backpropagation is carried out with non-batch weight updates and a learning rate of 0.1. Once

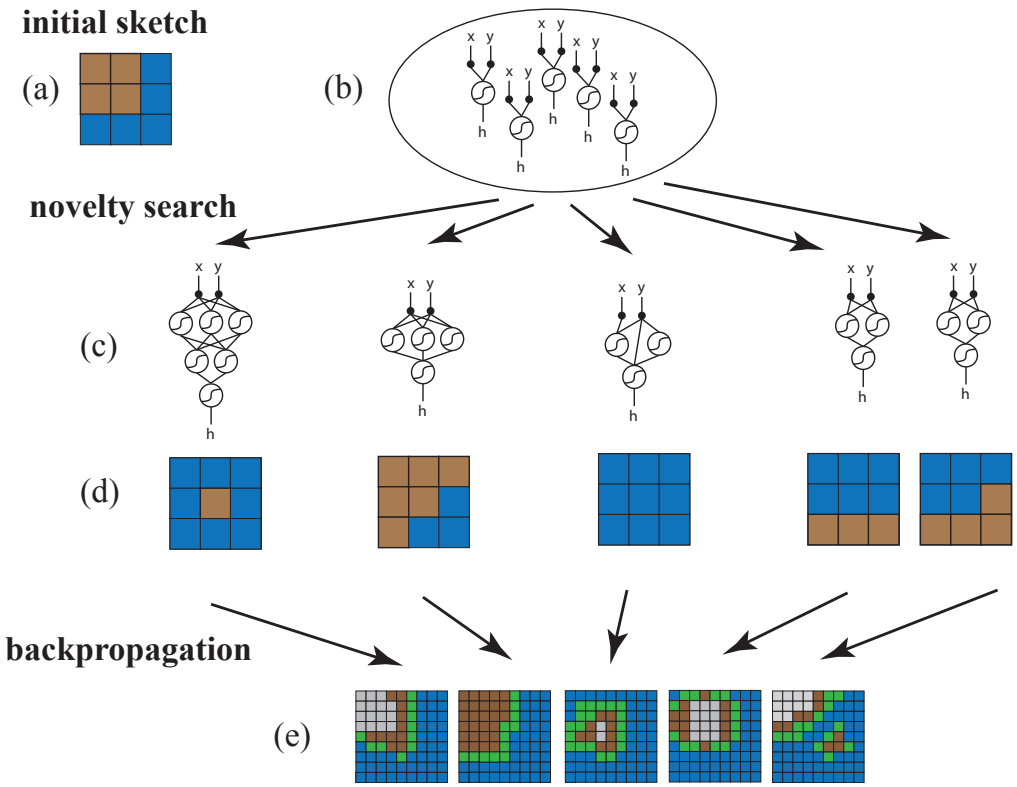


Fig. 10.5: A visualization of the impact of novelty search: an initial population (b) of similar, simple networks are evolved towards larger topologies (c) and dissimilar phenotypes (d), expanding the expressiveness of generated artifacts. Once evolution sufficiently explores the search space, gradient search constricts expressiveness towards artifacts that conform to the user-defined dataset of the initial sketch (a). The final, trained ANNs generate more diverse refined maps (e).

training is complete, the ANN is used to generate a more detailed map, increasing both the resolution LOD and the height zone LOD by one step; thus the network has a larger number of coordinates for inputs, while its h outputs correspond to more precise height zones (see Fig. 10.4e).

Neuroevolution: As the maps' resolution increases, the dataset of input-output pairs becomes more complex and requires a larger network to approximate. Additionally, as the user is presented with various detailed map suggestions during the iterative refining step, varying the topology and initial weights of the networks prior to training is likely to create more variation in the final results. For these two reasons, a short evolutionary run optimizes the ANNs towards novelty [132]. Evolution is carried out via neuroevolution of augmenting topologies (NEAT), which has a chance of increasing the number of layers, the number of nodes, and the number of links of the neural networks in the population [233]. Following the novelty search paradigm [132], evolution optimizes networks towards maximizing the novelty score ρ of eq. (3.3). The distance characterization of the networks, i.e. $dist(i,j)$ in eq. (3.3), is calculated as:

$$dist(i,j) = \frac{1}{T}\sum_{t=1}^{T} |h_i(t) - h_j(t)| \tag{10.1}$$

where T is the number of tiles of the encoded map on the same resolution, i.e. (d) in Fig. 10.5 and $h_i(t)$ is the h value at tile t's midpoint of the map encoded by network i.

Evolution is carried out for 20 generations on a population of 20 individuals, with the 5 fittest networks per generation stored in an archive of novel individuals and the closest 5 individuals considered when evaluating ρ. If no prior refining has occurred during the current session, the initial population in the evolutionary run consists of fully connected networks with randomly initialized weights and one hidden layer with four nodes. If a map and its encoding network has already been selected during previous refining steps, the initial population in the evolutionary run consists of mutations of the selected network, thus preserving its more elaborate topology. In order to bypass the problem of recombining networks of different topologies, evolution takes place only via mutation by adding a new node (10% chance) or a new link (15% chance) to the network, or otherwise modifying the weight of one randomly selected link. The selection of individuals for mutation is made via a fitness-proportionate roulette-wheel scheme. Once evolution is terminated, the eight fittest networks are selected and trained using backpropagation, as described above (see Fig. 10.5).

User Interface

Sentient World aims to assist the user both in the generation and in the refinement of terrain models; the former is accomplished through a simple map editor and the latter through the presentation of maps of higher detail. The map editor screen (Fig. 10.6a) allows the user to paint the map's tiles using brushes for different height zones. In addition to the height zones in Fig. 10.3b, the user can designate black tiles in the map, which act as wildcards and can be of any height. Black tiles are not included in the dataset for training the ANNs in the iterative refining process.

The map selection screen (Fig. 10.6b) allows the user to inspect the refined maps generated according to Section 10.1.1. The interface allows up to eight maps to be shown, although identical maps are omitted. The user may select a single map among presented ones, in which case the following actions become available:

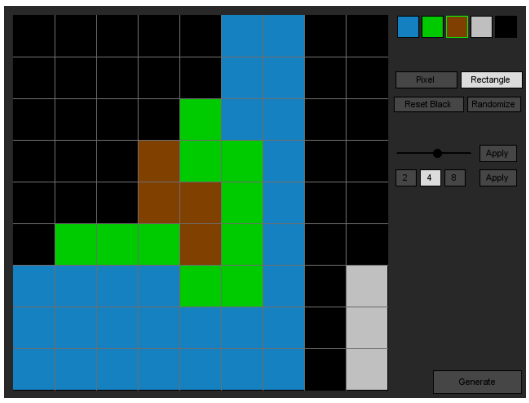
Continue which re-runs the generative algorithms on the current level of detail, with an initial population seeded by the ANN of the selected map.

Refine which runs the generative algorithms on the next level of detail, using the height data of the selected map as the training dataset.

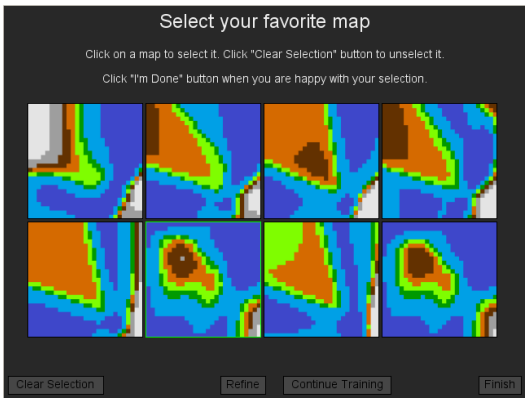
Edit & Refine which allows the user to load the selected map in the map editor and make manual adjustments; the modified map is used as the training dataset to generate the maps of the next level of detail.

Finish which uses the ANN of the selected map to generate the full resolution heightmap (Fig. 10.6d), allowing for further calculations and for exporting to a file.

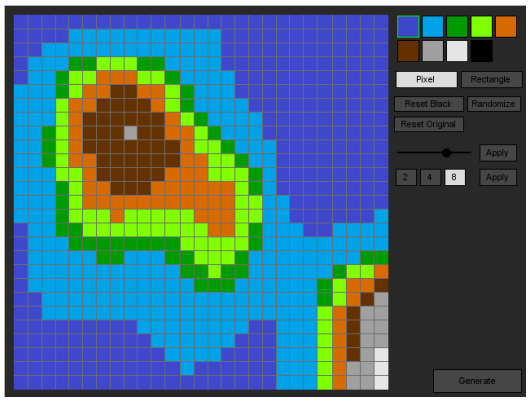
If the user selects no map among those presented, they have the option to re-attempt the map generation process (on the current level of detail) with a new initial population. The available user actions are also shown in Fig. 10.2; manual editing is an optional component to the process of refining, and appears in a dotted outline.



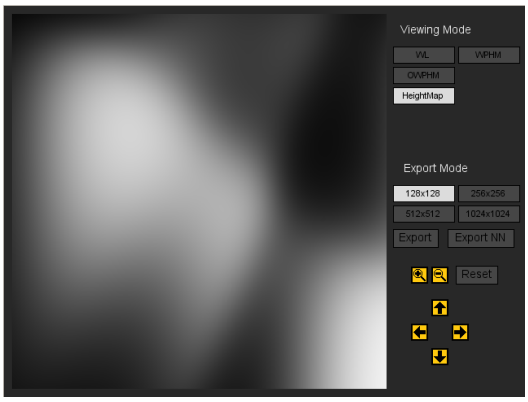
(a) Sketching interface while the user draws a new map sketch (with LOD of 2).



(b) Selection interface while the user selects a higher resolution map sketch (with LOD of 3) to refine the sketch from Fig. 10.6a.



(c) Sketching interface while the user manually refines the selected map from Fig. 10.6b.



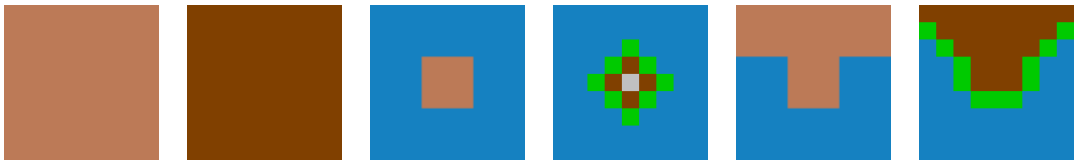
(d) The full resolution of selected map from Fig. 10.6b, displayed as a heightmap.

Fig. 10.6: The User Interface for the Sentient World tool.

10.1.2 Experiments with Iterative Refining

To evaluate the potential of the iterative refinement approach and the efficacy of the proposed method, a number of sample maps are refined through the algorithm described in Section 10.1.1. These sample maps (shown in Fig. 10.7) have three distinct patterns — i.e. Land (L), Island (I) and Shore (S) — on two LODs. The maps were selected for their diversity — e.g. the patterns in map L1 are much simpler and easier to learn than those of map I2. In a simulated run of Sentient World, the above maps are refined by eight ANNs, since that is the number of presented maps in the Sentient World interface. The impact of gradient and novelty search is tested via two experiments: in the first, backpropagation (BP) is used to train eight randomly initialized fully-connected ANNs with a hidden layer of four nodes. In the second, backpropagation is used to train the eight fittest ANNs evolved via novelty search from a population of 20 ANNs for 20 generations; the initial population's ANNs has the same topology as the randomly initialized ANNs of the first experiment.

The performance measures considered in this study include the *runtime*, derived from an



(a) L1 (b) L2 (c) I1 (d) I2 (e) S1 (f) S2

Fig. 10.7: The six initial maps which will be refined by the Sentient World tool. Maps L1, I1, S1 have LOD=1 while maps L2, I2, S2 have a similar form to their respective coarse counterparts, but with LOD=2.

Template Map	BP		BP with novelty search		
	ANN time (s)	Average Distance	Evolution time (s)	ANN time (s)	Average Distance
L1	0.14	0.103 (0.023)	0.15	0.16	0.122 (0.031)
L2	0.23	0.013 (0.002)	0.33	0.58	0.028 (0.005)
I1	7.89	0.036 (0.008)	0.08	4.18	0.036 (0.009)
I2	1183.15	0.046 (0.004)	0.27	684.08	0.046 (0.005)
S1	6.81	0.012 (0.006)	0.15	2.67	0.029 (0.019)
S2	569.25	0.016 (0.004)	0.20	680.86	0.031 (0.004)

Table 10.1: Comparison of the refinement processes for different template maps, using backpropagation (BP) with and without novelty search — i.e. BP trained on random ANNs vs. BP trained on initial ANNs guided by novelty search. Running times of the ANN training (ANN time) and evolution (Evolution time) and the average distance between maps are the performance measures considered.

Intel i7 at 2.10GHz with 8 GB of RAM, and the *average distance* between the refined maps; significance is tested through standard t-tests (significance is 5% in this set of experiments). Average distance $\bar{d}$ is calculated as:

$$\bar{d} = \frac{1}{P(P-1)} \sum_{i=1}^{P} \sum_{\substack{j=1 \\ j \neq i}}^{P} dist(i,j) \tag{10.2}$$

where P is the number of presented maps (P=8 in this set of experiments) and $dist(i,j)$ is the distance metric of (10.1) but calculated on the refined maps, i.e. see Fig. 10.5e.

The results of the different maps' refinement for the two approaches are shown in Table 10.1 containing the mean values collected across 20 individual runs, with standard deviation among runs shown in parentheses. Figure 10.8 displays the refined maps encoded by the trained ANNs of the most successful run in terms of average distance. We observe that for L and S patterns novelty search succeeds in significantly increasing the diversity of generated maps. Inspecting the most successful artifacts in Fig. 10.8, backpropagation combined with novelty search creates far more visually interesting and complex maps for L1 and S2, compared to the repetitive patterns when applied on simpler networks. For the Island patterns (I1, I2), backpropagation with and without novelty search generates maps which conform equally well to the user-defined dataset; however, novelty search does not enhance the diversity of these particular map patterns. With respect to computational time, the larger topology of networks evolved via NEAT increases the training time for backpropagation in the L and S patterns.

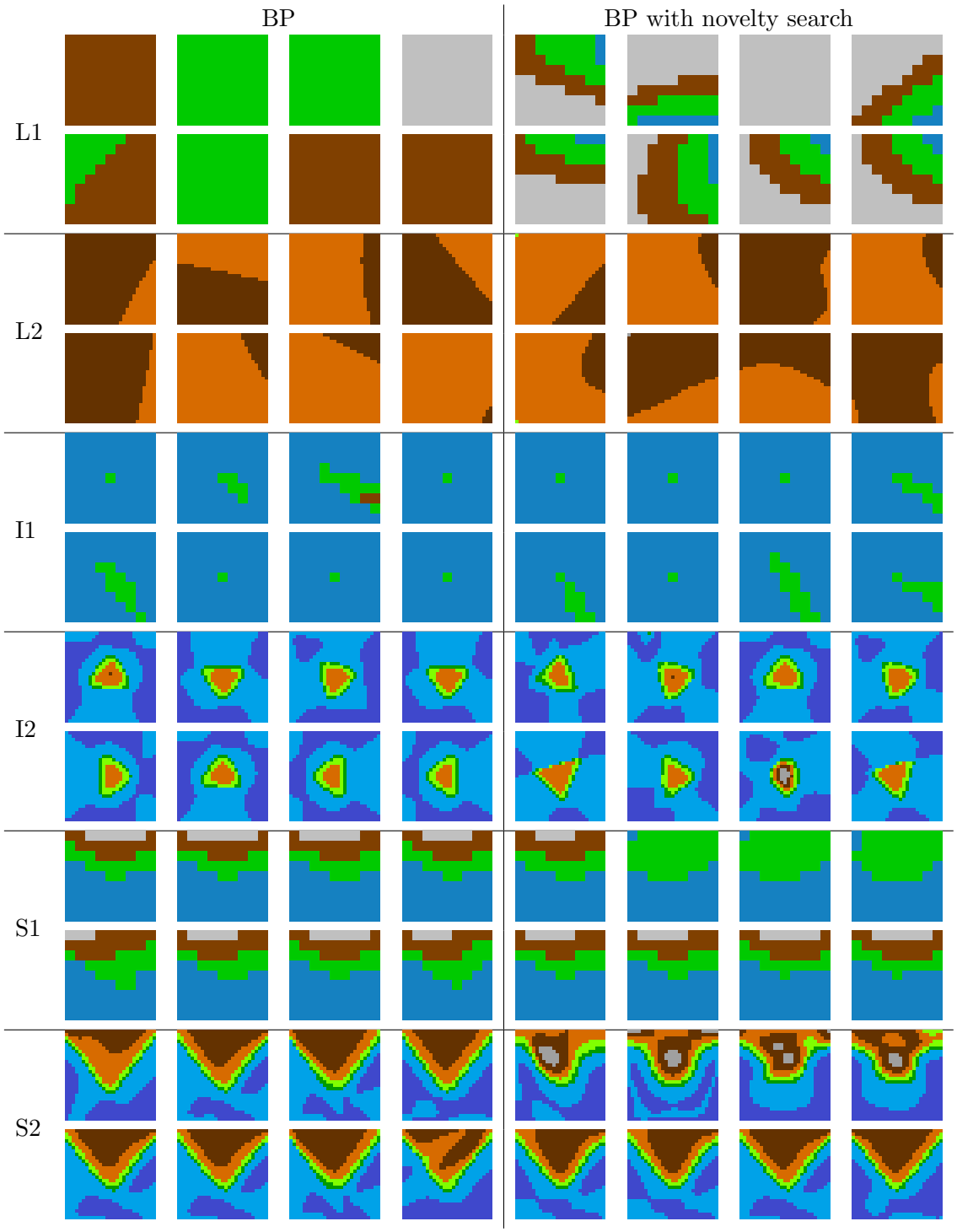


Fig. 10.8: Comparison of the eight refined maps trained via backpropagation (BP) with and without novelty search, for each template map. The maps are collected from the run with the highest average distance.

Small networks appear able to easily encode the simple patterns in the L and S datasets; backpropagation on random small networks can therefore quickly learn such patterns, but creates visually uninteresting results. On the other hand, there is a surprising decrease in the computational time required to train the larger evolved networks for the Island patterns, compared to the training time for random small networks. While not necessarily creating more diverse results, novelty search enhances backpropagation for such maps since the larger networks are able to learn these complex patterns faster than the small random networks. The presented sample maps showcase that novelty search can contribute — with minimal computational overhead compared to that of backpropagation — to faster training and more diverse results, although one often at the expense of the other.

10.1.3 Discussion

The combination of stochastic and gradient search for the generation of novel maps conforming to designer intentions has shown promising results on the sample maps used to evaluate the proposed algorithm for iterative refining. The most important future research step is to test the tool with human designers, by collecting usability metrics, questionnaires, or verbal feedback and by measuring the impact of back-propagation and novelty search. A concern for the presented approach is the required runtime between iterations of sketching and refining. In larger resolutions and complex patterns, the size of the dataset makes training slow (e.g. 20 minutes for sample map I2). It is not realistic for a computer-aided design tool promoting human-machine dialog to require such long periods of inaction from the human user. Future work will address the issue by reducing the number of maximum epochs before training terminates, by training each network in its own thread running on parallel (similar to Sentient Sketchbook) and by showing the refined maps while training is under way, allowing users to terminate training prematurely if they find the maps interesting. An alternative solution for reducing training time in high-detail sketches is to take advantage of problem decomposition: an ANN can be pretrained to conform to a low-detail version of the user-defined map before learning the high-resolution map patterns.

Although gradient search helps preserve designer intentions during the generative process, it requires "close" supervision which may not be appropriate in cases where the designer does not wish to specify all the map details. While designers often have specific ideas on the heightmap of their terrain, other properties such as temperature or humidity are much more difficult to manually specify and would increase the cognitive load on the side of the designer. Future work will explore the unsupervised search of patterns via neuroevolutionary algorithms such as NEAT [233], in circumstances where the designer provides high-level specifications such as vegetation on areas of the map and the algorithm optimizes the underlying conditions (temperature, humidity, and soil consistency maps) for the satisfaction of those specifications. Combining neuroevolution with constrained optimization has been quite successful for the generation of content which satisfies strict design requirements [136, 138]. Additionally, the elevation patterns stored in the trained ANNs can be used as scaffolds [105] for generating complementary maps (such as vegetation or temperature maps) through the use of CPPNs [231].

The visual appearance of the final maps is limited to both the representation employed and the training process followed. The sigmoid functions used in the ANNs often generate very "smooth" landscapes, with rounded shorelines and smooth elevations. More interesting features

could be added via noise, but the randomness would remove the controllable aspect of this tool. The use of other activation functions might create more interesting shapes, but such networks can only be trained through evolution, such as CPPN-NEAT [231]. Otherwise, fast and deterministic erosion algorithms [177] could be applied to the heightmaps for a more realistic appearance.

10.2 Adaptive Models of Strategy Game Level Quality

Apart from adapting the suggestions of Sentient Sketchbook to the preferences of the tool's current user, the adaptive models in Section 4.3 can be used to interactively evolve complete strategy game levels. Although this method affords a lower degree of designer control than the manual edits of Sentient Sketchbook, it allows larger-scale game levels to be created with little designer effort, i.e. via a few interactions. The experiments described in this Section differ from the designer models implemented for Sentient Sketchbook since user control is less direct (unlike suggestions in Sentient Sketchbook which start from a human-authored sketch), the population is not replaced with mutations of the preferred map every time a map is chosen, and there is only one population unlike designer models of Sentient Sketchbook where the weighted sum used for style and process was only used to calculate the fitness of feasible individuals.

Unlike standard interactive evolution, the adaptive models of Section 4.3 use a number of domain-specific predefined fitness dimensions whose impact is adjusted according to content rankings by users. These fitness dimensions differ from those of Sentient Sketchbook and some of them pertain to purely visual or stylistic properties. In this particular study, Rank-Based Interactive Evolution (RIE) is used to generate personalized strategy game maps and its optimization behavior is tested, in a controlled experiment with artificial users, against Choice-Based Interactive Evolution (CIE) and variants of traditional Interactive Evolution.

10.2.1 Strategy Map Generation

For the purposes of this study, complete overland maps are being evolved to maximize several aesthetic and gameplay properties important to strategy games. These maps are abstractions of levels used in successful strategy games such as *Starcraft* (Blizzard 1998). Each map has a size of 64 by 64 tiles which can be either passable or impassable, two player bases, ten resources and ten regions of impassable tiles (see Fig. 10.9). Like strategy game levels in Sentient Sketchbook, the map layout assumes that each player starts at one of the bases and gathers resources to produce units; units move through passable tiles in order to attack the opponent's base.

Each map is encoded in an array of 74 real numbers within $[0, 1]$. Each player base or resource is encoded in two parameters, corresponding to the x and y coordinate if multiplied by the map's width and height respectively. Each impassable region is encoded in five parameters and can be a line or a rectangle, diversified by a parameter t; a rectangle ($t < 0.5$) is represented by the coordinates of its diagonal corners, while a line ($t \geq 0.5$) by the coordinates of an origin point, an angle and a distance.

Fig. 10.9: A sample map generated by the algorithm: light areas represent passable tiles, dark areas represent impassable tiles, blue rhombi represent resources and the two white circles represent the players' bases.

The 74 parameters which represent the game map are optimized by a genetic algorithm [85]. The gene with the highest fitness is transferred to the next generation, while suitable parents are selected using fitness-proportional roulette wheel selection; the parameters of selected parents are recombined via 2-point crossover. An offspring of two parents has a mutation probability of 1%, while a single parent has a 5% chance of being mutated and copied to the next generation. When mutation is applied, the parameter array's order may be reversed (15% chance) or 1 to 5 parameters are modified by a random number following a normal distribution with 0 mean and 0.33 standard deviation — results are bound within [0,1]. Mutation favors local search via small changes, but the occasional drastic change is permitted in order to hinder premature convergence.

The maps are evaluated on gameplay and aesthetic features; gameplay features are largely inspired by previous work on evolving strategy game maps [245], while aesthetic features are inspired by studies on visual perception [193, 2], previous work on computational models of visual aesthetics [138, 137], and popular strategy game design patterns such as "King of the Hill" map templates in *Age of Mythology* (Ensemble Studios, 2002). For the purposes of this study, ten fitness functions are defined in eq. (10.3)-(10.12) falling under the following categories:

Spatial Navigation: These fitness dimensions evaluate the passable space of the map, and focus on navigation around and between player bases: *base distance* fitness (f_{BD}) in eq. (10.3) rewards long distances between bases, *base space* fitness (f_{BS}) in eq. (10.4) rewards passable areas around a base and *choke point* fitness (f_{CP}) in eq. (10.5) rewards narrow passes between

bases.

$$f_{BD} = min\left\{1, \frac{d_B}{w_M + h_M}\right\} \tag{10.3}$$

$$f_{BS} = \frac{\bar{a}}{C_1^2} \tag{10.4}$$

$$f_{CP} = \begin{cases} \frac{1}{2}\left(1 - \frac{w_C}{C_1} + h_A\right) & \text{if } w_C < C_2 \\ C_3\left(1 - \frac{w_C}{d_M}\right) & \text{if } w_C \geq C_2 \\ 0 & \text{if } w_C = 0 \end{cases} \tag{10.5}$$

where d_B is the distance between the two player bases; w_M, h_M are the map's width and height in tiles, respectively; $\bar{a}$ is the average number of passable tiles within a C_1 by C_1 tile grid centered around every player base ($C_1 = 7$ in this study); w_C is the smallest width on the shortest path between the two player bases; $d_M = (w_M + h_M)/2$ is the map's diagonal; C_2 is the maximum width of a chokepoint and C_3 limits the reward of narrow paths which do not constitute choke points ($C_2 = 10$ and $C_3 = 0.01$ in this study); h_A is the heuristic of the alternate path, assuming the chokepoint was closed: $h_A = min\{1, (d_B - d'_B)/d_M\}$, where d'_B is the shortest path between bases if the chokepoint becomes impassable.

Resource Distribution: These fitness dimensions evaluate how resources are placed on the map, in terms of player bases and aesthetic criteria: *nearby resources* fitness (f_{NR}) in eq. (10.6) rewards resources near both player bases, *nearby resource balance* fitness (f_{RB}) in eq. (10.7) rewards equality between player bases' nearby resources and the aesthetically-oriented *king of the hill* fitness (f_{KH}) in eq. (10.8) rewards resources placed in the central area of the map.

$$f_{NR} = \frac{r_{N,1} + r_{N,2}}{2r} \tag{10.6}$$

$$f_{RB} = \frac{|r_{N,1} - r_{N,2}|}{r} \tag{10.7}$$

$$f_{KH} = \frac{r_M}{r} \tag{10.8}$$

where r the number of all resources in the map; $r_{N,i}$ is the number of resources within 16 tiles from the base of player i; r_M the number of resources in a square of width $w_m/2$ and height $h_m/2$ in the center of the map.

Aesthetic Appearance: These fitness dimensions evaluate aesthetic properties of the maps; since the maps are presented as 2D images, it is important to consider certain visual cues on which humans evaluate artifacts universally [193, 2], focusing on the *balance* and *division* of impassable regions. The *unconnected segments* fitness (f_{US}) in eq. (10.9) rewards the presence of distinct "chunks" of impassable tiles, *Symmetry (X and Y)* fitnesses (f_{Sx} and f_{Sy}) in eq. (10.10) and eq. (10.11) reward balance of impassable tiles between the left and right halves and between the top and bottom halves respectively, and *Quadrant Symmetry* (f_{QS})

in eq. (10.12) rewards balance of impassable tiles in neighboring map quadrants.

$$f_{US} = \frac{I_U}{I} \tag{10.9}$$

$$f_{Sx} = 1 - \frac{1}{I}|I_T - I_B| \tag{10.10}$$

$$f_{Sy} = 1 - \frac{1}{I}|I_L - I_R| \tag{10.11}$$

$$f_{QS} = 1 - \frac{1}{I}\sum_{i=2}^{3}(|I_{Q_i} - I_{Q_1}| + |I_{Q_i} - I_{Q_4}|) \tag{10.12}$$

where I the number of impassable tiles in the map; I_U the number of unconnected impassable chunks calculated using a 4-direction flood fill algorithm on all impassable tiles; I_T, I_B, I_L, I_R the number of impassable tiles in the top, bottom, left and right halves of the map respectively; I_{Q_i} is the number of impassable tiles in quadrant i of the map.

10.2.2 Preference Modeling

According to the adaptive model methodology of Section 4.3, the dimensions of strategic and visual quality of Section 10.2.1 can be aggregated into a weighted sum in order to embody a user's preference model. Using this weighted sum, or preference score (F), as the objective function of a genetic algorithm, this method evolves maps towards high scores in many different dimensions.

This study uses both Choice-Based Interactive evolution (CIE) as described in Section 4.3.1 and Rank-Based Interactive evolution (RIE) as described in Section 4.3.2. The weights of the 10 fitness dimensions described in eq. (10.3)–(10.12) are adjusted in every iteration via equations (4.1) for CIE and (4.2) for RIE, with a weight update step $\alpha = 0.01$. The weights are adjusted until the selected map has the highest preference score F among those presented; the adjustment process can be prematurely terminated if the preference score difference between the highest scoring map and the selected map starts to increase or after $3 \cdot 10^5$ weight updates. The final adjusted weights are divided by $\sum_i |w_i|$ resulting in normalized weight values.

10.2.3 Experiments

Similar to Chapter 9, a number of artificial users are designed to simulate human selection of content. These controllable, noise-free users can assess each adaptive model's ability to predict their personal taste based on their selections. This experiment does not assess the ability of the models to capture human taste. The experiment instead assesses the efficiency of capturing user criteria and of optimizing content towards them, assuming that the fitness functions included in such a model are representative of user criteria for evaluating content.

Using the criteria of artificial users, two measures of performance are considered for the two-step adaptation process: a) the *population's best individual* or real best individual (R) as evaluated by a user's taste, which demonstrates how well evolution can optimize to the user's taste, and b) the *estimated best individual* (E) as evaluated by the preference model, which

demonstrates how well the preference model can predict the user's taste. Experiments in this section compare the following algorithms:

UB The *Upper Bound* (or Bound) of the adaptive approaches assumes full knowledge of the user's taste; it optimizes a population to the fitness used to select content like a standard genetic algorithm. For UB the real best individual is identical with the estimated best individual, so the latter is omitted.

RIE The proposed approach to rank-based interactive evolution presented in Section 4.3.2, using an aesthetic model derived from the full ordering of eight sample maps according to the fitness used to select content.

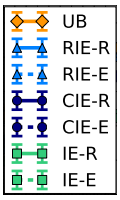
CIE The existing approach to choice-based interactive evolution presented in Section 4.3.1, using an aesthetic model derived from the choice of the single fittest map among eight sample maps according to the fitness used to select content.

IE A traditional interactive evolution approach, driven by the rankings of eight sample maps according to the fitness used to select content. In line with literature on interactive evolution [242], artificial user ratings are discretized to fractions of 8: the fittest individual has a fitness of $\frac{8}{8}$ and the least fit individual a fitness of $\frac{1}{8}$. Since the eight maps presented to the artificial user are a small subset of those in the population, this IE approach predicts fitness scores [242] of unpresented individuals based on their genotypical distance from the ranked individual: any individual i in the population is assigned a fitness score f_i of $f_i = d_{i,p}q(p)$ where p is the presented individual with the smallest distance between genotypes ($d_{i,j}$) among presented individuals j and $q(p)$ is the discrete fitness assigned by the artificial user to p; $q(p) = \frac{k}{8}$ where k is the rank of individual p according to the fitness used to select content. Note that while a similar IE model was used for experiments with Sentient Sketchbook in Section 9.1, the genotypical difference $d_{i,j}$ in Sentient Sketchbook is a one-to-one match with phenotypical difference; due to the indirect encoding of strategy game maps, there is no such match with the phenotype in these experiments.

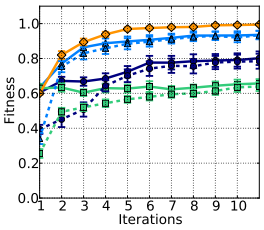
Presented experiments are made on a population of 100 individuals, which evolve for 100 generations divided into *iterations*. In each iteration eight maps are selected from the population and presented to the artificial user who either ranks them or selects their favorite among them. Presented maps include the estimated best individual and the seven individuals with the largest genotypical distance ($d_{i,j}$) from the estimated best individual. Once the preference model is adjusted, the current population is evolved for 10 generations and a new iteration begins. Every new iteration updates the preference model but continues evolution on the previous iteration's population. After a sequence of 10 iterations, content generation is expected to more closely match the user's taste; in the case of artificial users, their taste is the fitness used for selecting content. Mean values and standard error values are collected from 20 independent runs of each approach.

User preference based on a single fitness dimension

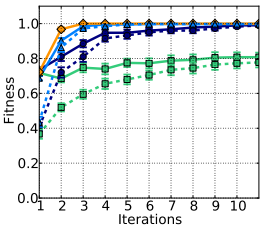
The simplest artificial user selects content according to one of the fitness dimensions defined in eq. (10.3)-(10.12); these users are identified as A_f where f the fitness notation (for instance,



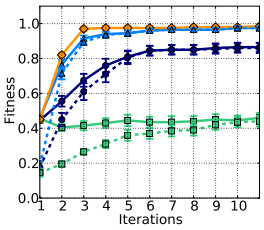
(a) Legend.



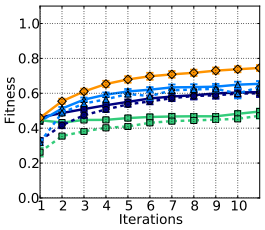
(b) Progress of f_{BD} for Artificial User A_{BD}.



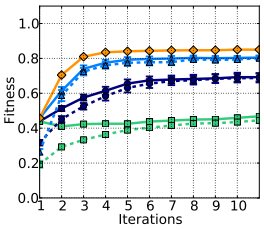
(c) Progress of f_{NR} for Artificial User A_{NR}.



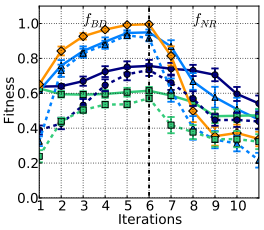
(d) Progress of f_{KH} for Artificial User A_{KH}.



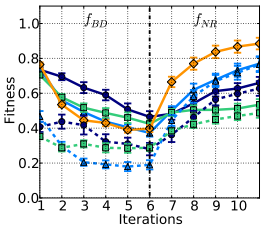
(e) Progress of f_{C1} for Artificial User A_{C1}.



(f) Progress of f_{C2} for Artificial User A_{C2}.



(g) Progress of f_{BD} for Artificial User A_{alt}.



(h) Progress of f_{NR} for Artificial User A_{alt}.

Fig. 10.10: Optimization progress for evolutionary runs with an artificial user selecting based on different fitnesses (single, composite and alternating). Solid lines represent the individual with the highest stated fitness in the population (R), and dotted lines represent the preference models' estimated best individual's stated fitness dimension (E). Error bars represent the standard error for 20 individual runs.

A_{BD} selects content according to f_{BD}). The optimization progress for three sample artificial users (A_{BD}, A_{NR}, A_{KH}) is included in Fig. 10.10; a comparison of the final best individuals of all artificial users can be found in Table 10.3.

User preference based on composite fitness functions

In order to simulate a more complex model of human taste, artificial users in this experiment take into account multiple criteria, represented as a weighted sum of multiple fitness dimensions from eq. (10.3)-(10.12). Two sample artificial users are tested (A_{C1}, A_{C2}), selecting content according to the fitness functions f_{C1} and f_{C2} respectively:

- $f_{C1} = 0.5f_{BD} + 0.3f_{NR} + 0.2f_{KH}$
- $f_{C2} = 0.2f_{BD} + 0.3f_{NR} + 0.5f_{KH}$

The choice of f_{C1} and f_{C2} is expected to highlight the limitations of the weighted sum approach for non-dominated multiobjective evolution as well as the impact, if any, of the adaptive models to this particular limitation. Both of these functions combine f_{BD} (which rewards distant player bases), f_{NR} (which rewards many resources near both player bases) and f_{KH} (which rewards resources in the map's middle): f_{BD} is in conflict with f_{NR} since player bases cannot simultaneously be far apart from each other and near all resources, while both f_{BD} and f_{NR} are independent of f_{KH}. These experiments also explore how different weights of the fitness dimensions of f_{C1} and f_{C2} affect the optimization process.

	UB	RIE	CIE	IE
A_{C1}				
f_{C1}	0.75 (0.02)	0.65 (0.02)	0.61 (0.02)	0.46 (0.01)
f_{BD}	0.88 (0.04)	0.66 (0.06)	0.57 (0.06)	0.61 (0.03)
f_{NR}	0.48 (0.03)	0.45 (0.06)	0.58 (0.06)	0.37 (0.01)
f_{KH}	0.81 (0.03)	0.85 (0.03)	0.71 (0.04)	0.22 (0.03)
A_{C2}				
f_{C2}	0.85 (0.01)	0.80 (0.01)	0.69 (0.02)	0.46 (0.02)
f_{BD}	0.28 (0.01)	0.23 (0.02)	0.37 (0.06)	0.26 (0.03)
f_{NR}	1.00 (0.00)	0.95 (0.03)	0.81 (0.05)	0.65 (0.03)
f_{KH}	0.99 (0.01)	0.94 (0.02)	0.75 (0.04)	0.44 (0.03)

Table 10.2: The contributing fitness scores of the real best individual after 10 iterations for artificial users A_{C1} and A_{C2}. Standard error for 20 runs is included in parentheses.

Figures 10.10e and 10.10f show the progress of the two adaptive models with artificial users selecting content based on a linear combination of three fitness scores; a comparison of the final best individuals of all artificial users can be found in Table 10.3. Table 10.2 contains information about the individual fitnesses of the best individuals after 100 generations.

Dynamic user preference based on variant fitness values

In order to simulate the levels of dynamicity and stochasticity existent in human nature (and preference), an experiment was conducted with an artificial user that changes the fitness used to select content after a number of iterations. The presented artificial user (A_{alt}) selects content according to f_{BD} for the first five iterations and according to f_{NR} for the next five iterations. The two fitnesses are conflicting since maps with high f_{NR} values must have both player bases near every resource, meaning that the bases cannot simultaneously be far apart from each other. Figures 10.10g and 10.10h show the progress of the two fitnesses f_{BD} and f_{NR} both while they are the user's selection criterion and when they are not. A comparison of the best individuals according to f_{NR} after 10 iterations can be found at Table 10.3.

Conclusions of the Experiment

Table 10.3 assesses the ability of the different algorithms to optimize their individuals over 10 iterations, using the UB algorithm to derive the algorithms' performance benchmarks.

As expected, the Upper Bound (with full knowledge of the agent's selection scheme) outperforms both CIE and RIE, especially in the early stages of evolution when the interactive approaches are still approximating the user's taste. Figures 10.10b-10.10d show how UB reaches optimal values within a few iterations. There are however fitnesses which rely on very specific map patterns (such as A_{CP}) or try to combine conflicting criteria (such as A_{C1} and A_{C2}): in such cases UB struggles to find fit individuals, and its optimization is slow (shown by the low best scores in Table 10.3). On the other end, there are fitnesses (such as A_{BS} or A_{RB}) that have optimal individuals already in the initial population, and their optimization is effortless.

	P_{th}	A_{BD}	A_{BS}	A_{CP}	A_{NR}	A_{RB}	A_{KH}	A_{US}	A_{Sx}	A_{Sy}	A_{QS}	A_{C1}	A_{C2}	A_{alt}
UB	Best	1.00	1.00	0.75	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.85	0.88	1.00
RIE	50%	20	20	7	20	20	20	19	20	20	20	20	20	19
	70%	19	20	4	20	20	20	9	20	20	20	17	20	13
	90%	15	20	1	20	20	20	2	20	20	20	1	12	8
	100%	12	20	0	19	20	15	2	11	12	0	0	0	3
CIE	50%	20	20	6	20	20	20	8	20	20	20	20	19	19
	70%	12	20	2	20	20	19	7	20	20	20	9	16	10
	90%	8	20	0	20	20	10	2	20	20	20	3	1	2
	100%	7	20	0	17	20	6	1	16	16	1	0	0	0
IE	50%	18	20	2	20	20	11	8	20	20	20	16	15	13
	70%	6	20	0	18	20	1	0	20	20	19	0	2	3
	90%	0	20	0	6	20	0	0	20	20	6	0	0	1
	100%	0	20	0	1	20	0	0	11	11	0	0	0	0

Table 10.3: Number of runs (out of 20 trials) that the population's best individual for the compared algorithms reached the stated performance threshold, P_{th}, defined as the percentage of the best fitness score found by UB (shown in the first line).

Interactive evolution does not perform well in most of these experimental settings, primarily because its optimization progress is sensitive to the initial selection of presented maps. While CIE and RIE select their estimated best individual and seven maps of the most different genes to it, interactive evolution does not have an estimated best individual before the first iteration; the initial presented maps are selected based on their genetic distance from other individuals in the population. There is no guarantee, therefore, that the initial presented maps will have any of the features rewarded by the fitnesses of the artificial user. This often results in a drop in the maximum fitness score in the population after the first iteration, as the highest rated map may still be much worse than unpresented maps; such unpresented maps receive lower fitness scores and are not preferred for selection. Note that unlike experiments with Sentient Sketchbook in Section 9.1, where the initial population before the 1st iteration were mutations of the same empty map, the initial population here is randomly initialized; moreover, the population after each iteration is not replaced by mutations of the selected map, which likely had a positive effect on the fitness and similarity of resulting populations in the experiments of Section 9.1.

Looking at the results of IE more closely, in cases where a random mutation is likely to discover fit individuals (such as in f_{NR}), IE manages to overcome the initial drop in fitness scores and increase the population's fitness. This is more pronounced for f_{BS} and f_{RB}, since many optimal individuals exist already in the initial population and are likely to be presented; the population quickly converges to these optimal individuals, and IE performs as well as the other approaches. However, when highly-fit solutions are difficult to find (such as for A_{CP}, A_{C1} and A_{C2}), IE hardly optimizes its initial population and the best fitness scores rarely reach over 50% of UB's performance. Finally, IE fails to handle shifting objectives in A_{alt}, primarily because when the shift in user taste changes, the population is already converged to maps with high fitness in f_{BD} but low fitness in f_{NR}. This accentuates the sensitivity of IE to presented individuals: at the 6th iteration for A_{alt} the presented individuals will inevitably have low fitness in f_{NR}, causing IE's sub-par performance with A_{alt} compared to A_{NR}.

Comparing the progress of the rank-based and choice-based approaches, the large amount of information contained in the absolute ranking of content for the former approach results in

a much more accurate adaptation of the preference model in early stages of evolution. This is evident in the estimated best individuals' progress (dotted lines) in Fig. 10.10; it is even more pronounced in Fig. 10.10h, where the switch in the user's selection criterion is registered much quicker for the estimated best individual of RIE than that of CIE. As evolution pushes towards the estimated best individual according to the preference model, the more accurate RIE model results in the faster optimization of the population's best individual. RIE is faster to optimize both its estimated and its real best individual than CIE for all artificial users, except in cases where optimization is equally effortless for all tested approaches (e.g. f_{BS} and f_{RB}). The models' efficiency varies depending on the fitness function and how easy it is to optimize: Table 10.3 demonstrates how, in cases where UB struggles to discover fit individuals (such as A_{US} and A_{CP}), RIE and (moreso) CIE fail to match the performance of UB. Finally, while composite functions are slow to optimize for UB, RIE manages to perform well; both CIE and RIE, however, put less emphasis on the fitness dimensions with the largest weight than UB, and their best individuals are often less dominated by it (especially for f_{BD} in A_{C1}). The large standard error in A_{C1} and A_{C2} for CIE and RIE approaches points to an unpredictable behavior when conflicting fitnesses are simultaneously optimized, as their weights are often readjusted in each iteration to give a single fitness prominence over others.

10.2.4 Discussion

The highly controllable nature of artificial users allowed the experiments of Section 10.2.3 to demonstrate the ability of RIE and CIE to capture an artificial user's preference in a few iterations. Indeed, the preference model is very quickly adjusted when an artificial user selects content according to one of the fitnesses included in the model and content is evolved to satisfy the artificial user's taste. As with experiments of Chapter 9, there is a strong assumption that the fitness functions included in the adaptive model are indicative of human criteria of assessing strategy game maps. In order to validate the full potential of RIE, an experiment with human users is necessary to test whether the fitness functions included in the user model match human taste. Moreover, experiments with human users can assess any user fatigue caused by the cognitive load of relative comparisons when ranking multiple maps.

Since CIE and RIE models are contingent on engineered fitnesses, they are sensitive to bad fitness definitions. This was obvious in the experiments of 10.2.3, since different fitness dimensions often have very different score values for "typical" maps, such as f_{BS} which is optimal for most maps (even randomly generated ones) and f_{CP} which requires very specific map patterns to reach even average scores. In order for the adaptive models to more accurately detect all fitness dimensions as well as to avoid optimization being dominated by one dimension in the weighted sum, a more fair distribution of fitness scores should be in place. The fitness design of Sentient Sketchbook's quality evaluations (Section 5.2.1 was largely inspired by these findings; the different fitness dimensions of Sentient Sketchbook are not conflicting and are not as disparate in terms of optimization behavior.

The representation for strategy game maps was straightforward both to describe and to implement, and can create patterns that are immediately appraised by a user. Small changes in most chromosomes correspond to small changes in a single map feature's placement, which helps preserve map structure through the genetic operators used. However, the *type* parameter of impassable regions was considered to be problematic, as the change from line to rectangular

area is abrupt at a specific value (0.5) and lines and rectangles are represented differently in the genotype. The map sketches' direct mapping between genotype and phenotype in Sentient Sketchbook of Section 5.5.1) was motivated by this finding and limits the destructive potential that a single parameter change can cause in the strategy game maps of this experiment.

10.3 Computers as Creators and Critics: DeLeNoX

Exploring the limits of computational initiative, is it possible to have a mixed-initiative tool where both initiatives are provided by the computer? Would a computational creator be able to benefit from a computational critic that evaluates the generated artifacts without accounting for the method and rationale behind the creative process? Can the search for promising solutions change based on the feedback of a computational critic? Experiments in Chapter 9 and Section 10.2 actually tested this, although the agent-critics selecting content were mere "stand-ins" simulating human users. However, deep learning techniques used for computer vision can be used as computational critics which do not necessarily evaluate content similarly to human users and are not imitators of human taste but autonomously creative entities.

The last experiment described in this thesis proposes a method for exploring the space of possible artifacts based on a novelty heuristic which changes according to a critic observing the artifacts generated previously by a computational creator. Novelty search reflects on its creative path by adjusting its characterization of diversity according to the artifacts it currently generates. As this characterization drives the exploration of the search space, adapting it to break current patterns of its artifacts will lead to the exploration of the search space in a new dimension, orthogonal to the one currently being explored. This will not only result in a more thorough exploration of the search space but also simulates, to a degree, the way humans are creative via lateral thinking [55] and reinterpreting creative problems [88].

The Deep Learning Novelty Explorer (DeLeNoX) system attempts to enhance the exploration afforded by novelty search by iteratively adjusting the characterization of distance. DeLeNoX combines phases of exploration through constrained novelty search (the creator initiative) with phases of transformation through deep learning autoencoders (the critic initiative). Fig. 10.11 visualizes this closed creative loop. In the exploration phases, DeLeNoX finds the most diverse set of artifacts possible given a particular distance function. In the transformation phases, it characterizes the found artifacts by obtaining a low-dimensional representation of their differences. This is done via autoencoders, a novel technique for nonlinear principal component analysis [11]. The features found by the autoencoder are orthogonal to the bias of the current CPPN complexity, ensuring that each exploratory phase has a different bias than the previous.

The case study of DeLeNoX is the generation of two-dimensional spaceships which can be used in space shooter games such as *Galaga* (Namco 1981). Automatically generating visually diverse spaceships which however fulfill constraints on believability addresses the "content creation" bottleneck of many game titles. The spaceships are represented as images generated by Compositional Pattern-Producing Networks (CPPNs) [230] with constraints on which shapes are viable spaceships. Exploration is done through constrained novelty search, which tries to increase diversity in a population according to a measure of difference between individuals. The distance measure inherently privileges some region of the search space over others, in particular when searching at the border of feasible search space. Additionally, CPPNs with

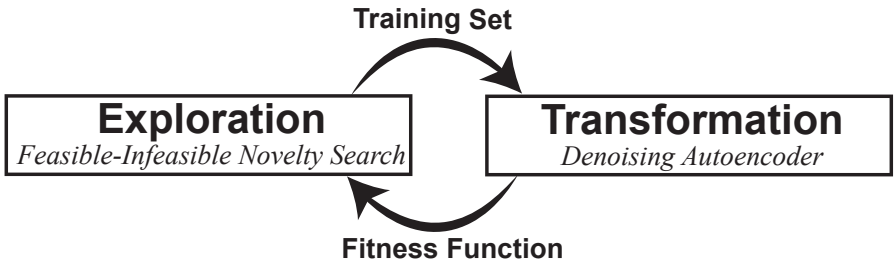


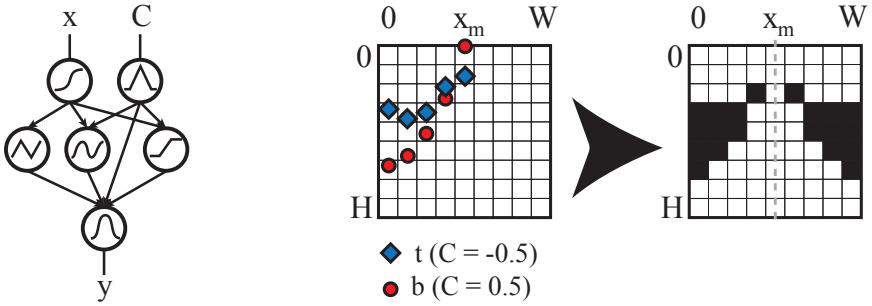
Fig. 10.11: Exploration transformed with DeLeNoX: the flowchart includes the general principles of DeLeNoX (bold) and the methods of the presented case study (italics).

different topologies are likely to create specific patterns in generated spaceships, with more complex CPPNs typically creating more complex patterns. Therefore, in different stages of this evolutionary complexification process, different regions of the search space will be under-explored. Many artifacts that are expressible within the representation will thus most likely not be found; in other words, there are limitations to creativity because of search biases.

In order to alleviate this problem and achieve a fuller coverage of space, the biases are algorithmically *characterized* from the search process and the representation. This is what the autoencoders do. These autoencoders are applied on a set of spaceships resulting from an initial exploration of the space. As described in Section 3.1.2, the encoder part of a trained autoencoder is a function from a complete spaceship (phenotype) to a relatively low-dimensional array of real values. The output of this function is then used to compute a new distance measure, which differs from previous ones in that it better captures typical patterns at the current representational power of the spaceship-generating CPPNs. Changing the distance function amounts to changing the exploration process of novelty search, as novelty search is now in effect searching along different dimensions (see Fig. 10.11). This accomplishes *transformed* exploratory creativity, not by changing or abandoning any constraints, but by adjusting the search bias. This can be seen as analogous to changing the painting technique of a painter, the analysis sequence of an inventor, or introducing new plot devices for a writer. All of the spaceships that are found by the new search process could in principle have been found by the previous processes, but were very unlikely to be.

10.3.1 Domain Representation

The proof-of-concept of DeLeNoX focuses on the the creation of spaceship sprites, where exploration is performed via constrained novelty search which ensures a believable appearance, while transformation is performed via a denoising autoencoder which finds typical features in the spaceships' current representation (see Fig. 10.11). Spaceships are stored as two-dimensional sprites; the spaceship's hull is shown as black pixels. Each spaceship is encoded by a Compositional Pattern-Producing Network (CPPN), which is able to create complex patterns via function composition [230]. A CPPN is ideal for visual representation as it can be queried with arbitrary spatial granularity (infinite resolution); however, this study uses a fixed resolution for simplicity. Unlike standard artificial neural networks where all nodes have the same activation function, each CPPN node may have a different, pattern-producing function; six activation functions bound within $[0, 1]$ are used in this study (see Fig. 10.12a). To generate



(a) CPPN. (b) Spaceship representation.

Fig. 10.12: Fig 10.12a shows a sample CPPN using the full range of pattern-producing activation functions available. Fig. 10.12b shows the process of spaceship generation: the coordinates 0 to x_m, normalized as 0 to 1 (respectively) are used as input x of the CPPN. Two C values are used for each x, resulting in two points, top (t) and bottom (b) for each x. CPPN input x and output y are treated as the coordinates of t and b; if t has a higher y value than that of b then the column is empty, else the hull extends between t and b. The generated hull is reflected vertically along x_m.

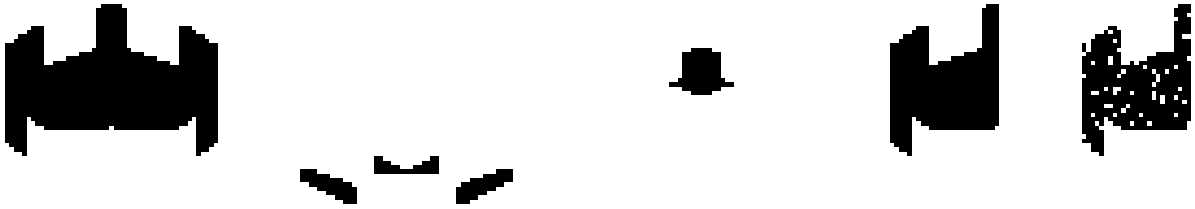
a spaceship, the sprite is divided into a number of equidistant columns equal to the sprite's width (W) in pixels. In each column, two points are identified as top (t) and bottom (b); the spaceship extends from t to b, while no hull exists if t is below b (see Fig. 10.12b). The y coordinate of the top and bottom points is the output of the CPPN; its inputs are the point's x coordinate and a constant C which differentiates between t and b (with $C = -0.5$ and $C = 0.5$, respectively). Only half of the sprites' columns, including the middle column at $x_m = \lceil \frac{W}{2} \rceil$, are used to generate t and b; the remaining columns are derived by reflecting vertically along x_m.

A sufficiently expanded CPPN, as a superset of a multi-layer neural network, is theoretically capable of representing any function. This means that any image could in principle be produced by a CPPN. However, the interpretation of CPPN output used here means that images are severely limited to those where each column contains at most one vertical black bar. Additionally, the particularities of the NEAT complexification process, of the activation functions used and of the distance function which drives evolution make the system heavily biased towards particular shapes. It is this latter bias that is characterized within the transformation phase.

10.3.2 Transformation Phase: Denoising Autoencoder

The core innovation of DeLeNoX is the integration of autoencoders described in Section 3.1.2 in the calculation of the novelty heuristic (described in the next section), which is used to explore the search space according to the current representational power of the encoding CPPNs.

For the purposes of detecting the core visual features of the generated spaceships, DeLeNoX uses denoising autoencoders to transform the spaceship's sprite to a low-dimensional array of real values, which correspond to the output of the encoder. Since spaceships are symmetrical along x_m, the training set consists of the left half of every spaceship sprite (see Fig. 10.13d). The encoder has $H \cdot \lceil \frac{W}{2} \rceil$ inputs (P in Fig. 3.1), which are assigned a corrupted version of the



(a) (b) (c) (d) (e)

Fig. 10.13: Sample spaceships of 49 by 49 pixels, used for demonstrating DeLeNoX. Fig. 10.13a is a feasible spaceship; Fig. 10.13b and 10.13c are infeasible, as they have disconnected pixels and insufficient size respectively. The autoencoder is trained to predict the left half of the spaceship in Fig. 10.13a (Fig. 10.13d) from a corrupted version of it (Fig. 10.13e).

spaceship's half-sprite; corruption is accomplished by randomly replacing pixels with 0, which is the same as randomly removing pixels from the spaceship (see Fig. 10.13e). The encoder has N neurons which correspond to the number of high-level features captured; each feature q_i is a function of the input P as $sig(W_i \cdot P + b_i)$ where $sig(x)$ the sigmoid function and $\{W_i, b_i\}$ the feature's learnable parameters (weight set and bias value, respectively). The output P' of the decoder is an estimation of the uncorrupted half-sprite derived from $Q = [q_1, q_2, \ldots, q_N]$ via $P' = sig(W' \cdot Q + B')$; the denoising autoencoder uses tied weights and thus W' is the transpose of $W = [W_1, W_2, \ldots, W_N]$. The parameters $\{W, B, B'\}$ are trained via backpropagation [200] according to the mean squared error between pixels in the uncorrupted half-sprite with those in the reconstructed sprite.

10.3.3 Exploration Phase: Constrained Novelty Search

The spaceships generated by DeLeNoX are expected to be useful for a computer game; spaceships must have a believable appearance and sufficient size to be visible. Specifically, spaceships must not have disconnected pixels and must occupy at least half of the sprite's height and width; see examples of infeasible spaceships in Fig. 10.13b and 10.13c. In order to optimize feasible spaceships towards novelty, content is evolved via feasible-infeasible novelty search (FINS) as described in Section 4.2. Based on the constraints of this problem, the infeasible population selects parents based on their proximity to the feasible border (f_{inf}), defined as:

$$f_{inf} = 1 - \frac{1}{3}\left[\max\{0, 1 - \tfrac{2w}{W}\} + \max\{0, 1 - \tfrac{2h}{H}\} + \tfrac{A_s}{A}\right]$$

where w and h is the width and height of the spaceship in pixels; W and H is the width and height of the sprite in pixels; A is the total number of black pixels on the image and A_s the number of pixels on all disconnected segments.

For the feasible population, the paradigm of novelty search is followed in order to explore the full spectrum of the CPPNs' representational power. According to novelty search, the novelty score $\rho(i)$ for a feasible individual i amounts to its average difference with the k closest feasible neighbors within the population or in an archive of past novel individuals [132]. In DeLeNoX, the difference used to calculate ρ is the Euclidean distance between the high-level features

discovered by the denoising autoencoder; thus $\rho(i)$ is calculated as:

$$\rho(i) = \frac{1}{k} \sum_{m=1}^{k} \sqrt{\sum_{n=1}^{N} [q_n(i) - q_n(\mu_m)]^2}$$

where μ_m is the m-th-nearest neighbor of i (in the population or the archive of novel individuals); N is the number of hidden nodes (features) of the autoencoder and $q_n(i)$ the value of feature n for spaceship i. As with the training process of the denoising autoencoder, the left half of spaceship i is used as input to $q_n(i)$, although the input is not corrupted.

10.3.4 Experimentation

DeLeNoX will be demonstrated with the iteratively transformed exploration of spaceships on sprites of 49 by 49 pixels. The experiment consists of a series of *iterations*, with each iteration divided into an exploration phase and a transformation phase. The exploration phase uses constrained novelty search to optimize a set of diverse spaceships, with "diversity" evaluated according to the features of the previous iteration; the transformation phase uses the set of spaceships optimized in the exploration phase to create new features which are better able to exploit the regularities of the current spaceship complexity. Each exploration phase creates a set of 1000 spaceships, which are generated from 100 independent runs of the FINS algorithm for 50 generations; the 10 fittest feasible individuals of each run are inserted into the set. Given the genetic operators used in the mutation scheme, each exploration phase augments the CPPN topology by roughly 5 nodes. While the first iteration starts with an initial population consisting of CPPNs with no hidden nodes, subsequent iterations start with an initial population of CPPNs of the same complexity as the final individuals of the previous iteration. The total population of each run is 200 individuals, and parameters of novelty search are $k = 20$ and $l = 5$. Each evolutionary run maintains its own archive of novel individuals; no information regarding novelty is shared from previous iterations or across runs. Forgetting past visited areas of the search space is likely to hinder novelty search, but using a large archive of past individuals comes with a huge computational burden; given that CPPN topology augments in each iteration, it is less likely that previous novel individuals will be re-discovered, which makes "forgetting" past breakthroughs an acceptable sacrifice.

Each transformation phase trains a denoising autoencoder with a hidden layer of 64 nodes, thus creating 64 high-level features. The weights and biases for these features are trained in the 1000 spaceships created in the exploration phase. Training runs for 1000 epochs, trying to accurately predict the real half-sprite of the spaceship (see Fig. 10.13d) from a corrupted version of it (see Fig. 10.13e); corruption occurs by replacing any pixel with a white pixel (with 10% chance).

The progress of DeLeNoX is observed for 6 iterations. For the first iteration, the features driving the exploration phase are trained on a set of 1000 spaceships created by randomly initialized CPPNs with no hidden nodes; these spaceships and features are identified as "initial". The impact of transformation is shown via a second experiment, where spaceships evolve for 6 iterations using the initial set of features trained from simple spaceships with no transformation phases between iterations; this second experiment is named "static" (contrary to the proposed "transforming" method).

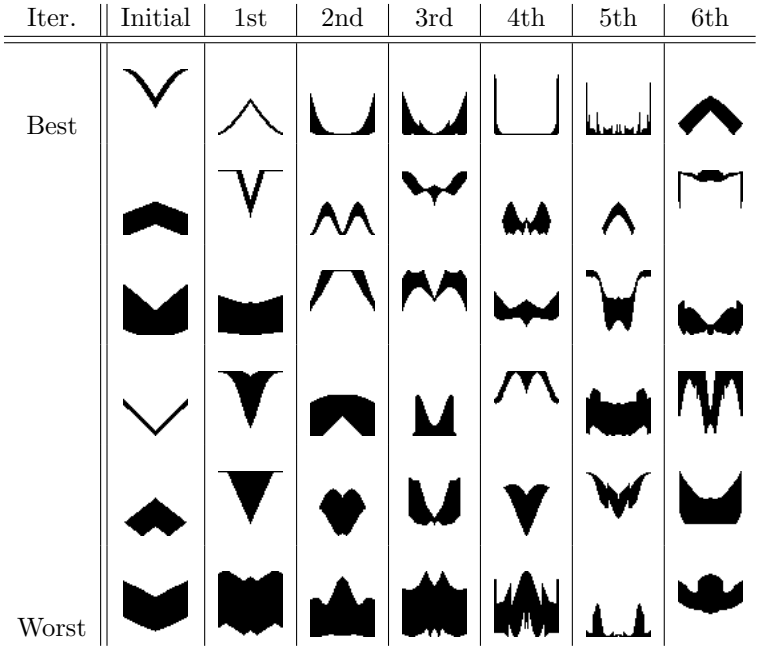


Fig. 10.14: Sample spaceships among the results of each iteration of exploration; such spaceships comprise the training set for detecting the next iteration's features (transforming run). The best and worst spaceship in terms of difference (using the previous iteration's features) is included, along with spaceships evenly distributed in terms of difference.

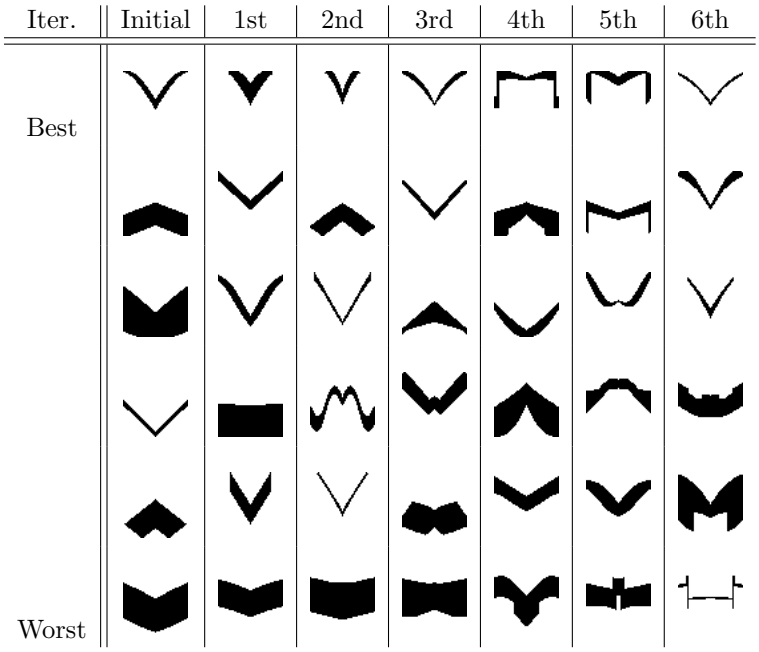


Fig. 10.15: Sample spaceships (sorted by difference) among the results of each iteration of exploration driven by static features trained on the initial spaceship set (static run).

The final spaceships generated in the exploration phase of each iteration are shown in Fig. 10.14 for the transforming run and in Fig. 10.15 for the static run. These figures show six samples selected based on their diversity (according to the features on which they were evolved); Fig. 10.14 and 10.15 therefore not only showcase the artifacts generated by DeLeNoX, but the sampling method demonstrates the shapes which are identified as "different" by the features. A larger sample of the spaceships generated by the two methods is shown in Appendix B.

In Fig. 10.14, the shifting representational power of CPPNs is obvious: CPPNs with no hidden nodes tend to create predominantly V-shaped spaceships, while larger networks create more curved shapes (such as in the 2nd iteration) and eventually lead to jagged edges or "spikes" in later iterations. While CPPNs can create more elaborate shapes with larger topologies, Fig. 10.14 includes simple shapes even in late iterations: such an example is the 6th iteration, where two of the sample spaceships seem simple. This is likely due to the lack of a "long-term memory", since there is no persistent archive of novel individuals across iterations.

In terms of detected features, Fig. 10.17 displays all of the 64 features trained in each transformation phase of the transforming run; the static run uses the "initial" features (see Fig. 10.17a) in every iteration. The shape of the spaceships directly affects the features' appearance: for instance, the simple V-shaped spaceships of the initial training set result in features which detect diagonal edges. The features become increasingly more complex, and thus difficult to identify, in later iterations: while in the 1st iteration straight edges are still prevalent, features in the 5th or 6th iterations detect circular or vertical areas.

Comparing Fig. 10.15 with Fig. 10.14, it is clear that despite the larger CPPN topologies of later iterations, spaceships evolved in the static run are much simpler than their respective ones in the transforming run. Exploration in the static run is always driven by simple initial features (see Fig. 10.17a), showing how the features used in the fitness function ρ bias search. On the contrary, the transformation phase in each iteration counters this bias and re-aligns exploration towards more visually diverse artifacts.

The diversity of spaceships and the quality of detected features can be gleaned from Fig. 10.16, in which features trained in different iterations of the transforming run generate distance metrics which evaluate the diversity of every iteration's training set, both for the transforming and for the static run. Diversity is measured as the Euclidean distance averaged from all spaceship pairs of the training set of an iteration. In the transforming run, the highest diversity score for a feature set is usually attained in the training set of the following iteration (e.g. the initial features score the highest diversity in the 1st iteration's spaceships). This is expected, since the features of the previous iteration are used in the distance function driving novelty search in the next iteration. This trend, however, does not hold in the last 3 iterations, possibly because patterns after the 3rd iteration become too complex for 64 features to capture, while the simpler patterns of earlier iterations are more in tune with what they can detect. It is surprising that features of later iterations, primarily those of the 3rd and 6th iteration, result in high diversity values in most training sets, even those of the static run which were driven by the much simpler initial features. It appears that features trained in the more complicated shapes of later iterations are more general — as they can detect patterns they haven't actually seen, such as those in the static run — than features of the initial or 1st iteration which primarily detect straight edges (see Fig. 10.17).

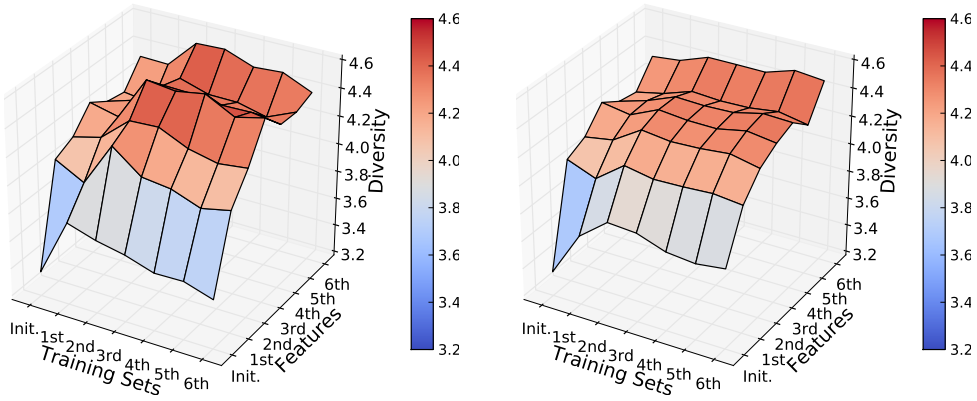


Fig. 10.16: Diversity scores of the training sets at the end of each iteration's exploration phase, derived from the feature sets trained in the transformation phases of the transforming run. The training sets of the transforming run are evaluated on the left figure, and those of the static run on the right.

10.3.5 Discussion

DeLeNoX was implemented in a proof-of-concept generative system which transforms exploration of the search space in order to counter the biases of the representation and the evolutionary process. While short, the included case study demonstrates the potential of DeLeNoX in several distinct but complementary ways. The shifting representation of augmenting CPPNs benefits from the iterative transformations of the novelty heuristic which is used to evolve it, as demonstrated by early features which detect straight lines versus later features which focus on areas of interest. Using the early, simple features for evolving complex CPPNs is shown to hinder exploration since the representational bias which caused those features to be prevalent has been countered by augmenting topologies. On the other hand, the iterative exploration guided by features tailored to the representation creates a more diverse training set for the autoencoder, resulting in an overall improvement in the features detected as shown by the increased diversity scores of later features on the same data. This positive feedback loop, where the exploration phase benefits from the transformation phase, which in turn benefits from the improved divergent search of exploration is the core argument for DeLeNoX. It should be noted, however, that for this case study DeLeNoX is not without its own biases, as the increasingly diverse training set eventually challenges the feature detector's ability to capture typical patterns in the latest of presented iterations; suggestions for countering such biases will be presented below.

The current proof-of-concept is an example of exploration via high-level features derived by compressing information based on their statistical dependencies. The number of features chosen was arguably arbitrary; it allows for a decent compression (980 pixels to 64 real values) and measuring the Euclidean distance for novelty search is computationally manageable. At the same time, it is large enough to capture the most prevalent features among generated spaceships, at least in the first iterations where spaceships and their encoding CPPNs are simple. As exploration becomes more thorough — enhanced both by the increased representational power of larger CPPNs and by more informed feature detectors — typical patterns become harder to find. It could be argued that as exploration results in increasingly more

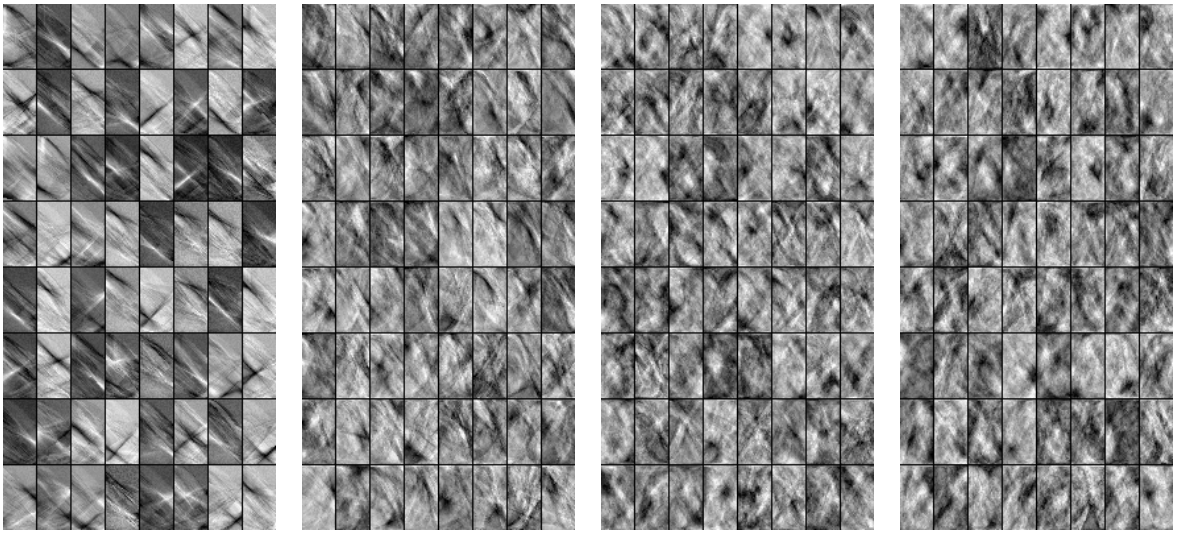
diverse content, the number of features should increase to counter the fewer dependencies in the training set; for the same reasons, the size of the training set should perhaps increase. Future experiments should evaluate the impact of the number of features and the size of the training set both on the accuracy of the autoencoder and on the progress of novelty search. Other experiments should explore the potential of adjusting these values dynamically on a per-iteration basis; adjustments can be made via a constant multiplier or according to the quality of generated artifacts.

It should be pointed out that the presented case study uses a single autoencoder, which is able to discover simple features such as edges. These simple features are easy to present visually, and deriving the distance metric is straightforward based on the outputs of the autoencoder's hidden layer. For a simple testbed such as spaceship generation, features discovered by the single autoencoder suffice — especially in early iterations of novelty search. However, the true potential of DeLeNoX will be shown via stacked autoencoders which allow for truly *deep* learning; the outputs from the upper layers of such a deep belief network [11] represent more "abstract" concepts than those of a single autoencoder. Using such robust features for deriving a novelty value is likely to address current limitations of the feature extractor in images generated by complex CPPNs, and can be applied to more complex problems.

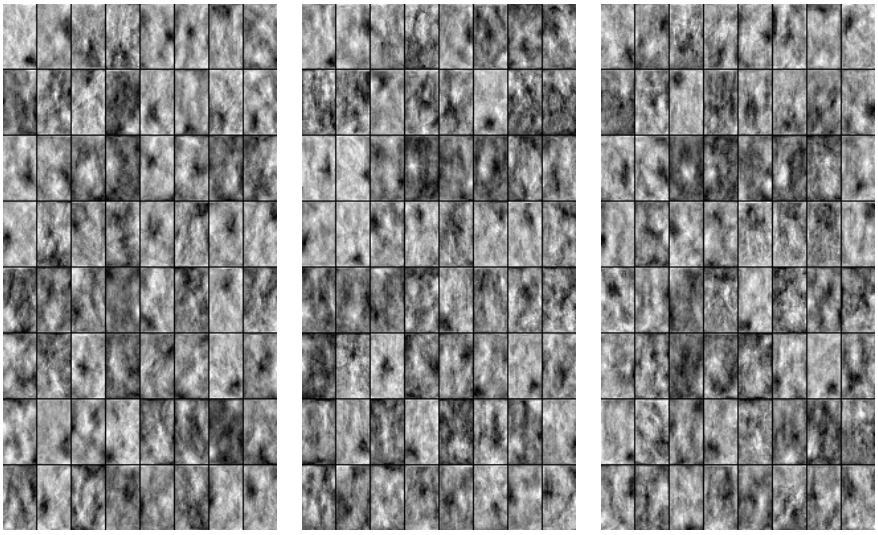
The proof-of-concept described above is a good demonstration of DeLeNoX due to the evolutionary complexification of CPPNs; the indirect mapping between genotype and phenotype and the augmenting topologies both warrant the iterative transformation of the features which drive novelty search. A direct or static mapping would likely find the iterative transformation of the search process less useful, since representational bias remains constant. However, any indirect mapping between genotype and phenotype including neuroevolution, grammatical evolution or genetic programming can be used for DeLeNoX.

10.4 Summary

This Chapter covered three different alternatives to the sketching paradigm of Sentient Sketchbook, exploring the degree that computational initiative can be used during the game design process. While the artifacts generated by all three prototypes are primarily visual, each type of artifact has its own set of affordances and constraints: terrain generated by Sentient World does not have any inherent constraints on playability as it is intended more for world building in a tabletop role-playing game than for gameplay in a digital game, interactively evolved strategy game levels have rather specific desirable objectives both for gameplay but also for visual impression, while spaceships created by DeLeNoX have strict constraints on what can be visible and identifiable as a spaceship. The different prototypes afford different types of mixed-initiative design, although DeLeNoX stretches the definition unless one considers "computer-computer initiative" where some algorithms act as creators and others as critics. While the evaluation of these prototypes is not as expansive as that of Sentient Sketchbook, several conclusions were drawn and directions for future work were proposed in the prototypes' individual sections. More general conclusions, limitations and extensions of the line of research followed in this thesis are laid out in the next Chapter.



(a) Initial (b) 1st (c) 2nd (d) 3rd



(e) 4th (f) 5th (g) 6th

Fig. 10.17: The 64 trained features at the end of each iteration. The visualization displays the weights of each pixel of the input (i.e. the left half of the spaceship's sprite). Weights are normalized to black (lowest) and white (highest).

Chapter 11

Discussion

This thesis set out to explore methods for aiding game design via artificial intelligence techniques. Towards that end, Sentient Sketchbook has been implemented to address the task of level design. While initial tests focused on the design of strategy game levels (motivated by earlier work on strategy game maps documented in Section 10.2), the abstraction of map sketches has allowed for the design of levels for multiple game genres as shown in Section 7.2. The task of level design was augmented both by the interface of Sentient Sketchbook, which uses map sketches rather than complete levels and has a straightforward but informative interface, but also by algorithmical innovations mostly focusing on generating suggestions that the designer may find interesting, and on modeling the designer's preferences, goals and process to increase the designers' interest in those suggestions. The former has been accomplished via evolutionary computation: a number of existing methods such as the feasible-infeasible two-population genetic algorithm and novelty search have been extended and applied to the domain of map sketch evolution. The task of designer modeling has primarily been accomplished through adaptive models which derive preference from the user's choice of one or more artifacts from a set. The various algorithms have been also tested in other mixed-initiative game design tools in Chapter 10, where the level of user feedback varied: Sentient World required a human to initiate a first concept, rank-based interactive evolution of strategy game levels used human feedback to indirectly guide the generative algorithms, while DeLeNoX required no feedback as it substituted human judgment with computational critics.

This thesis has thus explored how and to what degree computational initiative can interweave with human initiative in the generation of artifacts. During this exploration, however, a number of limitations — mostly pertaining to scalability — were discovered in the genetic algorithms used. Additionally, there is an upper bound to the generality of the conclusions drawn by experiments in this thesis, especially when it comes to how people co-create with machines for tasks outside level design or via methods other than sketching. Section 11.1 will discuss the limitations of the algorithms introduced but also of the experiments reported in this thesis, while Section 11.2 will describe the next steps which can address these limitations and expand the scope and generality of this thesis' findings.

11.1 Limitations

In order to create a tool which could use an artificial designer as a co-creator in real-time, a number of design decisions — such as using a domain focused on visual identification (level design) and working on abstract sketches rather than final maps — were taken and a number of algorithms were implemented. Each of the components, from optimization to novelty search to

actual design process, were tested individually in their respective Chapters within this thesis. Unsurprisingly, each component works well for the task at hand (i.e. map sketching in Sentient Sketchbook) but comes with several limitations when applied to larger — or different — tasks or to tasks other than map sketching. This section discusses the limitations found in each of these components; Section 11.2 will provide directions for future work in each component which can address the limitations discussed here.

11.1.1 Limitations of FI-2pop GA

Experiments with feasible-infeasible two-population genetic algorithms, reported in Chapter 7, showed that FI-2pop GAs are appropriate for the task of constrained optimization of map sketches. The largest strength of the algorithm is that it can discover feasible individuals quickly, even in highly constrained spaces and for different types of levels. The quick discovery of feasible solutions speeds up the optimization of playable map sketches, allowing for a quick improvement in the fitness function being optimized. While this initial "burst" of relatively good playable content is particularly important for the real-time requirements of Sentient Sketchbook (and any other mixed-initiative design tool), the FI-2pop GA tends to quickly settle on an optimum (likely local) and converge the entire feasible population towards it. In many experiments of Chapter 7 the maximum fitness of the feasible population does not improve dramatically after the first 20 generations, while the average fitness converges to that of the best individual — usually by directly copying the best map's patterns. This behavior indicates that the FI-2pop GA as applied to map sketches may suffer from premature convergence, which is a limitation that should be addressed. The use of recombination on a genotype which directly represents the phenotype may be a reason for the apparent convergence. Mutation showed that a higher maximum fitness can be reached and at the same time the average population does not converge to the best individual as obviously as with recombination. However, mutation often leads to fewer feasible individuals and slows down optimization in highly constrained spaces; for the lightweight genetic algorithms needed for mixed-initiative game design tools (in terms of population size and number of generations) this behavior may not be appropriate. Less destructive mutation operators could be devised, however, by tweaking the likelihood of impassable tiles being inserted into the map sketch, as this directly affects the likelihood of unreachable special tiles.

Another important limitation of the experiments reported in this thesis is the behavior of FI-2pop GA when multiple objectives are being targeted. The aggregation of multiple fitness dimensions is admittedly a naive solution to the complex problem of multiobjective optimization. However, the fitness functions of map sketches as described in Section 5.2.1 are not particularly conflicting: in the case of f_{exp} and f_{saf} for strategy maps, optimizing one dimension is very likely to improve the score in the other dimension since all of f_{exp}, f_{saf} and f_{res} benefit from bases being far away from each other. While b_{res}, b_{saf} and b_{exp}, when optimized individually, result in maps with nearby bases and thus low scores in f_{res}, f_{saf} and f_{exp}, experiments in Section 7.1.2 showed that a balance and a fitness dimension can be simultaneously optimized with no obvious impact in GA performance and thus there is no tradeoff between e.g. b_{res} and f_{res}. Since the design of the evaluations of level quality in Section 5.4.1 are parametrizable and can be reconfigured for any type of map sketch, it is possible that for a specific level setup there will be conflicting objectives and the aggregation of all fitness

dimensions into a weighted sum will cause the FI-2pop GA to underperform. This suboptimal behavior does not affect the current version of Sentient Sketchbook, since each shown suggestion is optimized to a single fitness dimension in its individual thread, but may become an issue when the adaptive models of preference or process are integrated. Since preference modeling as envisioned by choice-based and rank-based interactive evolution uses a weighted sum (with adapted weights) as the single fitness to be optimized, there is no straightforward solution for handling a potential problem with conflicting objectives other than carefully designing the fitness dimensions so that they have similar value ranges and can reach them with relatively similar ease. The current set of evaluations of Chapter 5 largely accomplish that, although f_{res} and to a lesser degree f_{saf} reach lower values than the other dimensions — especially in small strategy game map sketches.

11.1.2 Limitations of Constrained Novelty Search

Experiments with constrained novelty search in Chapter 8 showed that the choice of genetic operators, the shape of the constrained spaces, and the goals of the creative process all affect which method is ideal for the task at hand. More specifically, novelty search via mutation alone was shown to lead to more diverse feasible individuals and a faster discovery of feasible solutions in highly constrained spaces. Mutation, by its very nature, may be more efficient at exploration than the convex search of recombination operators, but the high diversity scores it attains for most novelty search methods can easily be traced to the direct genotype to phenotype mapping and to the distance metric (d_{vis}), which directly compares phenotypical appearance. Due to these factors, novelty search essentially rewards individuals with different genotypes. Therefore, even if mutation is destructive (e.g. rendering many feasible individuals infeasible or making substantial changes to the genotype), it would lead to higher diversity scores than a less destructive crossover operator. The current mutation operator seems to be both harmful, as it results in fewer feasible individuals than recombination, and beneficial, as it discovers feasible individuals faster and more reliably than recombination. Before any conclusions can be drawn regarding the reliability of mutation for novelty search, further experimentation should test its behavior with different mappings from genotype to phenotype and, more importantly, with different distance metrics which do not rely only on phenotypical appearance. Some early investigations of novelty search of map sketches targeting visual impressiveness or quality-based diversity have been reported in [190].

Experiments presented in Chapter 8 demonstrate broadly how the different methods of novelty search, both constrained and unconstrained, perform in different types of constrained spaces. However, there are several directions for enhancing this first systematic study of constrained exploration. The previous paragraph already noted that more experiments should test how mutation and recombination perform when search targets diversity in a dimension less tied to genotypical form. Depending on the distance metric used, mutation could prove less efficient or demonstrate larger deviations in the behavior of different search methods. A distance metric could also include a notion of feasibility, e.g. by assigning a high distance score when comparing feasible content with infeasible content. Such a distance metric would likely impact unconstrained novelty search, which mostly suffered in the current experiments from a distance metric lacking any indication of the value of feasible individuals. Apart from different distance metrics or genotype to phenotype mappings, other performance metrics could also be

introduced for measuring the exploration of a search space. Measuring the diversity of past discoveries, such as the novel archive or populations of previous generations, could evaluate a "temporal" aspect of exploration; a high final feasible diversity, for instance, is less valuable if the entire evolutionary run was spent exploring the same area of feasible space. Including such past discoveries in a measure of exploration could warrant experiments testing the impact of the size of the novel archive and the number of closest individuals used for evaluating the novelty score of eq. 5.10. Preliminary tests with different numbers of novel individuals stored in the archive per generation, or with different values of k in eq. 5.10 did not seem to impact the average feasible diversity in any meaningful way, and were omitted from Chapter 8.

11.1.3 Limitations of Adaptive Models

Experiments presented in Chapter 9 showed that choice-based interactive evolution can be applied to Sentient Sketchbook in order to learn a model of user style and apply it in the generation of more appropriate suggestions. The performance of choice-based interactive evolution was tested with artificial users, which selected more (in Section 9.1.1) or less (in Section 9.1.3) according to the evaluations of quality already integrated in Sentient Sketchbook. The obvious limitation in this set of experiments is the lack of testing with actual human users, where the heuristics of human designers may vary from — or even be orthogonal to — the evaluation formulas in the adaptive model. The CIE_{zero} method was tested as an alternative of CIE which does not make assumptions about the human users' initial goals; that does not mean that it is free of assumptions, however, as it still adapts the weights of the evaluations of Sentient Sketchbook. Testing (either with artificial or with human users) what artifacts are produced by CIE when the target is unrelated to strategic quality (e.g. visual symmetry or grouping of impassable tiles) would be interesting, although it is unclear what the results would demonstrate and how they could be systematically evaluated.

Beyond Sentient Sketchbook, the experiments with larger strategy game maps in Section 10.2 demonstrated the ability of RIE and CIE in capturing user preference in a few iterations; the preference model is very quickly adjusted when an artificial user selects content according to one of the fitnesses included in the model and content is altered to satisfy the artificial user's taste. As with Sentient Sketchbook, however, the system cannot detect preference criteria outside those evaluated by the provided fitnesses; at best, it can detect criteria created by the linear combination of those fitnesses, such as a map with symmetrical impassable areas on one axis but asymmetrical on the other. The evaluations which were implemented for larger strategy game maps include certain qualities of aesthetic appearance (which are often found in users' drawn maps as evidenced by Fig. 6.1), although aesthetic appearance was found to be easy to optimize — for the fitnesses used — even for standard interactive evolution. More so than in experiments with Sentient Sketchbook, the experiments of Section 10.2 demonstrate the deleterious effect of poorly designed fitnesses aggregated into a weighted sum, as was the case of f_{CP} (chokepoint evaluation) which had very low scores for most maps versus f_{BS} (base space evaluation) which had optimal or near-optimal scores for most maps. These limitations discovered in the experiments of Section 10.2 were the primary motivation for designing Sentient Sketchbook with smaller map sizes and using only six carefully designed fitness functions.

11.1.4 Limitations of Sentient Sketchbook

Sentient Sketchbook has been the most successful of the different mixed-initiative design prototypes in combining genetic algorithms and human design processes. It has also been shown that Sentient Sketchbook can accomplish mixed-initiative co-creativity [269] by generating suggestions which are considered creative milestones, as shown in Chapter 6. However, the current version of Sentient Sketchbook comes with several limitations mostly pertaining to its scope.

The formulas for evaluating level quality presented in Chapter 5 could be further enhanced. Additional metrics based on the affordances and constraints of other game genres (such as shooters and platformers) could be formulated: for instance, a shooter may require the line of sight coverage of each tile [248, 213] while platformers may need a notion of jump difficulty [9, 228] (e.g. a constraint on connectedness based on jump difficulty). The existing formulas could also be refined. Specifically, the safety metric is problematic when more than two reference tiles are included in the set; in such cases, unsafe tiles can be equally far from two of these elements, while remaining elements may be much farther way. Additionally, both exploration and safety metrics should place more importance on chokepoints, since crossing a chokepoint near a base or monster should be more difficult than crossing an open area. Finally, the current formulas evaluate the initial topology of the level but do not take into account the game's progression. Further enhancements to these evaluations could include simulations of actual gameplay; unlike the current metrics, however, such simulations would likely be particularly domain-dependent.

While the suggestions of Sentient Sketchbook are designed so that they can be generated in real-time, there is an upper bound to the size of the map sketch which can be evolved within what is considered a reasonable response time according to human users. In experiments with large strategy game maps, the generation of suggestions in 7 separate threads would require so much computational effort that users often had made further changes or additions to the map sketch by the time the suggestions were presented. In cases where the user made drastic changes to the sketch during that time (e.g. by adding a large area of impassable tiles), the suggestions would have visual characteristics of the user's map sketch of several steps back. As has been argued in the offline experiments of Chapter 7, large maps likely constitute the upper bound of what the fitness functions can evaluate (due to averaging effects from too many special tiles) as well as the generative algorithms due the extraneous time required for evolution, evaluation and discovery of feasible individuals.

11.1.5 Limitations of the Domain

Sentient Sketchbook allows users to design abstract game levels which can be translated in a straightforward manner into many different game genres. However, it is still restricted to level design rather than the larger problem of designing a complete game (including its game mechanics, storyline or sound effects). Other experiments with varying human and computational initiative documented in Chapter 10 also focus mostly on the visual impression of the generated artifacts: generated terrain in Sentient World is drawn on a top-down map, interactive evolution of strategy game levels requires the user to visually inspect each of them and DeLeNoX focuses on the appearance of spaceship hulls and uses autoencoders as pattern detectors of

images. The focus on visual appearance naturally limits how much the conclusions drawn from the interface design and user tests of such tools can be generalized. For instance, an editor of a game's story (or dialog) would have difficulties introducing more than a couple of generated suggestions to the user due to the cognitive overload of reading text. By contrast, the map sketches are simple enough for a user to visually parse at a glance, and thus even 12 generated suggestions can be presented without introducing too much user fatigue. Similarly, the visual feedback in the form of alternative views or metrics in Sentient Sketchbook is not portable in a straightforward way to an editor of game audio or game narrative; coloring spelling errors, grammar errors or style tip word options can be integrated (and have been for text editors or blogging software), but a major redesign of the interface and the algorithms would need to take place. Similarly, there is no straightforward way of implementing constraints or distance measures for other forms of content such as audio or narrative for the purposes of constrained novelty search. While there is interesting work on semantic differences of words based on their appearance in a corpus [167, 8], via WordNet [134] or via Google search queries [252], it is challenging to envision how these distances can be combined with novelty search, or estimate the quality of the final generated narrative artifacts.

11.2 Extensibility

This thesis set out to explore the potential of AI-assisted design via genetic algorithms in the game development process. Considerable effort has been placed both in exploring different methods of integrating computer generation into traditionally human-exclusive roles (summarized in Chapter 10) as well as in exploiting previous findings into a refined, elaborate design tool in the form of Sentient Sketchbook. Unsurprisingly, however, there is a wealth of potential domains and experimental setups which have not been yet investigated at the time of writing. This Section elaborates on future work which could build upon the findings of this thesis, address the limitations identified in Section 11.1, and transfer the algorithmical or design contributions of this thesis to other domains.

11.2.1 Extensibility of FI-2pop GA

Apart from the addition of the offspring boost mechanism, the FI-2pop GAs implemented in this thesis were largely conforming to the literature [123]. Based on the limitations of the FI-2pop GA in terms of performance when multiple fitnesses are aggregated into a weighted sum, there are a number of additions to the two-population paradigm that can enhance its ability to handle multiple objectives. Since any method of searching can be applied to each population individually (as demonstrated by feasible-infeasible novelty search), a multiobjective algorithm can be applied to the feasible population. Inspired by the ϵ-constraint method of multiobjective optimization [42], the feasibility criterion that all but one objectives should have a specific value could be used to optimize only one objective in the feasible population while other objectives remain constant. More realistically, for map sketches the feasibility criterion would be that remaining objectives are above a threshold of acceptable level quality. The infeasible population would contain individuals that do not satisfy this criterion, and would evolve towards improving the remaining objectives as well as reducing the number

of disconnected special tiles. The two-population paradigm can also use lexicographic ordering [42] to improve the quality of its multiobjective optimization. A particularly interesting technique by Kursawe [126] selected an objective at random (although probabilities could be biased) in every step, deleted a portion of the population that was not adequate with regards to this objective and generated offspring up to the population size from the survivors. Such a technique can be combined with two-population approaches, where the inadequate portion of the feasible population is placed in the infeasible population rather than be deleted; the infeasible population can then attempt to improve the fitness of these individuals according to the objective selected for the current step. Finally, the FI-2pop GA could iteratively introduce new constraints as a minimal threshold in a specific dimension. Thus, the FI-2pop GA could begin by minimizing the number of disconnected paths (as it stands currently) and once a large feasible population for these simple constraints has been attained, add an extra constraint on a minimal threshold for one or more quality evaluations; when a large feasible population for these new criteria is attained, the threshold of previous criteria can be increased or more thresholds in other quality evaluations can be added.

11.2.2 Extensibility of Constrained Novelty Search

Novelty search (and its constrained variant) has been used within this thesis to generate visually dissimilar map sketches which are expected to inspire the human user in Sentient Sketchbook, to create an initial seed for neural networks representing game terrain in Sentient World, and to diversify according to patterns discovered in previous generated artifacts in DeLeNoX. However, constrained novelty search can also be applied to engineering tasks, such as robotics. Robot locomotion is a domain where novelty search has been used quite successfully in recent years; the work of Risi and Stanley [196] and Morse et al. [170] uses novelty search to evolve quadruped robot controllers via HyperNEAT [232]. Unlike the search for visual diversity used in the different tools of this thesis, these robot controllers evolve to diversify a *behavioral* characterization, e.g. the robots' positions sampled at certain intervals during a simulation. There are a number of constraints on the behavior of a robot controller: for instance, Risi and Stanley [196] stop the simulation which evaluates a robot controller if the robot's torso falls under a specific height. Currently controllers deemed infeasible are assigned a novelty score of zero, amounting to minimal criteria novelty search. However, such infeasible controllers can be retained in a separate population which evolves either to maximize the diversity of their behavior via FI2NS or to minimize their distance from feasibility via FINS; the latter can be accomplished by awarding a high fitness to individuals which "fall" at a later time step than others, or which can return to an upright position after falling.

11.2.3 Extensibility of Adaptive Models

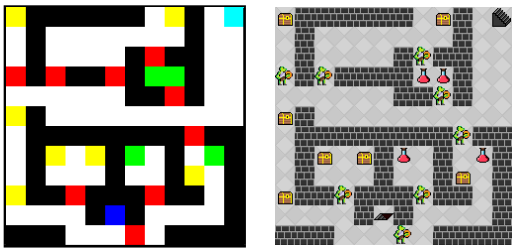
Regarding the designer modeling extensions of Sentient Sketchbook, the three models of style, process and symmetry goals focus on different aspects such as visual appearance or consistent preference. There is great potential in combining these different models in a meaningful way that obfuscates their individual limitations. Different symmetry values (sym) could be used as fitness dimensions for modeling style, along with (or instead of) strategic qualities; in that case, the model would discover a persistent visual style and bias search towards it. The

current model of style, on the other hand, could inform the search for new suggestions during the periods when the model of process is not being adapted due to lack of base tiles on the user's map sketches. Finally, the model of style could incorporate the detection of new features in a way similar to DeLeNoX, and thus not limit itself to six strategic qualities; this is likely easier to accomplish with a database of choices from multiple designers, allowing for a sufficient data volume to facilitate learning.

Regarding adaptive models in general, their biggest limitation is the bias introduced by the fitness dimensions which comprise them. Most experiments reported in this thesis use hand-crafted, more or less carefully designed fitness functions which have similar value ranges on "typical artifacts". However, the assumptions made for what designers are interested in when designing a game level may not hold in all circumstances. The lack of reported experiments with human users clearly points out that the immediate next steps consist of testing the different designer modeling additions with actual designers. Preliminary tests with human users on the task of creating spaceship hulls [138] showed that the visual patterns detected by the fitness functions were successfully perpetuated in future generations based on the users' choices; however, other patterns such as spaceships' similarity to real-world objects or animals were obviously not recognized by the adaptive model. Preliminary tests with human users on interactively evolving the complete strategy game maps of Section 10.2 showed that standard interactive evolution was as likely to create maps fitting to the users' preferences as rank-based interactive evolution. More experiments with human users would help finetune the fitness dimensions or inspire the introduction of others. Of particular interest are fitness dimensions of visual taste such as symmetry, which were introduced for the strategy game maps of Section 10.2 but were replaced by a different method of creating symmetric maps in Sentient Sketchbook's designer model of symmetry goals. For extensions and future work on adaptive models in a larger scale, the gameplay or visual properties of preferred game levels or other artifacts could be automatically learned via pattern detection algorithms. As an extension of DeLeNoX, for instance, the user could designate which spaceships are more appealing and the output of the autoencoders could be used to classify between desirable and undesirable content; this classifier can then be used as an additional constraint, with undesirable content placed in the infeasible population, or as a fitness function which tries to optimize the desirability of generated artifacts based not on hand-crafted but rather on entirely automatically derived quality metrics.

11.2.4 Extensibility of Sentient Sketchbook

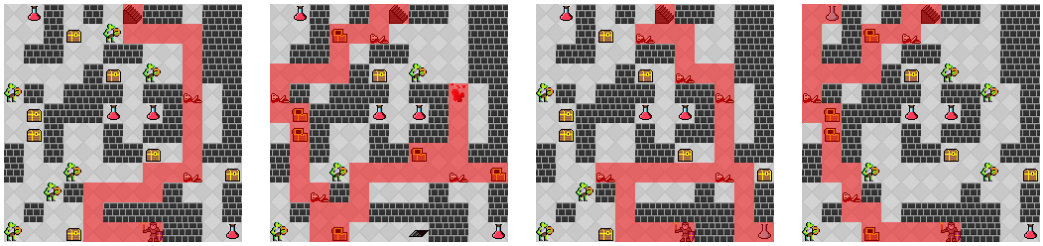
A primary weakness of the current version of Sentient Sketchbook is that it only generates (and evaluates) the initial state of the level. The human designers must hypothesize about the progress of a game played in the map sketch they are working on based only on their own intuition and design experience — without support from the tool. Since a primary goal of mixed-initiative design tools is to impart some expert knowledge to end-users and reduce their cognitive efforts, Sentient Sketchbook should be enhanced to take into account either real or simulated gameplay. The temporal aspect and interactions within games can provide additional information and evaluation criteria for level quality; for certain game genres such as tower defense, this temporal aspect is of utmost importance [261]. In order to accomplish that, a domain-specific AI agent can simulate a playthrough in a level; in adversarial games such as



(a) Sketch (b) MiniDungeon level

Fig. 11.1: A MiniDungeon sketch and the final level it creates, with impassable tiles in black, enemies in red, treasures in yellow, potions in green, level entrance in cyan and level exit in blue.

.

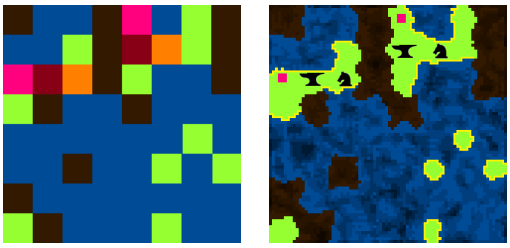


(a) Speedrunner (b) Treasure Collector (c) Monster Killer (d) Clone of a Human Player

Fig. 11.2: Example agent playtraces of a MiniDungeons level, showing different play personas.

FPS levels or strategy game maps, the AI agent would need to play against itself in self-play trials as performed by Browne and Maire [28]. An important factor for the applicability of simulations in a design environment is the time required for a full playthrough. Strategy games, for instance, often take more than 30 minutes for a human playthrough and even computer simulations often take several minutes; such a time investment is likely unacceptable during design, when a small change may require another simulation to be performed. A solution to this problem could be to use an abstraction of the actual gameplay, much like how the sketch is an abstraction of the final level. For instance, combat in a dungeon can be abstracted away as a drain in the hero's resources by a ratio of the monster's challenge rating, while strategic gameplay can be simulated by cellular automata or even flood fill algorithms — much in the way exploration is being evaluated in eq. (5.2).

There has already been work towards creating simulated gameplay that matches "typical" human play. This line of work uses map sketches (see Fig. 11.1) which correspond to dungeons — not unlike those of Section 7.2.1 — used in MiniDungeons, a prototype puzzle-roguelike game. Taking the sketch abstraction to the gameplay level, monsters are immobile and inflict a random number of hit point damage to the player before dying, simulating a potentially elaborate combat sequence with a single move action. The hero avatar begins at the level entrance and continues to the next level once it reaches the level exit; along the way, it can interact with monsters, collect treasures, or drink potions to restore its hit points. Artificial controllers have been trained via Q-learning [100] or via evolution [99] based on a priori defined personas which represent archetypical utility functions of human players (e.g. runner,



(a) Sketch (b) Civilization-style level

Fig. 11.3: A Civilization-style sketch and the final level it creates, with mountains in black, sea in blue, land in green, settlers' starting positions in magenta, horse resources in yellow and iron resources in red.

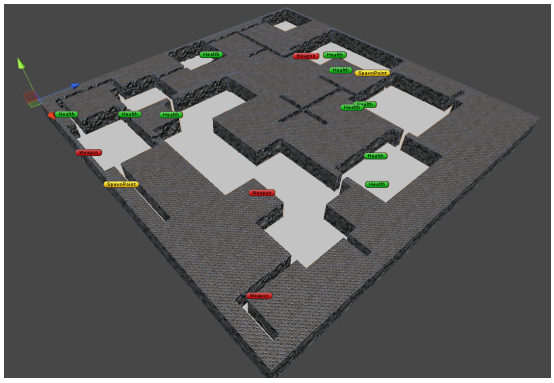
.

monster killer, treasure collector as seen in Fig. 11.2). Comparing such personas with human playtraces across the same levels, a good match is achieved between players and personas and thus it can be assumed that the artificial agents are representative of human play; as an extension, agents can even be evolved to match one human user explicitly [101] as per Fig. 11.2d. Using these personas to playtest unseen generated map sketches can provide a gameplay-based evaluation of dungeon quality, e.g. by measuring the number of steps needed for the speedrunner to complete the level or the number of treasures collected by the treasure collector. Moreover, these personas can be used as constraints, e.g. by integrating a constraint that all personas must be able to complete the level without dying at least 50% of the time (due to the randomness of combat). Finally, novelty search can search for game levels where the sequence of actions of the same personas are different from the rest of the population, replacing visual difference with gameplay difference as the distance function.

Other ways of integrating gameplay into the map sketches can be achieved using existing frameworks for AI control in strategy games, e.g. the μRTS framework of Ontañon [178] which operates on a similar map size as that of strategy game map sketches. On the other hand, simulations of strategic quality can be sped up by removing any notion of strategy and replacing it with rule-based processes such as cellular automata or limited intelligence processes such as swarm movements. Preliminary work in simulating strategy gameplay in *FreeCiv* (The Freeciv project 1996) via such processes is currently under way, while Civilization-style maps can be generated as sketches in the manner shown in Fig. 11.3.

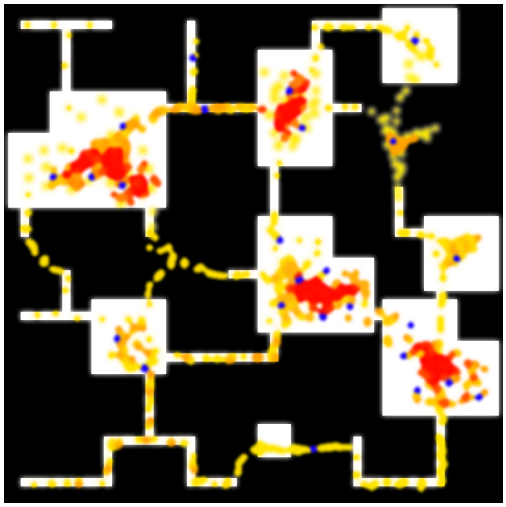
Finally, human players can be used for evaluating the game levels instead of simulated users; this is most appropriate outside of the game development process and falls outside the scope of mixed-initiative design. However, there has been preliminary work in transforming first person shooter map sketches into 3D destructible levels (see Fig. 11.4) and evaluating how human users interact with them (and each other) in multiplayer games. A potential extension could use gameplay metrics (e.g. number of deaths, shots fired, wins) or reported experience as a fitness function for generating the next map sketch/3D level for the players.

A common pattern between the three types of game levels investigated in Chapter 7 is the top-down view of the map sketch and the implicit assumption of an undirected graph for navigation. For games where backtracking is not part of the intended experience, such as

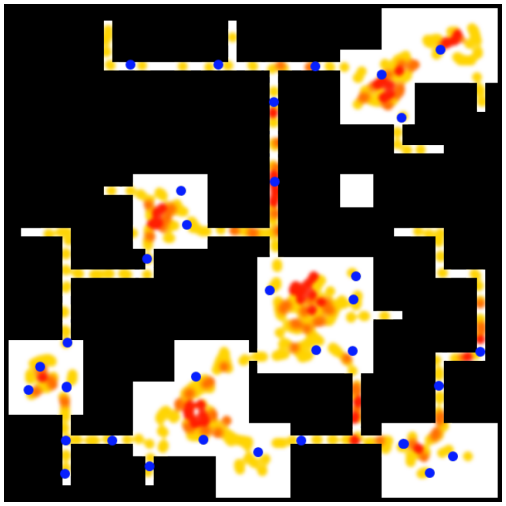


(a) 3D visualization of FPS level.

(b) Gameplay in the FPS.



(c) Heatmaps of FPS games with large voxels.



(d) Heatmaps of FPS games with small voxels.

Fig. 11.4: FPS game using Sentient Sketchbook maps in the same format as Section 7.2.2. Heatmaps show player positions (warm colors) and deaths (blue dots), collected from multiple two-player matches. In Fig. 11.4c large wall "chunks" (voxels) are destroyed with every shot, while in Fig. 11.4d the chunks are smaller. More "tunnels" and unexpected pathways are created with large voxels, while gameplay is more predictable with small voxels. The implementation and experiments are reported in "Map sketch evaluation through a complete multiplayer game featuring destructible terrain" by Lasse Astrup Jørgensen.

.

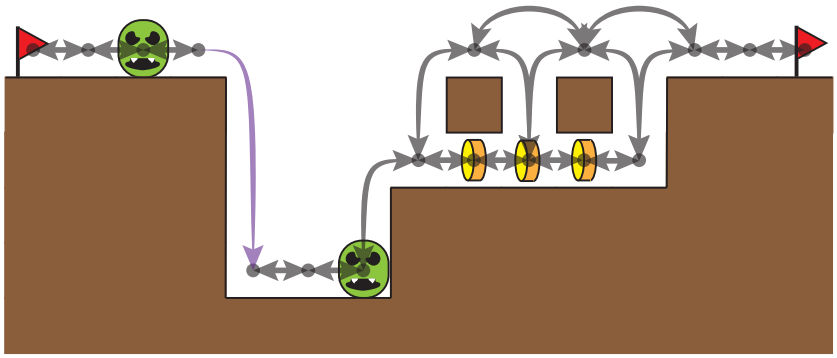
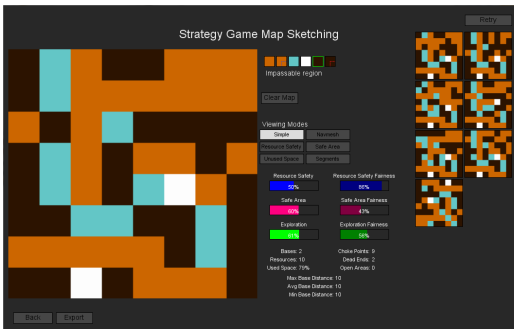


Fig. 11.5: An example of how a platformer level can be evaluated with the current formulas. The level starts and ends at the flags, enemies are in green and rewards as yellow coins; the player is assumed to jump up to 2 blocks. Unlike the current method, the platformer needs a directed graph, i.e. the first monster (top) can reach the second (bottom) but not vice versa since the purple arrow is one-directional; the player can't jump back to the ledge. The coin area is an example of high exploration, as there are multiple paths to the finish.

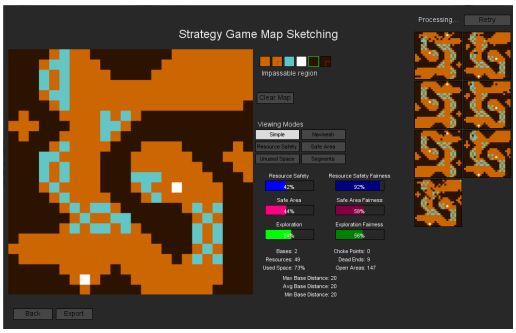
platformers, a directed graph would better represent the level progression (e.g. a cliff which an avatar can jump off but not climb up again). Such a directed graph can easily be integrated with the existing heuristics of strategic resource and area control, while exploration can be simulated as breadth-first-search. Figure 11.5 shows an example platformer level and how it can be evaluated with a directed graph. Similarly to platformer levels, adventure games — especially those featuring lock and key puzzles — can be evaluated via a directed graph to ensure that the key is reachable before the locked door is opened.

Another extension of Sentient Sketchbook could integrate the human authorship or procedural generation of the game rules which are in effect in the level. For instance, the designer can introduce flying units in the strategy game map sketches of Sentient Sketchbook, essentially changing how exploration should be evaluated. In that regard, simulation-based evaluations would be more flexible in integrating such rule changes provided that a sufficiently generic gameplaying agent was in place. The computer could potentially also suggest rule changes, such as a new jump mechanic which would decrease the difficulty of a platformer level. Work on rule generation to fit a specific level design has been performed via evolution [46] and via constraints applied to a planner [272], and could be applied to the map sketches of Sentient Sketchbook.

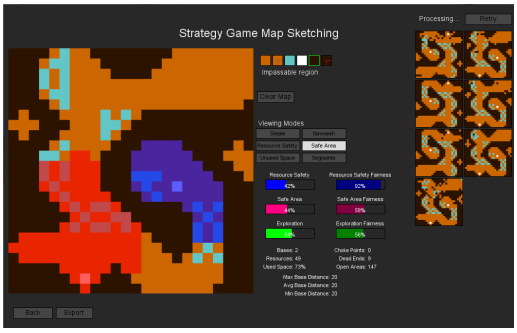
Finally, the notion of iterative refining as exemplified by Sentient World in Section 10.1 can be introduced to Sentient Sketchbook as a way of alternating the level of detail of the map sketches. Sentient Sketchbook currently operates on map sketches containing the absolute minimum in terms of special tiles and map resolution, allowing for fast prototyping. However, it may be desirable to grant the user more control over the generated results instead of relying completely on the algorithms that generate the final map as per Section 5.4.3. Following the paradigm of Sentient World, a designer may wish to provide the rough outline of a level for *Starcraft* (Blizzard 1998) using an 8 by 8 tile grid and using resources to symbolize both vespene gas and minerals, but by clicking a "refine" button on the interface can be brought to a sketch of a higher resolution (which still respects the specifications of the user's rough outline). In this refined map sketch, the user can finetune the position of impassable tiles on



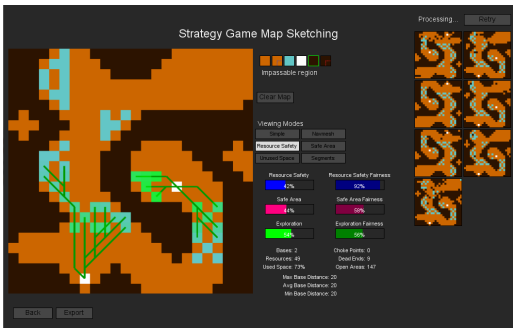
(a) Abstract map sketch



(b) Refined map sketch



(c) Safe Areas display of refined map



(d) Resource Safety display of refined map

Fig. 11.6: A mockup of how iterative refining can be integrated into Sentient Sketchbook to create map sketches of varying levels of detail. The design process can begin on the abstract map sketch of Fig.11.6a, and once the user desires they can "zoom" in to a less abstract version of the map (Fig. 11.6b) which is generated by rule-based algorithms. The same visualizations can be available on the more refined map (Fig. 11.6c–11.6d).

the higher resolution map, and can specify which resources are vespene gas and which ones are minerals. Figure 11.6 visualizes this process. Understandably, it is not possible to revert from the refined map to the rough outline without loss of information, so the design process will likely be one-directional. Moreover, the increased resolution of refined maps will likely need a new representation for the genetic algorithms as well as new evaluation functions for level quality, since Section 11.1.1 argued that large maps can not be generated in realtime using the current format. Likely the suggestions at the refined map level would need to focus more on local changes (e.g. moving a few resources around or swapping minerals with vespene gas) rather than large variations from the user's sketch. This is not only more computationally efficient, but corresponds better to the design stage where designers "finetune" the small details of their levels and do not wish for completely new ideas.

11.2.5 Beyond Levels, Beyond Games

All of the algorithms introduced in this thesis can be applied to problems outside that of mixed-initiative game design; a number of possible extensions have been suggested in their respective sections, such as applying constrained novelty search to robot locomotion. However,

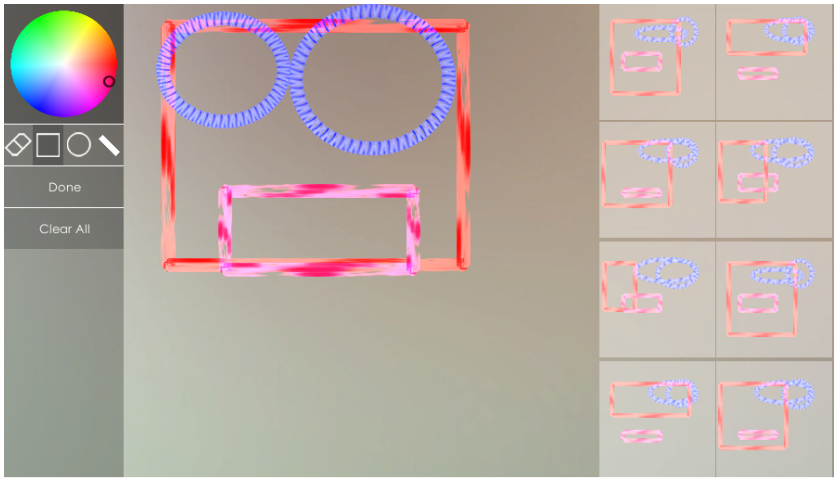


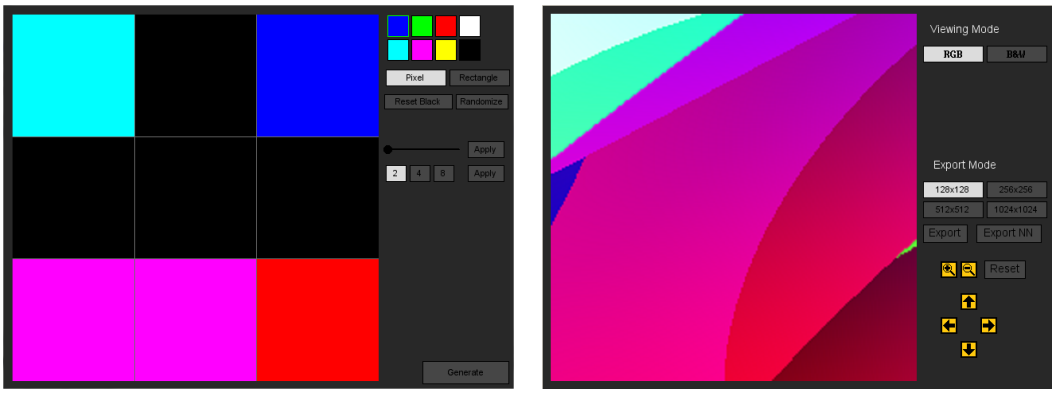
Fig. 11.7: The C^{2}Create graphical user interface: the canvas is at the center of the screen, with the toolbox to the left and the suggestions to the right.

the greater concept of mixed-initiative interaction via sketching can also find applications in industrial design or digital art.

Sketching has obviously been a large inspiration for most Computer-Aided Design applications since Sketchpad [237] in the 1960s. Transferring some of the sketching initiative of industrial CAD tools, which is still predominantly human, to computational processes such as genetic algorithms could be an interesting extension of the work presented here. In essence, the ideal application would combine the strengths of easy-to-use modern interfaces for design, from *SketchUp* (Trimble Navigation 2000) to *ZBrush* (Pixologic 2013), with evolutionary algorithms for design and engineering [106, 32, 115] in a fashion which would allow the human user to specify a design and receive suggestions by the evolutionary algorithm on mutations of that design which satisfy certain criteria and/or optimize certain performance or aesthetic properties important to the design.

The mixed-initiative design paradigm suggested in this thesis can also be applied to digital art; since all of the prototypes focus on the visual aspects of design, they can be re-appropriated for creating 2D art. Arguably, using choice-based or rank-based interactive evolution for creating 2D art is similar to existing work on partially interactive artists [150]. Moreover, art does not come with easily identifiable constraints such as the playability constraints of game levels; however, certain fundamental qualities of images can be integrated as constraints — e.g. that images exhibit more than one color in order to be remotely interesting. The C^{2}Create tool (Fig. 11.7) is an example of an interface for drawing the appearance of game assets such as the artwork of playing cards [147]. In C^{2}Create, the user's drawing is used to seed a population of suggestions which evolves according to constrained novelty search: the novelty metric is the pixel-to-pixel difference between two drawings while some minimal constraints are imposed so that generated suggestions cover enough size on the canvas without exceeding the canvas' borders.

Another potential mixed-initiative tool for digital art is Sentient World: while it currently generates heightmaps for use as gamewords, minimal changes — such as increasing the number of network outputs to three for RGB or HSB formats — could allow it to become a tool for



(a) Low-resolution user drawing (b) Full resolution image

Fig. 11.8: A mockup of how Sentient World can be re-appropriated as a drawing tool. The user draws using a limited color palette in Fig. 11.8a (black tiles assumed wildcards that can be of any color), and through iterative refining can create images of infinite resolution as shown in Fig. 11.8b.

visual artists. Using its population of neural networks in a similar way that PicBreeder [209] uses CPPNs [230] to generate images, Sentient World can create images of infinite resolution and color variety. In this hypothetical scenario visualized in Fig. 11.8, the human artist begins by creating a rough sketch with a few primary colors and through the iterative refining process can generate a final image which can be saved in any resolution and printed in any canvas size.

11.3 Summary

This Chapter concludes the thesis with general insights from the research performed while exploring the range of mixed-initiative design and the extent of computational initiative. While developing and evaluating the algorithms for Sentient Sketchbook and other prototypes of Chapter 10, certain limitations on the performance of algorithms and the generalizability of experiments' conclusions were found. Primarily limitations pertain to the way multiple objectives are handled, the choice of diversity measure and the assumptions regarding designers' preferences. Apart from addressing these limitations via tweaks to the algorithms and more experiments or user tests, several extensions of both the algorithms and of the proposed mixed-initiative paradigm were suggested. Provided a suitably abstract representation, the use of genetic algorithms during the design process is possible even for real-time user feedback and a proactive computational designer can foster mixed-initiative co-creativity not only during level design but also in other creative tasks such as engineering or digital art.

Bibliography

[1] Aristotle and Heath, M. 1996. Poetics. Penguin books. Penguin Adult. 23

[2] Arnheim, R. 2004. Art and visual perception: a psychology of the creative eye. University of California Press, revised and expanded edition. 22, 23, 72, 132, 133

[3] Ashlock, D. and McGuinness, C. 2013. Landscape automata for search based procedural content generation. *In* Proceedings of IEEE Conference on Computational Intelligence and Games. 18

[4] Asteriadis, S., Karpouzis, K., Shaker, N., and Yannakakis, G. N. 2012. Towards detecting clusters of players using visual and gameplay behavioral cues. *In* Proceedings of the International Conference on Games and Virtual Worlds for Serious Applications, pp. 140–147. 22

[5] Avedon, E. M. and Sutton-Smith, B. 1971. The study of games. J. Wiley. 9

[6] Bal, M. and van Boheemen, C. 2009. Narratology: Introduction to the Theory of Narrative. University of Toronto Press, 3rd edition. 22

[7] Baluja, S., Pomerleau, D., and Jochem, T. 1999. Towards automated artificial evolution for computer-generated images. *Musical networks* pp. 341–370. 23

[8] Baroni, M. and Lenci, A. 2010. Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics* 36:673–721. 156

[9] Bauer, A. and Popović, Z. 2012. Rrt-based game level analysis, visualization, and visual refinement. *In* Proceedings of Artificial Intelligence and Interactive Digital Entertainment. 155

[10] Beaney, M. 2005. Imagination and creativity. Open University Milton Keynes, UK. 24

[11] Bengio, Y. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning* 2:1–127. 30, 140, 148

[12] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. 2007. Greedy layer-wise training of deep networks. *In* Proceedings of Advances in Neural Information Processing Systems 19 (NIPS'06), pp. 153–160. 29

[13] Bentley, P. 1999. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. *In* Proceedings of the Genetic and Evolutionary Computation Conference, pp. 35–43. 39, 40

[14] Benyon, D. and Murray, D. 1993. Applying user modeling to human computer interaction design. *Artificial Intelligence Review* 7:199–225. 19

[15] Biles, J. A. Genjam: A genetic algorithm for generating jazz solos. 23, 37

[16] Binsted, K., Bergen, B., Coulson, S., Nijholt, A., Stock, O., Strapparava, C., Ritchie, G., Manurung, R., Pain, H., Waller, A., and O'Mara, D. 2006. Computational humor. *IEEE Intelligent Systems* 21:59–69. 23

[17] Binsted, K. and Ritchie, G. 1994. An implemented model of punning riddles. *In* Proceedings of the 12th National Conference on Artificial Intelligence, pp. 633–638. 23

[18] Birren, F. 1978. Color & Human Response: Aspects of Light and Color Bearing on the Reactions of Living Things and the Welfare of Human Beings. Van Nostrand Reinhold. 22

[19] Bjork, S. and Holopainen, J. 2004. Patterns in Game Design. Charles River Media. 52, 53

[20] Björnsson, Y., Enzenberger, M., Holte, R. C., and Schaeffer, J. 2005. Fringe search: Beating A* at pathfinding on game maps. *In* Proceedings of the IEEE Conference on Computational Intelligence and Games. 14

[21] Blair, D. and Meyer, T. 1997. Tools for an interactive virtual cinema. *In* Creating Personalities for Synthetic Actors, Towards Autonomous Personality Agents, pp. 83–91. 15

[22] Boden, M. 1990. The Creative Mind. Abacus. 22

[23] Boden, M. A. 2003. The creative mind: Myths and mechanisms. Routledge. 25

[24] Booker, C. 2004. The Seven Basic Plots: Why We Tell Stories. The Seven Basic Plots: Why We Tell Stories. Continuum. 22

[25] Boser, B. E., Guyon, I. M., and Vapnik, V. N. 1992. A training algorithm for optimal margin classifiers. *In* Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pp. 144–152. 27

[26] Bowman, C. 2010. Explorations of visual representation: Towards a language of movement. *In* Proceedings of the International Symposium on Electronic Art, pp. 171–173. 23

[27] Bown, O. 2014. Empirically grounding the evaluation of creative systems: Incorporating interaction design. *In* Proceedings of the International Conference on Computational Creativity. 22

[28] Browne, C. and Maire, F. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2:1–16. 18, 159

[29] Buro, M. 2003. Real-time strategy games: A new AI research challenge. *In* Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, pp. 1534–1535. 14

[30] Buxton, B. 2007. Sketching User Experiences: Getting the Design Right and the Right Design. Morgan Kaufmann. 51

[31] Caillois, R. 1961. Man, Play and Games. University of Illinois Press. 9, 10

[32] Campana, E. F., Liuzzi, G., Lucidi, S., Peri, D., Piccialli, V., and Pinto, A. 2009. New global optimization methods for ship design problems. *Optimization and Engineering* 10:533–555. 164

[33] Cardamone, L., Loiacono, D., and Lanzi, P. L. 2011. Interactive evolution for the procedural generation of tracks in a high-end racing game. *Interface* pp. 395–402. 18, 39

[34] Cazenave, T. 2006. Optimizations of data structures, heuristics and algorithms for path-finding on maps. *In* Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 27–33. 14

[35] Champandard, A. J. 2002. The dark art of neural networks. *In* AI Game Programming Wisdom. 28

[36] Cheng, P. C.-H., Lowe, R. K., and Scaife, M. 2001. Cognitive science approaches to understanding diagrammatic representations. *Artificial Intelligence Review* 15:79–94. 2, 24

[37] Cho, S.-B. and Lee, J.-Y. 1999. Emotional image retrieval with interactive evolutionary computation, pp. 57–66. *In* Advances in Soft Computing. Springer London. 37

[38] Churchill, D. and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in starcraft. *In* Proceedings of the IEEE Conference on Computational Intelligence and Games, pp. 1–8. 14

[39] Clark, A. 1998. Being there: Putting brain, body, and world together again. MIT press. 24

[40] Clune, J. and Lipson, H. 2011. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. pp. 141–148. 40

[41] Clune J., L. H. 37

[42] Coello Coello, C. A. 1999. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information systems* 1:269–308. 33, 79, 83, 156, 157

[43] Coello Coello, C. A. 2010. Constraint-handling techniques used with evolutionary algorithms. *In* Proceedings of the Genetic and Evolutionary Computation Conference, pp. 2603–2624. ACM. 34

[44] Colton, S., Charnley, J., and Pease, A. 2011. Computational Creativity Theory: The FACE and IDEA models. *In* Proceedings of the 2nd International Conference on Computational Creativity. 22

[45] Cook, M., Colton, S., and Pease, A. 2012. Aesthetic considerations for automated platformer design. *In* Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference. 15

[46] Cook, M., Colton, S., Raad, A., and Gow, J. 2013. Mechanic miner: Reflection-driven game mechanic discovery and level design. *In* Proceedings of Applications of Evolutionary Computation, pp. 284–293. 162

[47] Correia, J., Machado, P., Romero, J., and Carballal, A. 2013a. Evolving figurative images using expression-based evolutionary art. *In* Proceedings of the International Conference on Computational Creativity, pp. 24–31. 23

[48] Correia, J., Machado, P., Romero, J., and Carballal, A. 2013b. Feature selection and novelty in computational aesthetics. *In* Evolutionary and Biologically Inspired Music, Sound, Art and Design - Second International Conference, EvoMUSART 2013, Vienna, Austria, April 3-5, 2013. Proceedings, volume 7834 of *Lecture Notes in Computer Science*, pp. 133–144. Springer. 23

[49] Cortes, C. and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20:273–297. 27

[50] Crawford, C. 2004. Chris Crawford on Interactive Storytelling. New Riders. 15

[51] Damásio, A. 1994. Descartes' Error: Emotion, Reason, and the Human Brain. Putnam Publishing. 22

[52] Damasio, A. R. 2001. Some notes on brain, imagination and creativity. *The origins of creativity* pp. 59–68. 23

[53] Davis, I. L. 1999. Strategies for strategy game AI. Technical Report SS-99-02. 14

[54] Dawkins, R. 1986. The Blind Watchmaker. W. W. Norton & Company. 37

[55] De Bono, E. 1970. Lateral thinking: creativity step by step. Harper & Row. 2, 23, 24, 44, 140

[56] Deb, K. 2000. An efficient constraint handling method for genetic algorithms. 186:311–338. 34

[57] Dipaola, S., Carlson, K., McCaig, G., Salevati, S., and Sorenson, N. 2013. Adaptation of an autonomous creative evolutionary system for real world design application based on creative cognition. *In* Proceedings of the International Conference on Computational Creativity. 37

[58] Drachen, A. and Canossa, A. 2009. Analyzing spatial user behavior in computer games using geographic information systems. *In* Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era, pp. 182–189. 16

[59] Drachen, A., Canossa, A., and Yannakakis, G. N. 2009. Player modeling using self-organization in tomb raider: Underworld. *In* Proceedings of IEEE conference on Computational Intelligence and Games. 19

[60] Drachen, A., Sifa, R., Bauckhage, C., and Thurau, C. 2012. Guns, swords and data: Clustering of player behavior in computer games in the wild. *In* Proceedings of IEEE conference on Computational Intelligence and Games, pp. 163–170. 16

[61] Dubbin, G. A. and Stanley, K. O. 2010. Learning to dance through interactive evolution. *In* Proceedings of Applications of Evolutionary Computation, pp. 331–340. 40

[62] Dunn, O. 2012. Multiple comparisons among means. *Journal of the American Statistical Association* 56:52–64. 98

[63] Dyer, M. G. 1983. In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension. MIT Press. 15

[64] Ebner, M., Reinhardt, M., and Albert, J. 2005. Evolution of vertex and pixel shaders, pp. 261–270. *In* Genetic Programming, volume 3447 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 37

[65] Edmonds, E. A., Bilda, Z., and Muller, L. 2009. Artist, evaluator and curator: three viewpoints on interactive art, evaluation and audience experience. *Digital Creativity* 20:141–151. 23

[66] Ekeus, H., Abdallah, S., Plumbley, M., and McOwan, P. W. 2012. The melody triangle: Exploring pattern and predictability in music. *In* Proceedings of the AIIDE workshop on Musical Metacreation. 23

[67] Entertainment Software Association 2012. Essential facts about the computer and video game industry. Retrieved December 12, 2012 from http://www.theesa.com/facts/pdfs/ESA_EF_2012.pdf. 1

[68] Feldman, L. 1995. Valence focus and arousal focus: Individual differences in the structure of affective experience. 69:53–166. 19

[69] Fogel, D. 1995. Phenotypes, genotypes, and operators in evolutionary computation. 39

[70] Forbus, K. D., Mahoney, J. V., and Dill, K. 2002. How qualitative spatial reasoning can improve strategy game AIs. *IEEE Intelligent Systems* 17:25–30. 14

[71] Frasca, G. 1999. Ludology meets narratology: Similitude and differences between (video)games and narrative. Originally published in Finnish in Parnasso 1999: 3, 365-71. Online: ludology.org. Accessed 28 Apr 2014. 10

[72] Frome, J. 2007. Eight ways videogames generate emotion. *In* Proceedings of DiGRA. 19

[73] Fuller, D. and Magerko, B. 2011. Shared mental models in improvisational theatre. *In* Proceedings of the 8th ACM conference on Creativity and cognition, pp. 269–278. 15

[74] Fullerton, T., Swain, C., and Hoffman, S. 2004. Game Design Workshop: Designing, Prototyping and Playtesting Games. CMP Books. 9

[75] Fürnkranz, J. and Hüllermeier, E. 2010a. Preference Learning. Springer-Verlag New York, Inc. 20, 29

[76] Fürnkranz, J. and Hüllermeier, E. 2010b. Preference learning, pp. 789–795. *In* Encyclopedia of Machine Learning. Springer US. 28, 29

[77] Gage, J. 1999. Color and Culture: Practice and Meaning from Antiquity to Abstraction. University of California Press. 22

[78] Galanter, P. The problem with evolutionary art is ... *In* Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part II. 23

[79] Gauci, J. and Stanley, K. 2007. Generating large-scale neural networks through discovering geometric regularities. *In* Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 997–1004. 40

[80] Gaver, W. W., Beaver, J., and Benford, S. 2003. Ambiguity as a resource for design. *In* Proceedings of the SIGCHI conference on Human factors in computing systems. 52

[81] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. 2012. Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. 17

[82] Genesereth, M. R., Love, N., and Pell, B. 2005. General game playing: Overview of the AAAI competition. pp. 62–72. 14

[83] Gervás, P. and León, C. 2014. Reading and writing as a creative cycle: the need for a computational model. *In* Proceedings of the International Conference on Computational Creativity. 23

[84] Giannatos, S., Nelson, M. J., Cheong, Y.-G., and Yannakakis, G. N. 2011. Suggesting new plot elements for an interactive story. *In* Proceedings of the AIIDE Workshop on Intelligent Narrative Technologies. 23

[85] Goldberg, D. 1989. Genetic algorithms in search, optimization, and machine learning. Addison-wesley. 132

[86] Gomez, F. and Mikkulainen, R. 1997. Incremental evolution of complex general behavior. *Adaptive Behavior* 5:317–342. 40

[87] Goodman, B. A. and Litman, D. J. 1992. On the interaction between plan recognition and intelligent interfaces. *User Modeling and User-Adapted Interaction* 2:83–115. 21

[88] Grace, K., Gero, J., and Saunders, R. Learning how to reinterpret creative problems. *In* Proceedings of International Conference on Computational Creativity. 140

[89] Graf, J. Interactive evolutionary algorithms in design. *In* Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms. 37

[90] Gruau, F., Whitley, D., and Pyeatt, L. 1996. A comparison between cellular encoding and direct encoding for genetic neural networks. *In* Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, pp. 81–89. 40

[91] Hadj-Alouane, A. 1993. A genetic algorithm for the multiple-choice integer program. Technical report, University of Michigan. Department of Industrial and Operations Engineering. 34

[92] Hao, Y. and Veale, T. 2010. An ironic fist in a velvet glove: Creative misrepresentation in the construction of ironic similes. *Minds and Machines* 20:635–650. 23

[93] Hastings, E. J., Guha, R. K., and Stanley, K. O. 2009a. Evolving content in the galactic arms race video game. *In* Proceedings of IEEE Conference on Computational Intelligence and Games, pp. 241–248. 18

[94] Hastings, E. J., Guha, R. K., and Stanley, K. O. 2009b. Interactive evolution of particle systems for computer graphics and animation. *Transactions of Evolutionary Computation* 13:418–432. 40

[95] Haykin, S. 1998. Neural Networks: A Comprehensive Foundation. Prentice Hall PTR, 2nd edition. 28

[96] Hinton, G. E. and Zemel, R. S. Autoencoders, minimum description length, and Helmholtz free energy. *In* Advances in Neural Information Processing Systems. 30

[97] Hoang, H., Lee-Urban, S., and Muñoz-Avila, H. 2005. Hierarchical plan representations for encoding strategic game AI. *In* Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 63–68. 14

[98] Holland, J. H. 1992. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press. 31, 32

[99] Holmgård, C., Liapis, A., Togelius, J., and Yannakakis, G. N. 2014a. Evolving personas for player decision modeling. *In* Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG). 159

[100] Holmgård, C., Liapis, A., Togelius, J., and Yannakakis, G. N. 2014b. Generative agents for player decision modeling in games. *In* Poster Proceedings of the 9th Conference on the Foundations of Digital Games. 159

[101] Holmgård, C., Liapis, A., Togelius, J., and Yannakakis, G. N. 2014c. Personas versus clones for player decision modeling. *In* Proceedings of the International Conference on Entertainment Computing (ICEC). 160

[102] Homaifar, A., Qi, C. X., and Lai, S. H. 1994. Constrained optimization via genetic algorithms. 62:242–253. 34

[103] Hoover, A. K. and Stanley, K. O. 2009. Exploiting functional relationships in musical composition. *Connection Science Special Issue on Music, Brain & Cognition* 21:227–251. 40

[104] Hoover, A. K., Szerlip, P. A., and Stanley, K. O. 2011a. Interactively evolving harmonies through functional scaffolding. *In* Proceedings of Genetic and Evolutionary Computation Conference. 23, 37

[105] Hoover, A. K., Szerlip, P. A., and Stanley, K. O. 2011b. Interactively evolving harmonies through functional scaffolding. *In* Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 387–394. 40, 130

[106] Hornby, G. S. 2004. Functional scalability through generative representations: the evolution of table designs. *Environment and Planning B: Planning and Design* 31:569–587. 164

[107] Horswill, I. D. 2009. Lightweight procedural animation with believable physical interactions. *IEEE Transactions on Computational Intelligence and AI in Games* 1:39–49. 14

[108] Horswill, I. D. and Foged, L. 2012. Fast procedural level population with playability constraints. *In* Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference. 17

[109] Howlett, A., Colton, S., and Browne, C. 2010. Evolving pixel shaders for the prototype video game subversion. *In* AI and Games Symposium (AISB'10). 18

[110] Hsu, F. C. and Chen, J.-S. 1999. A study on multi criteria decision making model: interactive genetic algorithms approach. *In* IEEE International Conference on Systems, Man, and Cybernetics, volume 3, pp. 634–639. 38

[111] Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. 2008. Label ranking by learning pairwise preferences. *Artificial Intelligence* 172:1897–1916. 29

[112] Hunicke, R., Leblanc, M., and Zubek, R. 2004. MDA: A formal approach to game design and game research. 11

[113] Jacob, B. L. 1995. Composing with genetic algorithms. Technical report. 23

[114] Jakobsen, T. 2001. Advanced character physics. *In* Proceedings of the Game Development Conference. 14

[115] Janssen, P. and Kaushik, V. 2013. Decision chain encoding: Evolutionary design optimization with complex constraints. *In* Proceedings of Evolutionary and Biologically Inspired Music, Sound, Art and Design, volume 7834 of *Lecture Notes in Computer Science*, pp. 157–167. Springer Berlin Heidelberg. 164

[116] Järvinen, A. 2002. Gran stylissimo: The audiovisual elements and styles in computer and video games. *In* CGDC Conference. 10

[117] John, B. E. and Kieras, D. E. 1996. The goms family of user interface analysis techniques: Comparison and contrast. *ACM Transactions of Computer-Human Interaction* 3:320–351. 20

[118] Jong, K. A. D. 2006. Evolutionary computation - a unified approach. MIT Press. 31, 32

[119] Jordanous, A. K. 2013. Evaluating computational creativity: a standardised procedure for evaluating creative systems and its application. PhD thesis, University of Sussex. 22

[120] Juul, J. 2005. Half-Real. MIT Press. 9

[121] Kaelbling, L. P., Littman, M. L., and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285. 27

[122] Kautz, H. A. 1987. A formal theory of plan recognition. PhD thesis, Bell Laboratories. 21

[123] Kimbrough, S. O., Koehler, G. J., Lu, M., and Wood, D. H. 2008. On a feasible-infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research* 190:310–327. 35, 43, 50, 156

[124] Koster, R. and Wright, W. 2004. A Theory of Fun for Game Design. Paraglyph Press. 19

[125] Kramer, O. and Schwefel, H.-P. 2006. On three new approaches to handle constraints within evolution strategies. *Natural Computing* 5:363–385. 35

[126] Kursawe, F. 1991. A variant of evolution strategies for vector optimization. *In* Proceedings of the 1st Workshop on Parallel Problem Solving from Nature, PPSN I, pp. 193–197. Springer-Verlag. 157

[127] Laird, J. E. and van Lent, M. 2000. Human-level AI's killer application: Interactive computer games. *In* Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 1171–1178. AAAI Press. 14

[128] Larkin, J. H. and Simon, H. A. 1987. Why a diagram is (sometimes) worth ten thousand words. *Cognitive science* 11:65–100. 24

[129] Larochelle, H. and Bengio, Y. 2008. Classification using discriminative restricted Boltzmann machines. *In* Proceedings of the Twenty-fifth International Conference on Machine Learning, pp. 536–543. 30

[130] Lehman, J. and Stanley, K. O. 2008. Exploiting open-endedness to solve problems through the search for novelty. *In* Proceedings of the International Conference on Artificial Life (ALIFE XI). MIT Press. 35

[131] Lehman, J. and Stanley, K. O. 2010. Revising the evolutionary computation abstraction: Minimal criteria novelty search. *In* Proceedings of the Genetic and Evolutionary Computation Conference. 36, 44, 105

[132] Lehman, J. and Stanley, K. O. 2011a. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation* 19:189–223. 35, 125, 143

[133] Lehman, J. and Stanley, K. O. 2011b. Improving evolvability through novelty search and self-adaptation. *In* IEEE Congress on Evolutionary Computation, pp. 2693–2700. 35

[134] Li, Y., McLean, D., Bandar, Z. A., O'shea, J. D., and Crockett, K. 2006. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering* 18:1138–1150. 156

[135] Liapis, A., Martínez, H. P., Togelius, J., and Yannakakis, G. N. 2013a. Adaptive game level creation through rank-based interactive evolution. *In* Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG). 112

[136] Liapis, A., Yannakakis, G. N., and Togelius, J. 2011a. Neuroevolutionary constrained optimization for content creation. *In* Proceedings of IEEE Conference on Computational Intelligence and Games, pp. 71–78. 18, 35, 40, 130

[137] Liapis, A., Yannakakis, G. N., and Togelius, J. 2011b. Optimizing visual properties of game content through neuroevolution. *In* Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference. 18, 23, 132

[138] Liapis, A., Yannakakis, G. N., and Togelius, J. 2012a. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games* 4:213–228. 48, 112, 130, 132, 158

[139] Liapis, A., Yannakakis, G. N., and Togelius, J. 2012b. Limitations of choice-based interactive evolution for game level design. *In* Proceedings of AIIDE Workshop on Human Computation in Digital Entertainment. 20

[140] Liapis, A., Yannakakis, G. N., and Togelius, J. 2013b. Designer modeling for personalized game content creation tools. *In* Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics. 20, 69

[141] Liapis, A., Yannakakis, G. N., and Togelius, J. 2013c. Generating map sketches for strategy games. *In* Proceedings of Applications of Evolutionary Computation, volume 7835, LNCS, pp. 264–273. Springer. 18

[142] Liapis, A., Yannakakis, G. N., and Togelius, J. 2013d. Sentient sketchbook: Computer-aided game level authoring. *In* Proceedings of the 8th Conference on the Foundations of Digital Games, pp. 213–220. 70

[143] Liapis, A., Yannakakis, G. N., and Togelius, J. 2013e. Sentient world: Human-based procedural cartography. *In* Proceedings of Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMusArt), volume 7834, LNCS, pp. 180–191. Springer. 21

[144] Licklider, J. C. R. 1960. Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics* HFE-1:4–11. 12

[145] Likert, R. 1932. A technique for the measurement of attitudes, pp. 1–55. *In* Archives of Psychology 14. 28

[146] Lim, C.-U. and Harrell, F. D. 2013. Modeling player preferences in avatar customization using social network data: A case-study using virtual items in team fortress 2. *In* Proceedings of IEEE conference on Computational Intelligence and Games. 16

[147] Lopes, P., Liapis, A., and Yannakakis, G. N. 2014. The C^{2}Create authoring tool: Fostering creativity via game asset creation. *In* Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG). 164

[148] Lubart, T. 2005. How can computers be partners in the creative process: classification and commentary on the special issue. *International Journal of Human-Computer Studies* 63:365–369. 3, 12

[149] Lynch, K. 1960. The Image of the City. MIT Press. 11

[150] Machado, P., Romero, J., Cardoso, A., and Santos, A. 2005. Partially interactive evolutionary artists. *New Generation Computing* 23:143–155. 164

[151] Machado, P., Romero, J., Santos, A., Cardoso, A., and Pazos, A. 2007. On the development of evolutionary artificial artists. *Computers & Graphics* 31:818–826. 23

[152] Maher, M. L., Fisher, D., and Brady, K. 2013. Computational models of surprise as a mechanism for evaluating creative design. *In* Proceedings of the International Conference on Computational Creativity. 22

[153] Mahlmann, T., Drachen, A., Togelius, J., Canossa, A., and Yannakakis, G. N. 2010. Predicting player behavior in tomb raider: Underworld. *In* Proceedings of IEEE conference on Computational Intelligence and Games, pp. 178–185. 16, 19

[154] Manurung, R., Ritchie, G., Pain, H., Waller, A., O'Mara, D., and Black, R. 2008. The construction of a pun generator for language skills development. *Applied Artificial Intelligence* 22:841–869. 23

[155] Martínez, H. P., Garbarino, M., and Yannakakis, G. N. 2011. Generic physiological features as predictors of player experience. *In* Proceedings of the 4th International Conference on Affective Computing and Intelligent Interaction, pp. 267–276. 22

[156] Mateas, M. and Sengers, P. 1999. Narrative intelligence. *In* Proceedings of AAAI Fall Symposium on Narrative Intelligence. 15

[157] Mateas, M. and Stern, A. 2002. A behavior language for story-based believable agents. *IEEE Intelligent Systems* 17:39–47. 13

[158] Mateas, M. and Stern, A. 2005. Procedural authorship: A case-study of the interactive drama façade. *In* Digital Arts and Culture. 10, 15

[159] Mawhorter, P. and Mateas, M. Procedural level generation using occupancy-regulated extension. 17

[160] McCormack, J. 1993. Interactive evolution of L-system grammars for computer graphics modelling, pp. 118–130. *In* Complex Systems: From Biology to Computation. ISO Press. 37

[161] McCoy, J., Treanor, M., Samuel, B., Reed, A. A., Wardrip-Fruin, N., and Mateas, M. 2013. Prom week. *In* Proceedings of the International Conference on the Foundations of Digital Games. 15

[162] Mchlis, J. The Enjoyment of Music: An Introduction to Perceptive Listening. W. W. Norton Limited. 22

[163] Meehan, J. R. 1976. The Metanovel: Writing Stories by Computer. PhD thesis. 15

[164] Michalewicz, Z. 1995. Do not kill unfeasible individuals. *In* Proceedings of the Fourth Intelligent Information Systems Workshop, pp. 110–123. 34, 36, 45

[165] Michalewicz, Z. 1996. Genetic algorithms + data structures = evolution programs. Springer-Verlag. 39

[166] Michalewicz, Z., Dasgupta, D., Le Riche, R., and Schoenauer, M. 1996. Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering Journal* 30:851–870. 33

[167] Mihalcea, R., Corley, C., and Strapparava, C. 2006. Corpus-based and knowledge-based measures of text semantic similarity. *In* AAAI, volume 6, pp. 775–780. 156

[168] Mitchell, T. 1997. Machine Learning. McGraw Hill. 27, 28

[169] Moraglio, A. 2011. Abstract convex evolutionary search. *In* Proceedings of the 11th workshop on Foundations of genetic algorithms, pp. 151–162. ACM. 101

[170] Morse, G., Risi, S., Snyder, C. R., and Stanley, K. O. 2013. Single-unit pattern generators for quadruped locomotion. *In* Proceeding of the Genetic and Evolutionary Computation Conference, pp. 719–726. ACM. 157

[171] Nareyek, A. 2007. Game AI is Dead. Long Live Game AI! *IEEE Intelligent Systems* 22:9–11. 14

[172] Nelson, M. J. and Mateas, M. 2005. Search-based drama management in the interactive fiction Anchorhead. *In* Proceedings of the First Annual Conference on Artificial Intelligence and Interactive Digital Entertainment. 23

[173] Nelson, M. J. and Smith, A. 2015. ASP with applications to mazes and levels. *In* N. Shaker, J. Togelius, and M. J. Nelson (eds.), Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer. 17

[174] Nojima, Y., Kojima, F., and Kubota, N. 2003. Trajectory generation for human-friendly behavior of partner robot using fuzzy evaluating interactive genetic algorithm. *In* Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation. 37

[175] Norton, D., Darrell, H., and Ventura, D. 2010. Establishing appreciation in a creative system. *In* Proceedings of the International Conference Computational Creativity, pp. 26–35. 23

[176] Novick, D. and Sutton, S. 1997. What is mixed-initiative interaction? *In* Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction. 2, 12

[177] Olsen, J. 2004. Realtime procedural terrain generation: Realtime synthesis of eroded fractal terrain for use in computer games. Technical report. 131

[178] Ontañon, S. 2013. The combinatorial multi-armed bandit problem and its application to real-time strategy games. *In* Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 160

[179] Orkin, J. 2002. Applying goal oriented action planning in games, pp. 217–229. *In* AI Game Programming Wisdom 2. Charles River Media. 14

[180] Orkin, J. 2005. Agent architecture considerations for real-time planning in games. *In* Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 105–110. 14

[181] Orkin, J. and Roy, D. 2007. The restaurant game: Learning social behavior and language from thousands of players online. *Journal of Game Development* 3:39–60. 15

[182] Pachet, F. and Roy, P. 2014. Non-conformant harmonization: the real book in the style of take 6. *In* Proceedings of the International Conference on Computational Creativity. 23

[183] Pearl, J. 1988. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann. 27

[184] Pedersen, C., Togelius, J., and Yannakakis, G. N. 2009. Modeling player experience in super mario bros. *In* Proceedings of the international conference on Computational Intelligence and Games, pp. 132–139. 18

[185] Perez, D., Togelius, J., Samothrakis, S., Rolhfshagen, P., and Lucas, S. M. 2013. Automated map generation for the physical traveling salesman problem. *IEEE Transactions on Evolutionary Computation* . 18

[186] Pérez, R. P. and Otoniel, O. 2013. A model for evaluating interestingness in a computer-generated plot. *In* Proceedings of the International Conference on Computational Creativity. 23

[187] Picard, R. W. 1997. Affective computing. MIT Press. 22

[188] Poli, R., Langdon, W. B., and McPhee, N. F. 2008. A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. 40

[189] Pomerleau, D. 1993. Knowledge-based training of artificial neural networks for autonomous robot driving. *In* Robot Learning. 27

[190] Preuss, M., Liapis, A., and Togelius, J. 2014. Searching for good and diverse game levels. *In* Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG). 153

[191] Propp, V. 1968. Morphology of the Folktale: Second Edition. American Folklore Society Bibliographical and Special Series. University of Texas Press. 22

[192] Raichel, D. 2000. The Science and Applications of Acoustics. Aip Series in Modern Acoustics and Signal Processing. Springer-Verlag GmbH. 22

[193] Ramachandran, V. S. and Hirstein, W. 1999. The science of art: a neurological theory of aesthetic experience. *Journal of consciousness Studies* 6:15–51. 23, 72, 132, 133

[194] Reintjes, J. F. 1991. Numerical control: making a new technology. Oxford University Press, Inc. 12

[195] Risi, S., Lehman, J., D'Ambrosio, D., Hall, R., and Stanley, K. O. 2012. Combining search-based procedural content generation and social gaming in the petalz video game. *In* Proceedings of Artificial Intelligence and Interactive Digital Entertainment Conference. 18

[196] Risi, S. and Stanley, K. O. 2013. Confronting the challenge of learning a flexible neural controller for a diversity of morphologies. *In* Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference, pp. 255–262. ACM. 157

[197] Ritchie, G. 2007. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines* 17:76–99. 22

[198] Romero, J., Machado, P., Carballal, A., and Osorio, O. 2011. Aesthetic classification and sorting based on image compression. *In* Applications of Evolutionary Computation, volume 6625 of *Lecture Notes in Computer Science*, pp. 394–403. Springer. 23

[199] Romero, J., Machado, P., Carballal, A., and Santos, A. 2012. Using complexity estimates in aesthetic image classification. *Journal of Mathematics and the Arts* 6:125–136. 23

[200] Rumelhart, D. 1995. Backpropagation: theory, architectures, and applications. Lawrence Erlbaum. 124, 143

[201] Sacks, O. 2008. Musicophilia. Knopf Doubleday Publishing Group. 22

[202] Salen, K. and Zimmerman, E. 2004. Rules of Play: Game Design Fundamentals. MIT Press. 9

[203] Scaltsas, T. and Alexopoulos, C. 2013. Creating creativity through emotive thinking. *In* Proceedings of the World Congress of Philosophy. 23, 25

[204] Schell, J. 2008. The Art of Game Design: A book of lenses. Taylor & Francis. 9, 10

[205] Schmitz, M. 2004. genoTyp, an experiment about genetic typography. *In* Proceedings of Generative Art. 37

[206] Schoenauer, M. 1996. Shape representations and evolution schemes. *In* Proceedings of the 5th Annual Conference on Evolutionary Programming, pp. 121–129. 39

[207] Schoenauer, M. and Michalewicz, Z. 1996. Evolutionary computation at the edge of feasibility. *In* Proceedings of the 4th Parallel Problem Solving from Nature, pp. 245–254. 34, 35

[208] Secretan, J., Beato, N., D'Ambrosio, D. B., Rodriguez, A., Campbell, A., Folsom-Kovarik, J. T., and Stanley, K. O. 2011. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation* 19:373–403. 23

[209] Secretan, J., Beato, N., D'Ambrosio, D. B., Rodriguez, A., Campbell, A., and Stanley, K. O. 2008. Picbreeder: evolving pictures collaboratively online. *In* Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, pp. 1759–1768. 37, 38, 40, 165

[210] Shaker, M., Shaker, N., and Togelius, J. 2013a. Evolving playable content for cut the rope through a simulation-based approach. *In* Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 13

[211] Shaker, M., Shaker, N., and Togelius, J. 2013b. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. *In* Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). AAAI Press. 13

[212] Shaker, N., Asteriadis, S., Yannakakis, G. N., and Karpouzis, K. 2013c. Fusing visual and behavioral cues for modeling user experience in games. *IEEE Transactions on System Man and Cybernetics, Special Issue on Modern Control for Computer Games* 43:1519–1531. 19

[213] Shi, Y. and Crawfis, R. 2013. Optimal cover placement against static enemy positions. *In* Proceedings of the 8th Conference on the Foundations of Digital Games. 155

[214] Sims, K. 1991. Artificial evolution for computer graphics. *In* Proceedings of the 18th annual conference on Computer graphics and interactive techniques, SIGGRAPH '91, pp. 319–328, New York, NY, USA. ACM. 23, 37

[215] Sims, K. 1992. Interactive evolution of dynamical systems. *In* Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, pp. 171–178. 37

[216] Sloman, A. 2002. Diagrams in the Mind? Springer. 25

[217] Smelik, R. M., Tutenel, T., de Kraker, K. J., and Bidarra, R. 2010. Interactive creation of virtual worlds using procedural sketching. *In* Proceedings of eurographics. 13

[218] Smith, A. M. 2012a. Mechanizing Exploratory Game Design. PhD thesis. 3

[219] Smith, A. M., Butler, E., and Popović, Z. 2013. Quantifying over play: Constraining undesirable solutions in puzzle design. *In* Proceedings of ACM Conference on Foundations of Digital Games. 17

[220] Smith, A. M. and Mateas, M. 2010. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. *In* Proceedings of IEEE Conference on Computational Intelligence and Games. 17, 54

[221] Smith, A. M. and Mateas, M. 2011. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games* 3:187–200. 17

[222] Smith, G. 2012b. Expressive Design Tools: Procedural Content Generation for Game Designers. PhD thesis. 3

[223] Smith, G., Whitehead, J., and Mateas, M. 2011. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games* . 13, 17, 54

[224] Smolensky, P. 1986. Information processing in dynamical systems: Foundations of harmony theory, pp. 194–281. *In* Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. 30

[225] Snodgrass, S. and Ontañon, S. 2013. Generating maps using markov chains. *In* Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics. 17

[226] Snowden, R., Snowden, R., Thompson, P., and Troscianko, T. 2012. Basic Vision: An Introduction to Visual Perception. OUP Oxford, 2 (revised edition) edition. 23

[227] Sorenson, N. and Pasquier, P. 2010. Towards a generic framework for automated video game level creation. *In* Proceedings of the International Conference on Evolutionary Computation in Games, volume 6024 of *Lecture Notes in Computer Science*, pp. 131–140. Springer. 35

[228] Sorenson, N., Pasquier, P., and DiPaola, S. 2011. A generic approach to challenge modeling for the procedural creation of video game levels. *IEEE Transactions on Computational Intelligence and AI in Games* 3:229–244. 155

[229] Spears, W. M. 1992. Crossover or mutation? *In* Foundations of Genetic Algorithms, Volume 2, pp. 221–237. Morgan Kaufmann. 32

[230] Stanley, K. O. 2006a. Exploiting regularity without development. *In* Proceedings of the AAAI Fall Symposium on Developmental Systems, Menlo Park, CA. AAAI Press. 38, 40, 140, 141, 165

[231] Stanley, K. O. 2006b. Exploiting regularity without development. *In* Proceedings of the AAAI Fall Symposium on Developmental Systems. AAAI Press. 130, 131

[232] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artifiicial Life* 15:185–212. 157

[233] Stanley, K. O. and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10:99–127. 38, 125, 130

[234] Sternberg, R. J. 1999. Handbook of creativity. Cambridge University Press. 23

[235] Sturtevant, N. and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. *In* Proceedings of the 20th National Conference on Artificial Intelligence, pp. 1392–1397. AAAI Press. 14

[236] Suits, B. 1978. The Grasshopper: Games, Life and Utopia. University of Toronto Press. 9

[237] Sutherland, I. E. 1963. Sketchpad: A man-machine graphical communication system. *In* Proceedings of the Spring Joint Computer Conference, AFIPS '63 (Spring), pp. 329–346. 12, 164

[238] Sutton, R. S. and Barto, A. G. 1998. Introduction to Reinforcement Learning. MIT Press. 27

[239] Swanson, R. and Gordon, A. S. 2012. Say anything: Using textual case-based reasoning to enable open-domain interactive storytelling. *ACM Transactions on Intelligent Interactive Systems* 2. 15

[240] Sweetser, P. and Wyeth, P. 2005. Gameflow: A model for evaluating player enjoyment in games. *ACM Computers in Entertainment* 3. 19

[241] Takagi, H. 2000. Active user intervention in an EC search. *In* Proceedings of the International Conference on Information Sciences., pp. 995–998. 39

[242] Takagi, H. 2001. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE* 89:1275–1296. Invited Paper. 37, 112, 135

[243] Tekofsky, S., Spronck, P., Plaat, A., van den Herik, H. J., and Broersen, J. 2013. Play style: Showing your age. *In* Proceedings of IEEE conference on Computational Intelligence and Games. 16

[244] Tetko, I. V., Livingstone, D. J., and Luik, A. I. 1995. Neural network studies, 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences* pp. 826–833. 28

[245] Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelback, J., and Yannakakis, G. 2010. Multiobjective exploration of the starcraft map space. *In* Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 265–272. 132

[246] Togelius, J., Yannakakis, G., Stanley, K., and Browne, C. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* . 16, 18

[247] Tokui, N. and Iba, H. 2000. Music composition with interactive evolutionary computation. *In* International Conference on Generative Art. 37

[248] Tremblay, J., Torres, P., Rikovitch, N., and Verbrugge, C. 2013. An exploration tool for predicting stealthy behaviour. *In* Proceedings of the AIIDE workshop on AI in the Game Design Process. 155

[249] Tychsen, A. and Canossa, A. 2008. Defining personas in games using metrics. *In* Proceedings of the 2008 Conference on Future Play: Research, Play, Share, pp. 73–80. 16

[250] van Kasteren, T. L. M., Englebienne, G., and Kröse, B. J. A. 2011. Hierarchical activity recognition using automatically clustered actions. *In* Proceedings of the Second international conference on Ambient Intelligence, pp. 82–91. Springer-Verlag. 22

[251] Veale, T. 2004. The challenge of creative information retrieval, pp. 457–467. *In* Computational Linguistics and Intelligent Text Processing, volume 2945 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 23

[252] Veale, T. 2011. Creative language retrieval: A robust hybrid of information retrieval and linguistic creativity. *In* Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1, pp. 278–287. 156

[253] Veale, T. and Hao, Y. 2007. Comprehending and generating apt metaphors: A web-driven, case-based approach to figurative language. pp. 1471–1476. 23

[254] Vile, A. and Polovina, S. 1998. Thinking of or thinking through diagrams? The case of conceptual graphs. *In* Thinking with Diagrams Conference, The University of Wales, Aberystwyth. 24

[255] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. 2008. Extracting and composing robust features with denoising autoencoders. *In* Proceedings of the 25th International Conference on Machine learning, pp. 1096–1103. 30, 31

[256] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machince Learning Research* 11:3371–3408. 30

[257] Walker, M. and Whittaker, S. 1990. Mixed initiative in dialogue: An investigation into discourse segmentation. *In* Proceedings of the 28th Annual Meeting on Association for Computational Linguistics, pp. 70–78. 12

[258] Wang, H.-C. 2008. Modeling idea generation sequences using Hidden Markov Models. *In* Proceedings of the 30th Annual Meeting of the Cognitive Science Society. 21

[259] Weber, B. G., Mateas, M., and Jhala, A. 2011. Building human-level ai for real-time strategy games. *In* AAAI Fall Symposium: Advances in Cognitive Systems. 14

[260] Weber, B. G., Mawhorter, P. A., Mateas, M., and Jhala, A. 2010. Reactive planning idioms for multi-scale game ai. *In* Proceedings of the IEEE Conference on Computational Intelligence and Games, pp. 115–122. 14

[261] Wetzel, B., Anderson, K., Koutstaal, W., and Gini, M. 2012. If Not Now, Where? Time and Space Equivalency in Strategy Games. *In* Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 158

[262] Whitley, D., Starkweather, T., and Bogart, C. 1990. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing* 14:347–361. 40

[263] Widrow, B. and Stearns, S. D. 1985. Adaptive signal processing. Prentice-Hall, Inc. 49

[264] Wiggins, G. A. 2006. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems* 19:449–458. 22

[265] Wiggins, G. A., Papadopoulos, G., Phon-Amnuaisuk, S., and Tuson, A. 1998. Evolutionary methods for musical composition. *In* Proceedings of the International Computer Music Conference. 23

[266] Wittgenstein, L. 2010. Philosophical investigations. John Wiley & Sons. 23

[267] Yannakakis, G. and Hallam, J. 2011. Rating vs. preference: A comparative study of self-reporting, pp. 437–446. *In* Affective Computing and Intelligent Interaction, volume 6974, LNCS. Springer. 28, 49

[268] Yannakakis, G. N. 2012. Game AI revisited. *In* Proceedings of ACM Computing Frontiers Conference. 14

[269] Yannakakis, G. N., Liapis, A., and Alexopoulos, C. 2014. Mixed-initiative co-creativity. *In* Proceedings of the 9th Conference on the Foundations of Digital Games. 4, 155

[270] Yannakakis, G. N., Spronck, P., Loiacono, D., and André, E. 2012. Player modeling, pp. 45–59. *In* Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). 18

[271] Yannakakis, G. N. and Togelius, J. 2011. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing* 99. 18

[272] Zook, A. and Riedl, M. O. 2014. Generating and adapting game mechanics. *In* Proceedings of FDG Workshop on Procedural Content Generation. 162

Appendix A

Final Maps of Sentient Sketchbook users

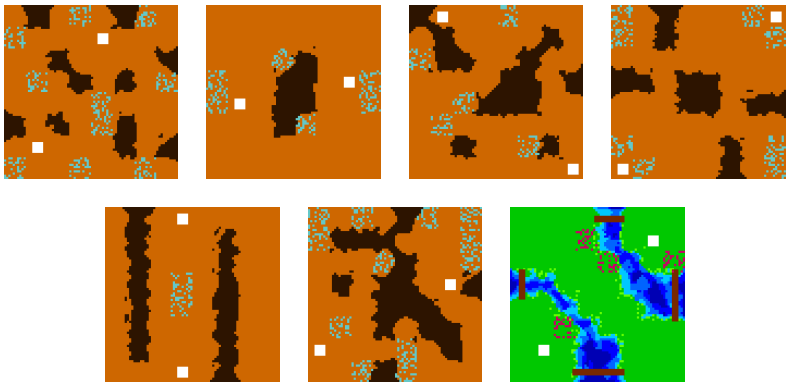


Fig. A.1: Small maps from Sentient Sketchbook users.

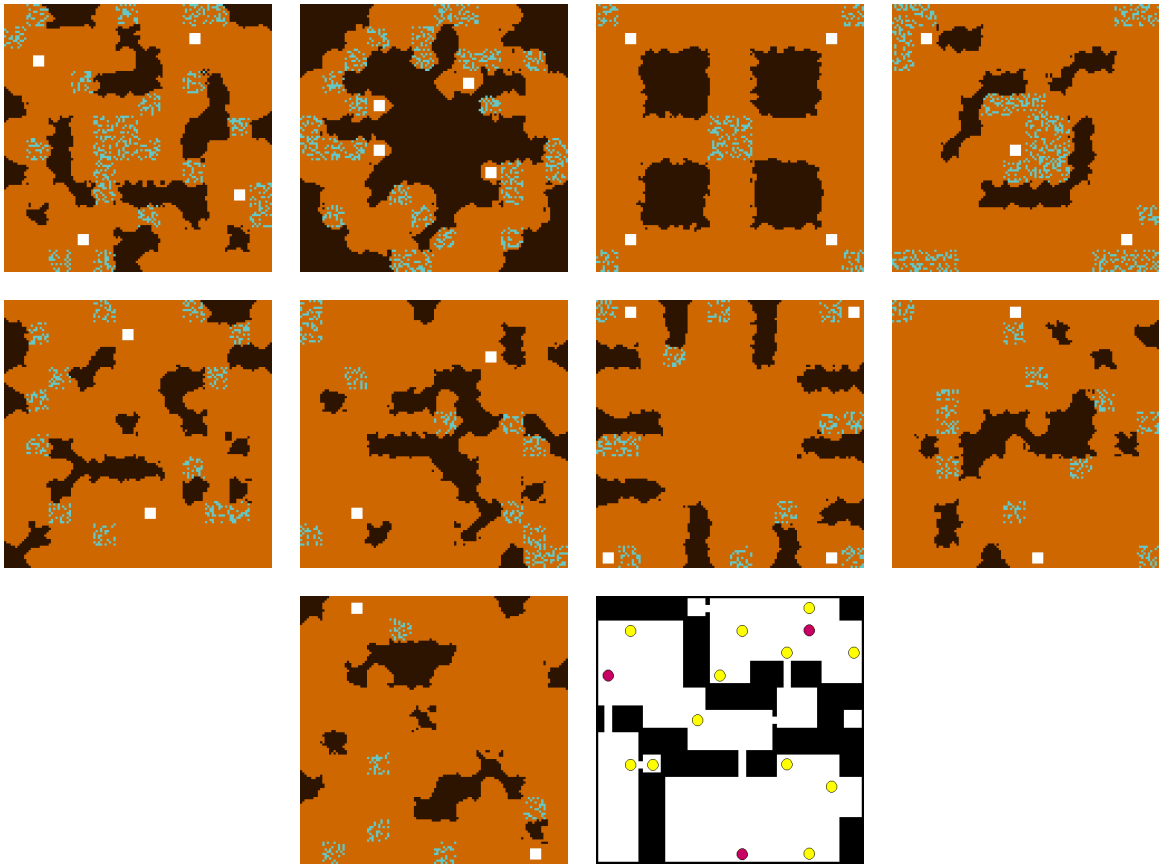


Fig. A.2: Medium maps from Sentient Sketchbook users.

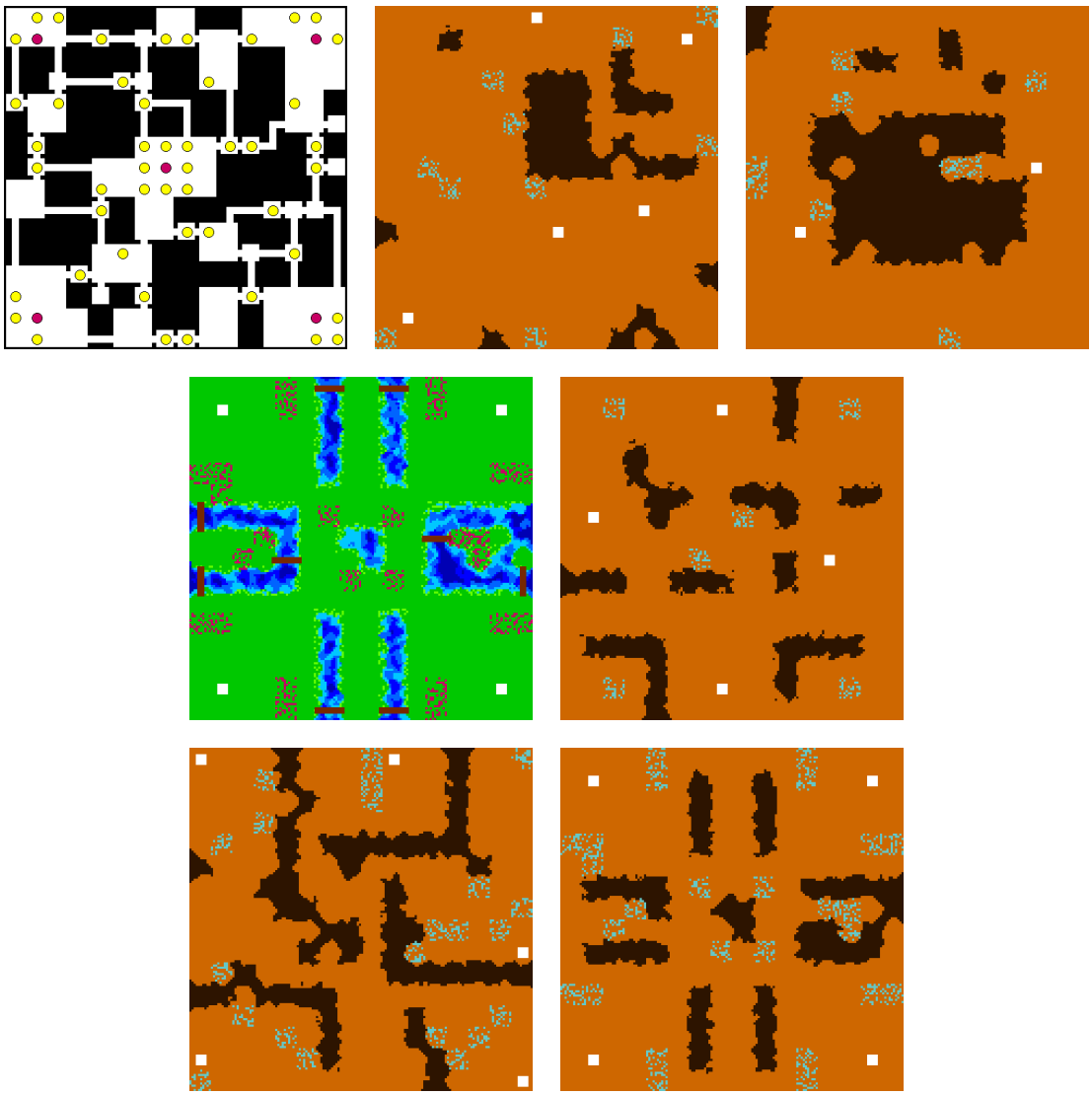


Fig. A.3: Large maps from Sentient Sketchbook users.

Appendix B

Spaceships of DeLeNoX

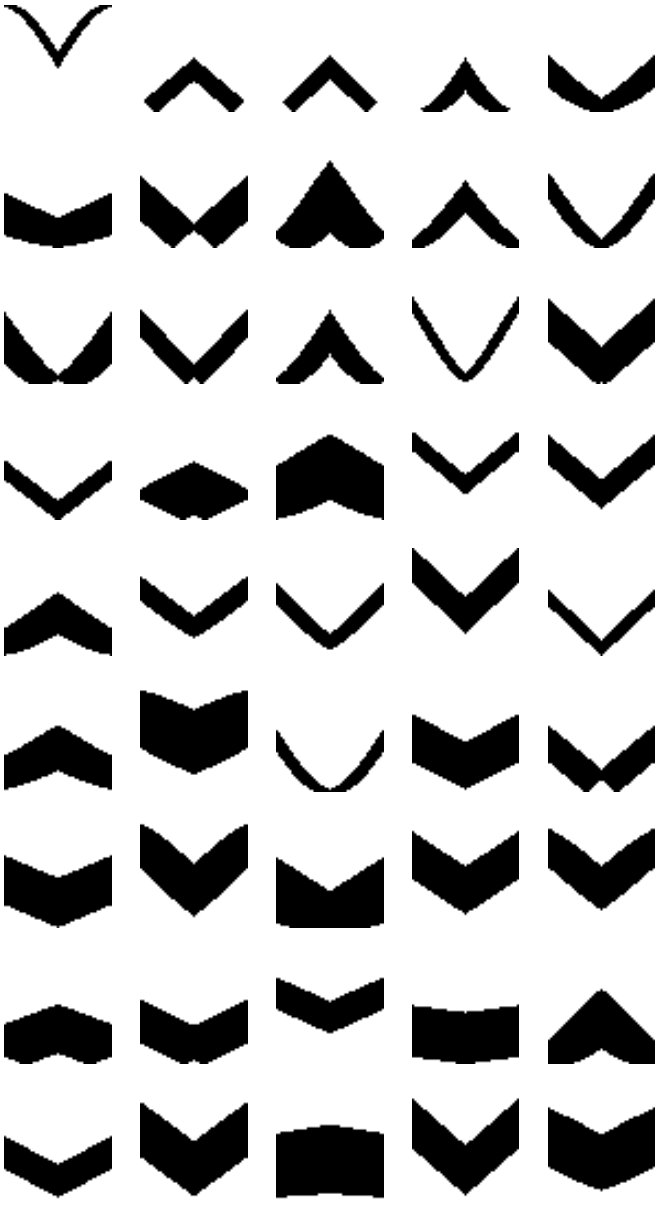


Fig. B.1: A sample of the 1000 initial spaceships with the simplest CPPN topology, sorted by difference according to the features found on this training set.

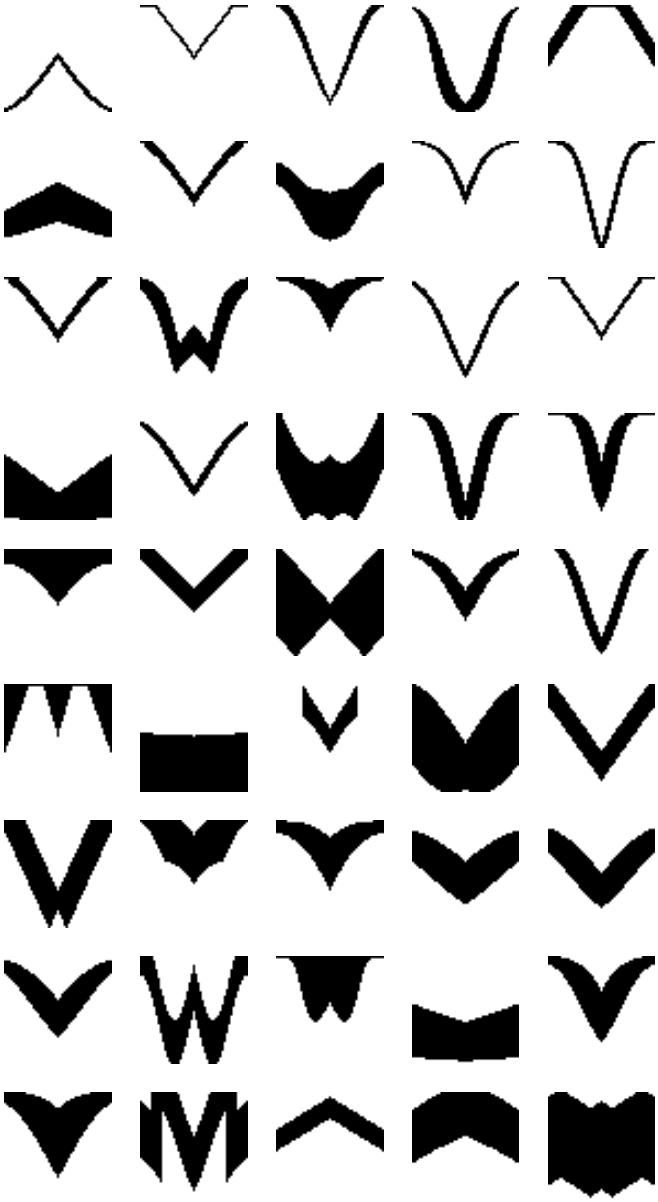


Fig. B.2: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.1 (first iteration of DeLeNoX), sorted by difference according to the features found on this training set.

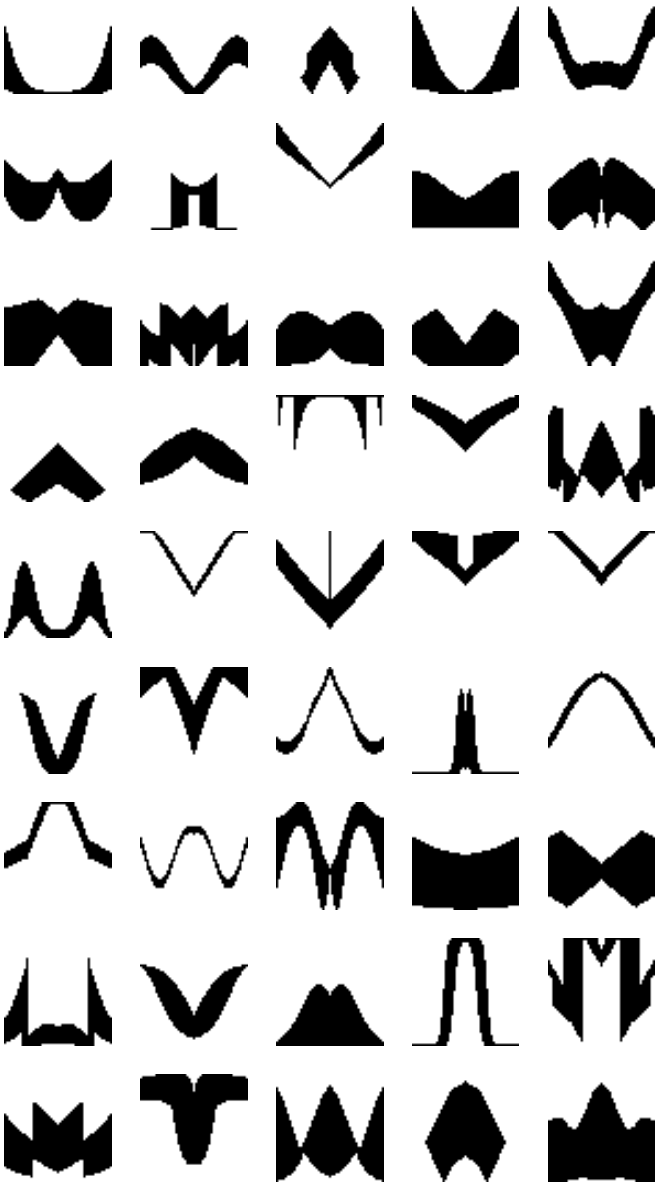


Fig. B.3: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.2 (second iteration of DeLeNoX), sorted by difference according to the features found on this training set.

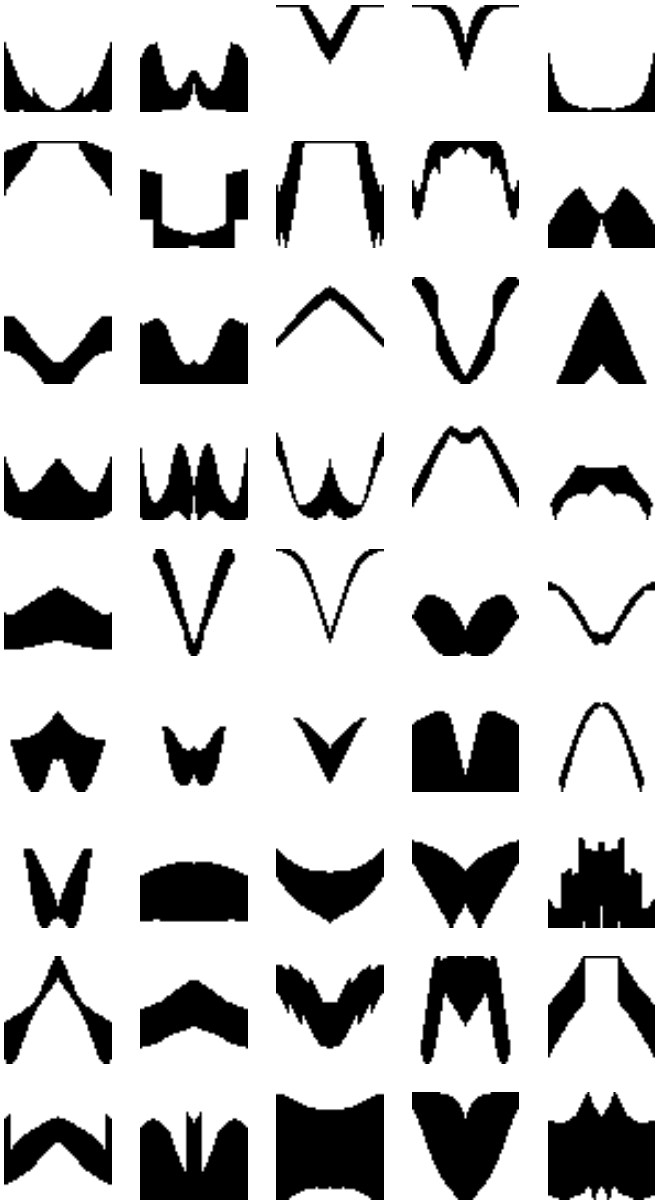


Fig. B.4: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.3 (third iteration of DeLeNoX), sorted by difference according to the features found on this training set.

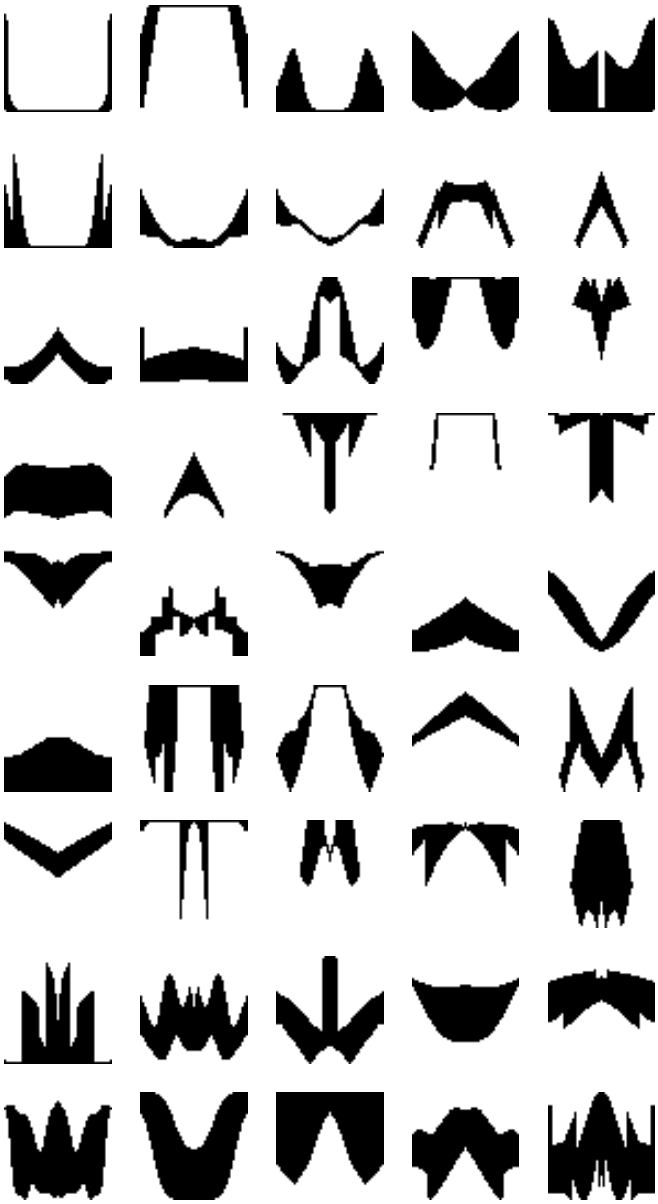


Fig. B.5: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.4 (fourth iteration of DeLeNoX), sorted by difference according to the features found on this training set.

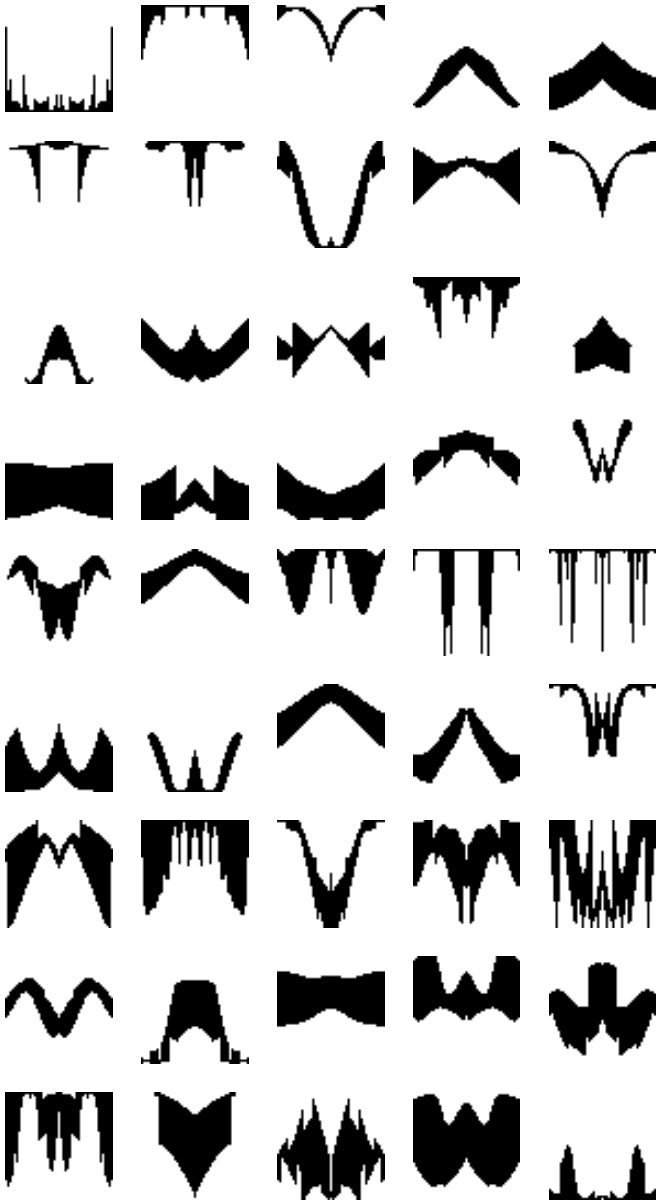


Fig. B.6: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.5 (fifth iteration of DeLeNoX), sorted by difference according to the features found on this training set.

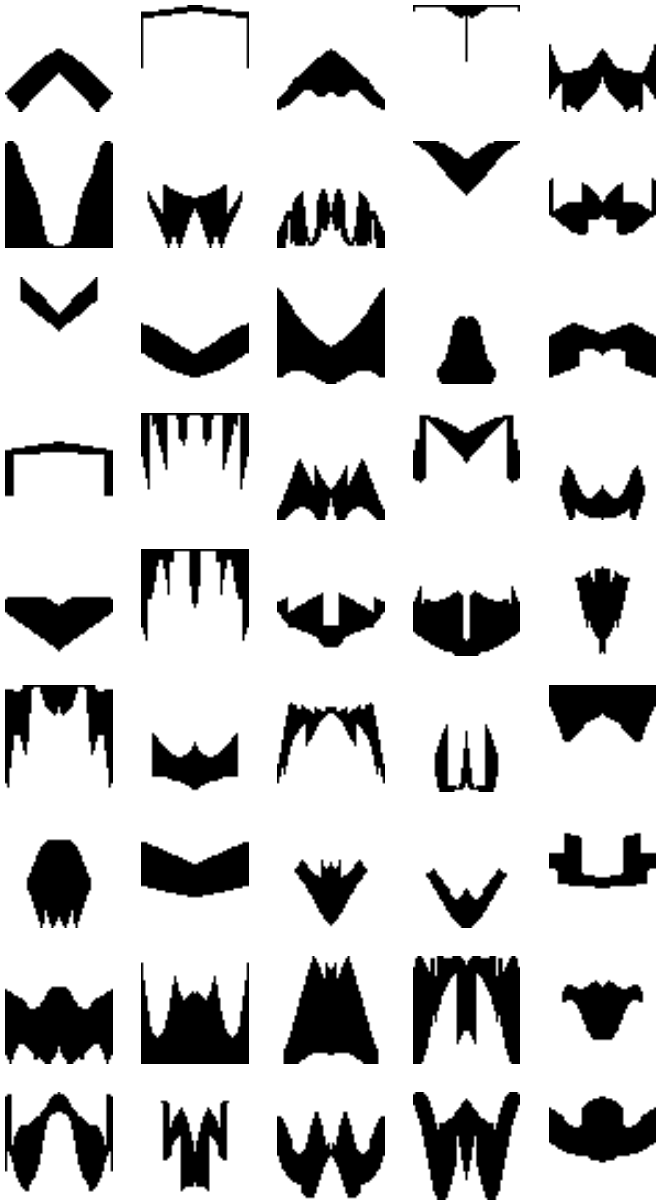


Fig. B.7: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.6 (sixth iteration of DeLeNoX), sorted by difference according to the features found on this training set.

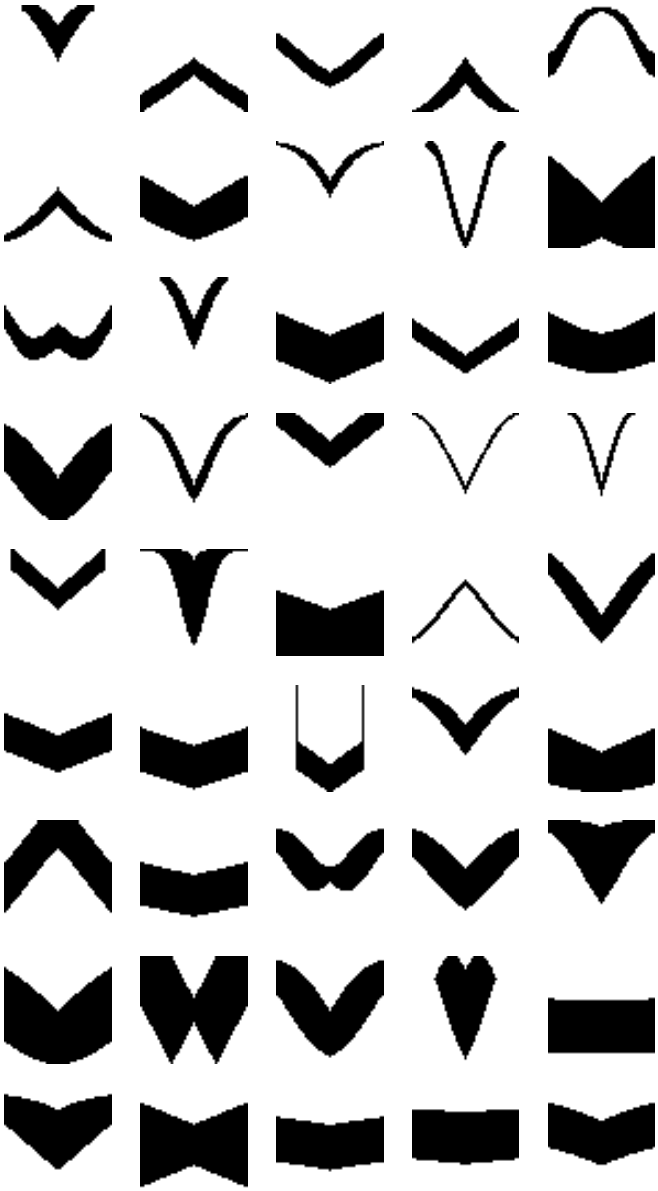


Fig. B.8: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.1 (first iteration of static runs), sorted by difference according to the features found on this training set.

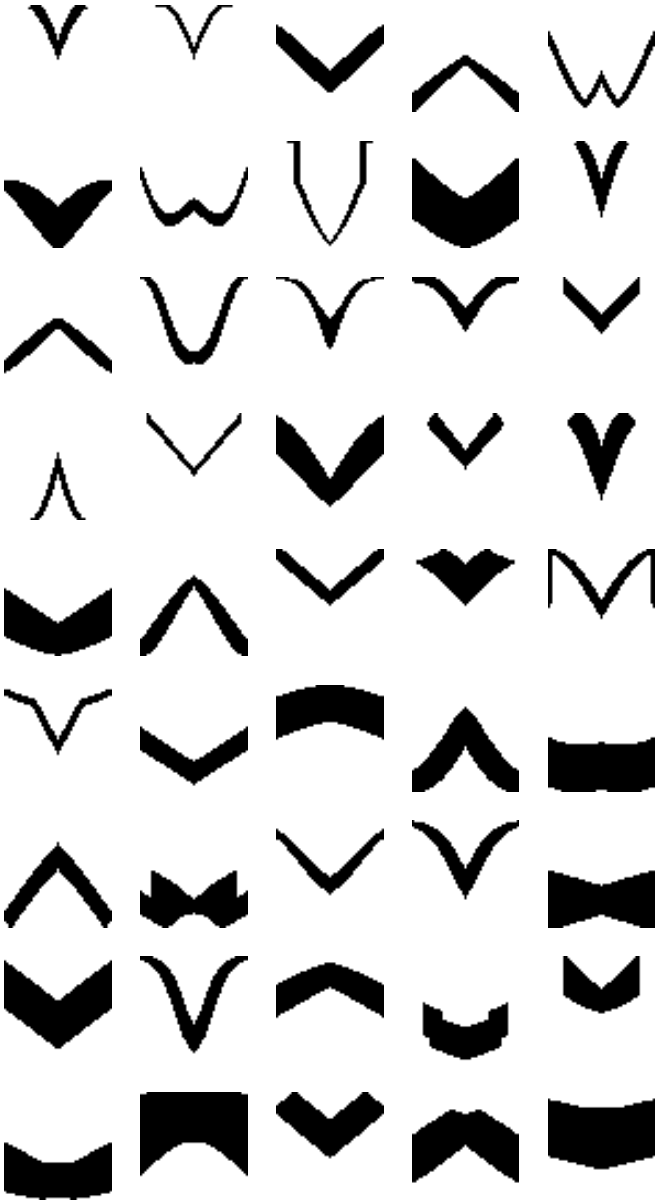


Fig. B.9: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.1 (second iteration of static runs), sorted by difference according to the features found on this training set.

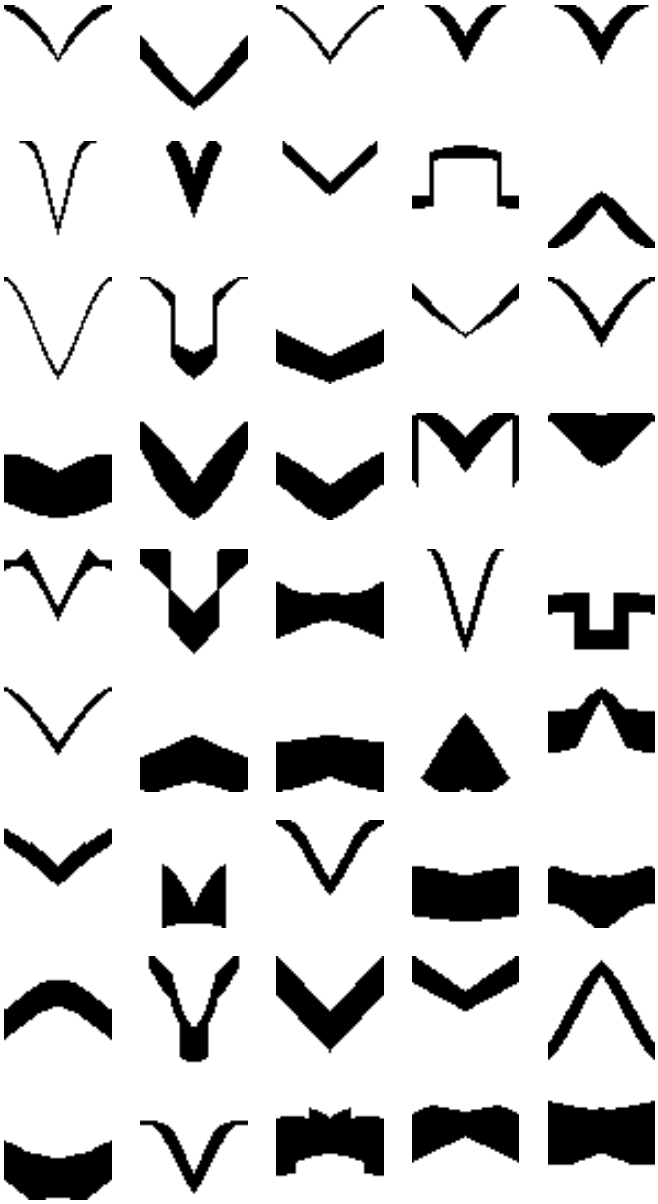


Fig. B.10: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.1 (third iteration of static runs), sorted by difference according to the features found on this training set.

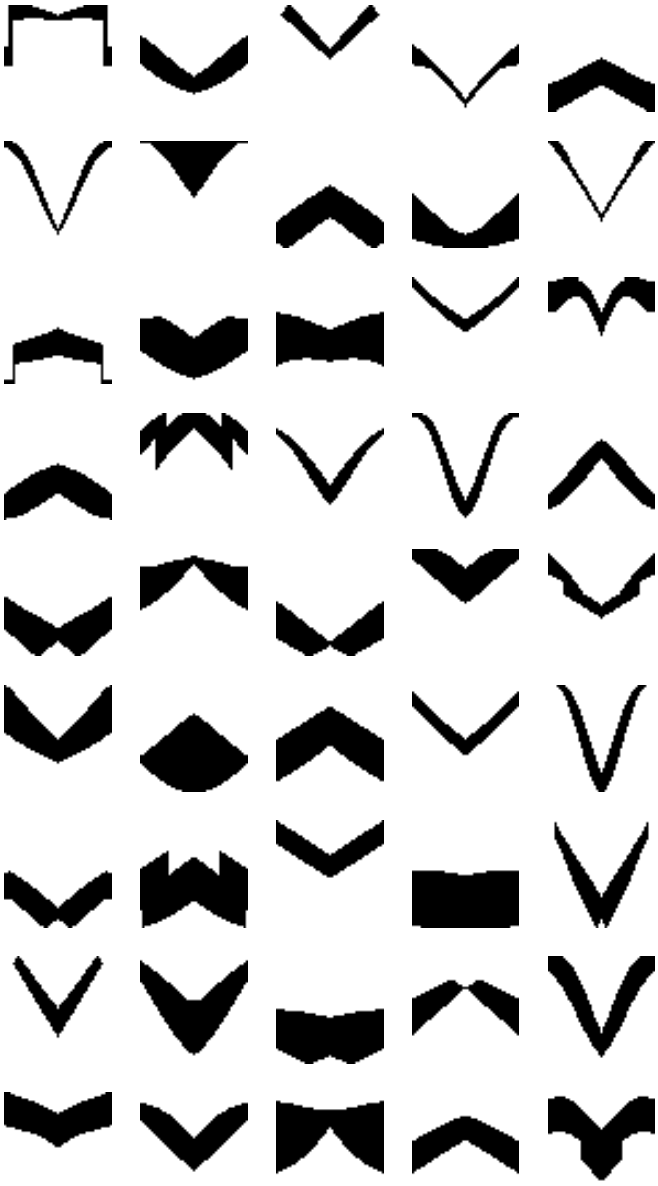


Fig. B.11: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.1 (fourth iteration of static runs), sorted by difference according to the features found on this training set.

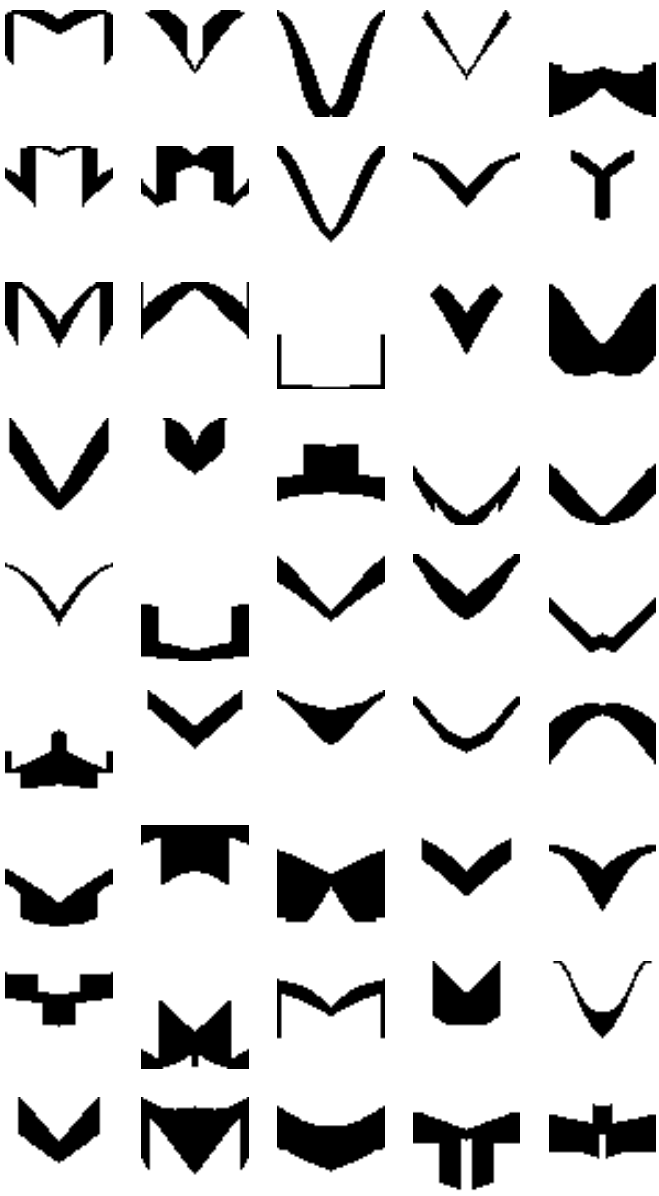


Fig. B.12: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.1 (fifth iteration of static runs), sorted by difference according to the features found on this training set.

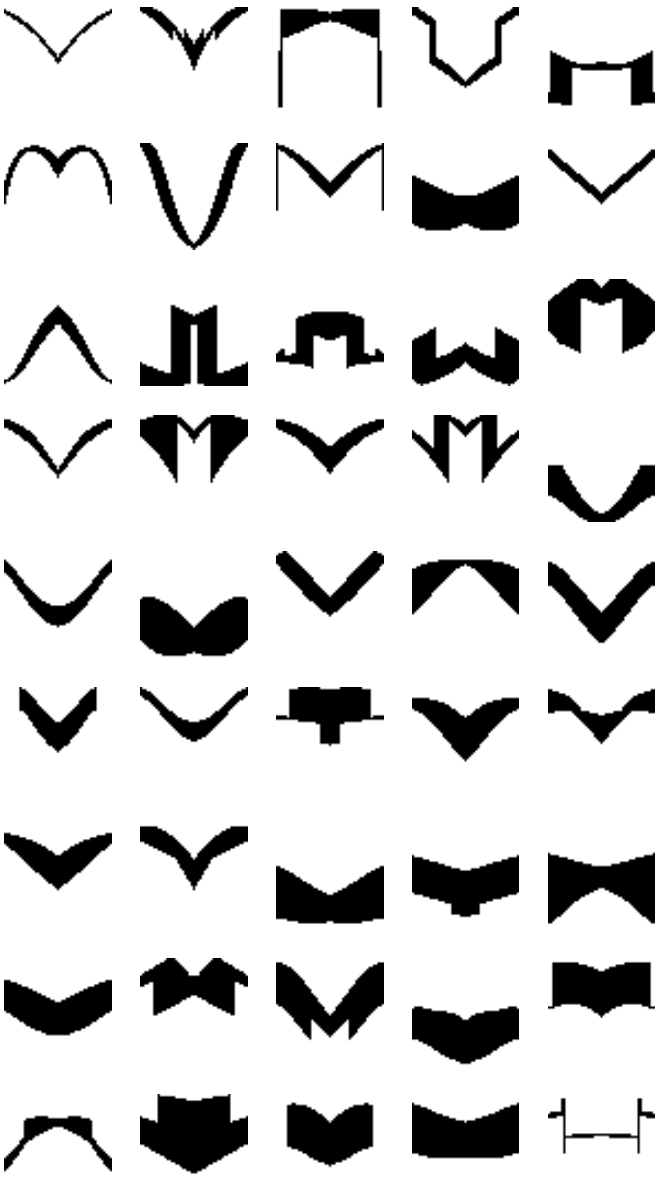


Fig. B.13: A sample of the 1000 spaceships evolved according to the features of the training set of Fig. B.1 (sixth iteration of static runs), sorted by difference according to the features found on this training set.