# Using a Surrogate Model of Gameplay for Automated Level Design

Daniel Karavolos
*Institute of Digital Games*
*University of Malta*
Msida, Malta
daniel.karavolos@um.edu.mt

Antonios Liapis
*Institute of Digital Games*
*University of Malta*
Msida, Malta
antonios.liapis@um.edu.mt

Georgios N. Yannakakis
*Institute of Digital Games*
*University of Malta*
Msida, Malta
georgios.yannakakis@um.edu.mt

***Abstract*—This paper describes how a surrogate model of the interrelations between different types of content in the same game can be used for level generation. Specifically, the model associates level structure and game rules with gameplay outcomes in a shooter game. We use a deep learning approach to train a model on simulated playthroughs of two-player deathmatch games, in diverse levels and with different character classes per player. Findings in this paper show that the model can predict the duration and winner of the match given a top-down map of the level and the parameters of the two players' character classes. With this surrogate model in place, we investigate which level structures would result in a balanced match of short, medium or long duration for a given set of character classes. Using evolutionary computation, we are able to discover levels which improve the balance between different classes. This opens up potential applications for a designer tool which can adapt a human authored map to fit the designer's desired gameplay outcomes, taking account of the game's rules.**

***Index Terms*—deep learning, surrogate model, artificial evolution, procedural content generation, computational creativity**

## I. Introduction

Despite a long history of procedural content generation (PCG) in the game industry, there is still a gap between the algorithms developed in the academic community and those applied in the commercial sector. In an effort to address one of the major concerns of the industry, there is an increasing research interest in giving designers more control over the generated content. There are several ways to ensure that a designer can influence the creative process, e.g. by creating a set of modular components that can be recombined via a generative grammar [1] or evolution [2], by placing constraints which can be solved via answer-set programming [3], or by adjusting the objectives of a search-based generator [4].

While there has also been work towards making the designer a part of the generative loop itself, e.g. via interactive evolution or through a mixed-initiative process [5], this paper will focus on tools that allow the designer to customize the generative space a priori. This can be done with visualizations that show the effect of generative parameters on the expressive range of the generator [6], allowing designers to more efficiently tune the algorithm's parameters to the intended outcomes, or by allowing designers to literally set targets (or acceptable ranges) for their desired output. Both search-based and constrained programming approaches can use such a designer-specified goal to bias their generative output.

This paper proposes a tool for designers to create or adapt levels towards specific gameplay outcomes. The tool takes advantage of a model of gameplay that can predict emergent gameplay properties from the level structure as well as the game's ruleset. This model, trained via deep learning on a large corpus of simulated plays with artificial agents, acts as a *surrogate model* [7] and bypasses the need for computationally expensive simulations. As a design tool, the quick response times afforded by this model can be used both for immediate feedback while the designer adapts the level or the game rules (both of which affect the gameplay outcomes). Moreover, as explored in this paper, the surrogate model can be used to generate new levels or variations of existing levels through a search-based process which uses the surrogate model instead of computationally expensive simulation-based evaluations.

The use case in this paper is competitive first person shooter (FPS) gameplay, which has easily quantifiable gameplay outcomes, i.e. gameplay balance (whether a player wins easily over another) and match duration (whether the match is over quickly). The game rules and level structures that are being provided as input to the surrogate model are the game's competing character classes and the top-down view of the FPS map. *Character classes* are common in shooter games: each competing player chooses a class, which may have a different survivability and speed as well as a signature weapon which fits its role (e.g. scout, sniper class). The FPS map itself has multiple floors, to allow for FPS level patterns such as sniping positions [8]. Experiments in this paper focus on the designer-guided aspect of this framework, allowing designers to specify a map they would like to improve, the target character classes that this map is intended for, and the desired game balance and duration. The system then adapts the level via recombination and mutation, to minimize the distance with the designer's desired gameplay outcomes, using the surrogate model to predict the evolving maps' gameplay properties. A thorough evaluation with a broad set of character classes and target match durations shows that the generative approach can improve the designer's map to better match the gameplay outcomes and better take advantage of specific classes' strengths and weaknesses.

## II. Related Work

Evaluation of game content has been a topic of broad academic inquiry, e.g. via heuristics, constraints, or a combination of the two. Game levels have often been parsed in terms of their geometric or path properties to estimate a modicum of balance between two or more competing players [9]–[11]. These heuristics observe the level structure alone, and are expected to be accurate only if the two players have the same in-game characteristics (e.g. available units, tech tree, or weapon power) and the same playstyle or skill. A more objective way of assessing game balance is through agent-based simulations or actual player traces. For the latter, interactive evolution can be used indirectly to adapt a level based e.g. on the combat time of a player [12]. For the former, artificial agents have often been used in simulations to assess each and every individual in the evolving population; this approach is computationally expensive and forms the bottleneck in simulation-based evaluations for search-based PCG [4]. Such agents are often simplistic (including those used in this paper), but agents may use more general gameplaying algorithms such as Monte-Carlo Tree Search [13] or may have different goals depending on the play persona they are trying to emulate [14]. Our proposed framework attempts to alleviate the computational cost of extensive simulations by using a model trained on a broad variety of level structures and diverging in-game characteristics.

This paper explores how machine learning can be used for procedural content generation as a surrogate model, indirectly influencing the fitness function of a search-based PCG algorithm [4]. So far, machine learned models are primarily used to directly manipulate game content [15]. For instance, neural networks have mostly been used to learn level patterns which are then applied directly to the level. For example, a recurrent neural network that predicts sequences of tiles is used to create levels for *Super Mario Bros.* (Nintendo 1985) in [16]; a convolutional neural network (CNN) is used to place resources on a pre-made *Starcraft II* (Blizzard 2010) map [17]. Other work has used autoencoders to learn patterns in *Super Mario Bros.* levels and use the encoding-decoding sequence to repair broken segments [18]. Finally, CNNs have been used to predict various characteristics of *Super Mario Bros.* levels based on player annotations [19], but these networks were not used for content generation. While machine learning has a long history in procedural generation [15], the proposed framework uses its learned model indirectly (i.e. to guide evolution) rather than directly. More importantly, it is the first attempt at using a model that has learned to combine both game rules (in the form of class parameters) and level properties for actual generation. The model combines three different facets of games, i.e. game design, level design and gameplay as discussed in [20]. The framework is thus the first step towards game facet orchestration where all facets of games are considered as a whole rather than e.g. considering only the structural parts of the level [10] or the properties of weapons [21] in a vacuum.

## III. Game Framework

This paper uses first-person shooter (FPS) games as a use case for mapping level structure and game parameters with gameplay outcomes; it uses that mapping to generate levels appropriate for a one versus one deathmatch game between two players. The players are assumed to be on an equal skill level, and each of them controls one avatar that belongs to a specific *character class*. Character classes are common in shooter games such as *Team Fortress 2* (Valve 2007) and have different gameplay styles and strategies, as well as a different signature weapon. This paper uses the same character class names and attributes from *Team Fortress 2* (TF2) as a benchmark for the level generator capabilities.

Deathmatch games are competitive: the player who score more kills on their opponent(s) is the winner. A session in a deathmatch game finishes usually after a specific time has elapsed or a specific number of kills by one or both players. The framework in this paper considers matches to be complete when a total of 20 kills is scored; a time limit of 600 seconds is also in place, but results from matches that timed out are ignored. The game used for these experiments is implemented in *Unity 2017*, a commercial game engine, and is based on an existing toolkit[1]. The two competing players start the game in opposite corners of a game level (described below).

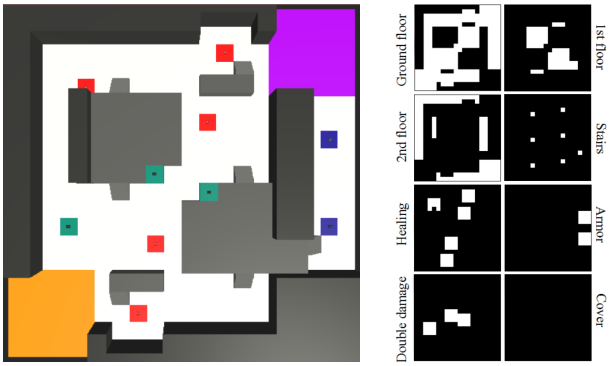
### A. Game Parameters

The character class of each avatar is represented by eight parameters. Two of these parameters are specific to the character, i.e. hit points and movement speed, while the other six are characteristics of their weapon: damage (per shot), accuracy (i.e. the size of the cone in which bullets are fired), rate of fire, clip size, the number of bullets per shot and weapon range. As noted earlier, the inspiration for the class parameters is from the TF2 game; experiments in this paper use parameter values from the game itself. The one addition to TF2 parameters was that of range, to discern when AI agents should shoot.

### B. Map properties

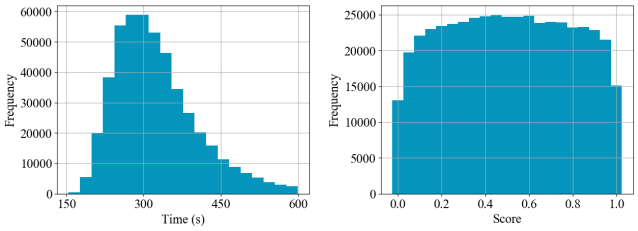
The maps in the game consist of a grid of $20 \times 20$ tiles, which can have three degrees of elevation. The ground floor and first floor are both traversable, while the second floor is inaccessible and acts as a barrier. Each tile only has one elevation (there are no tunnels or bridges). Players travel from the ground floor to the first floor via stairs; they can go down to the ground floor via stairs or drop off a ledge. An example map is shown in Figure 1a, where the floors are indicated in white, dark gray and black, and stairs between the ground and the first floor in light gray. The spawn point of the first player (P1) is always in the bottom left corner, while the second player (P2) always spawns in the top right corner. Furthermore, the maps can contain three types of pickups that are common in shooter games: a healthpack (increases health up to a maximum), armor (offers additional health which is depleted first) and a damage boost (player's bullets temporarily deal double damage).

[1] http://opsive.com/assets/DeathmatchAIKit/




(a) 3D shooter level (b) CNN inputs

Fig. 1: A view of the in-game 3D level and its transformation into CNN inputs. Orange and purple areas are the bases of player 1 and 2 respectively. Red tiles are healing locations, blue and turquoise tiles are armor and double damage powerups respectively.




(a) Game Duration (b) Kill Ratio

Fig. 2: Distribution of gameplay outcomes in the corpus.

## IV. A Surrogate Model of Gameplay

In order to learn a mapping between levels, game rules, and the gameplay outcomes, a deep learning approach was used. This Section describes the corpus used to learn patterns from, the machine learning approaches and their performance.

### A. Creating a Game Level Corpus

In order to provide a rich and expressive dataset to learn patterns from, gameplay outcomes were collected from simulations between artificial agents playing through a broad set of levels and character classes. Agents' behavior is controlled by behavior trees that were adapted from the original toolkit.

Agents played two matches in a unique combination of class pairs (one character class per player) and levels: each agent used the same character class twice, one starting from the bottom left corner of the level and the other starting from the top right corner of the level (orange and purple in Fig. 1). In total, $10^5$ levels and class pairs were generated, resulting in $2 \cdot 10^5$ data points regarding gameplay outcomes.

To generate a pair of character classes, 16 game parameters (8 per player) were normalized to a predetermined value range and a floating point value for each was randomly assigned; range was chosen randomly between short, medium and long.

To generate the game levels, a constructive map generator was implemented, using digging agents and generative grammars [22]. The overall layout of the map is created in a lower resolution representation ($4 \times 4$ grid of cells), crafting two paths between player bases: one on the upper side and one on the lower side of the diagonal. Agents then operate on each cell (which is $5 \times 5$ tiles in the final level) to connect the cell to its adjacent ones. The "walls" that remain from this process are transformed into first or second floor tiles randomly; cellular automata then add more first floor tiles. The algorithm places stairs on eligible tiles with a 20% probability. Each unreachable first floor tile is transformed into a second floor, in order to guarantee traversability of first floor tiles. After the level architecture is created, each cell has a 33% chance of having a pickup placed on a random tile within it.

Through the process described above, a dataset of $2 \cdot 10^5$ gameplay outcomes is created, out of which 6% are omitted as the matches are not completed within the time limit. For the remaining matches, the core gameplay outcomes that can be learned is the kill ratio (KR) of one player (P1) to the total kills, and the duration of the match in seconds. The distribution of these two gameplay outcomes is shown in Fig. 2. There is an almost uniform distribution between balanced matches (KR of 0.5) and matches where P1 had a clear advantage (KR near 1) or disadvantage (KR near 0). On the other hand, duration is skewed towards values around 300 seconds, with very few matches lasting below 200 seconds (2%) or over 500 seconds (4%). This may affect the accuracy of the deep learning approach, as will be discussed in Section IV-D.

### B. Data Input

Several steps are taken to process the data collected from simulated matches in order for the machine learning approach to read it. For gameplay outcomes (the intended outputs) the kill ratio and duration are normalized to the $[0, 1]$ range: kill ratio is already normalized while duration is min-max normalized. For character class parameters, all parameters are min-max normalized in order to generate character classes for the training set. For the game level, the grid of $20 \times 20$ tiles is transformed into multiple channels using a variant of one-hot encoding (see Fig. 1). In the channels that encode pickups, a tile with a pickup and all its adjacent tiles are 1, except second floor tiles. Pickups were given more prominence in their respective channel due to their scarcity in the level and their importance in gameplay.

### C. Convolutional Neural Network Architecture

Following a broad set of preliminary experiments with network architectures, activation functions and optimization strategies, the network chosen for this task is similar to [23]. This CNN has two separate information streams, one for the map and one for the pair of character classes. The level input is passed through two blocks of convolution and max-pooling, with 8 and 16 filters respectively. The convolutions are of size $3 \times 3$ (with zero-padding), and the end-result is a flat vector of 400 features for the level. The 16 parameters of the character classes are passed to a single fully-connected layer of 8 nodes,

the output of which is concatenated to the flat feature vector of the map. Finally, this combined feature vector is connected to a single fully connected layer of 32 nodes which connects to two outputs that predict the two gameplay outcomes (kill ratio and duration). All nodes use a ReLU activation function.

### D. Training Results

In order to validate the performance of our CNN architecture, baselines with several multi-layer perceptron models (MLP) were tested, as well as a perceptron and linear regression (LR). For the sake of brevity, the performance of the best MLP (with 128 neurons) is reported. Additionally, to assess the importance of each input modality, the same models are trained using only the character class parameters or only the map as input by leaving the remaining inputs at a value of 0. This paper focuses on a supervised regression task; the two accepted performance criteria for such tasks are (a) the model's prediction error and (b) how much of the variance in the data is explained by the model. The former is computed by the mean absolute error, $MAE_t$ and $MAE_{KR}$ for duration and score respectively. The latter is computed by the $R^2$ metric (with typical ranges of $[0, 1]$) for these dimensions ($R^2_t$ and $R^2_{KR}$). All models in this section were trained for 30 epochs, while early stopping was used to prevent over-fitting. The results reported are based on 10-fold cross-validation.

In general all baseline models perform very similarly, with a maximum difference in error of 0.01 for both dimensions. All baseline models can fairly accurately predict the kill ratio of P1 ($MAE_{KR} = 0.09$) and, surprisingly, the CNN is only slightly better ($MAE_{KR} = 0.07$). Similarly, the explained variance is fairly high with $R^2_{KR}$ values ranging from a minimum of $0.83$ for LR to $0.91$ for the CNN. This high accuracy even with simple models can be explained by the significant Pearsson correlations between score and 6 class parameters (out of 16).

In contrast, all models seem to struggle to predict game duration. Although the error is quite similar to score prediction ($0.09 \leq MAE_t \leq 0.10$ for all models), the explained variance is much lower. The perceptron and LR perform the worst predictions ($R^2_t = 0.48$), while both the MLP ($R^2_t = 0.55$) and CNN ($R^2_t = 0.57$) are somewhat better. The low mean squared error but low $R^2_t$ values is likely due to the skewed distribution of duration in the training set (see Fig. 2b).

All in all, the CNN model can predict both kill ratio and duration more accurately than baselines, although even simple models such as the perceptron perform surprisingly well.

## V. Generating Levels for Specific Game Outcomes

Our main goal is to explore how the trained surrogate model of Section IV can be used to generate a map with a specific game duration and kill distribution for a matchup between two character classes. To achieve this, a genetic algorithm is used to generate levels targeting a fitness function based on the outcomes of the surrogate model. The surrogate model is exploited by using the evolving map (transformed as per Section IV-B into readable input) and the character classes (which are specified a priori and do not change), and evaluating whether the gameplay output provided by the CNN matches some designer-specified goals. The fitness for the level is calculated based on the Euclidean distance between the vector of kill ratio and duration provided by the CNN and the designer-specified vector of intended kill ratio and duration. Evolution attempts to minimize this fitness score.

Regarding the specifics of the genetic algorithm itself, the level representation is based on tiles which contain all relevant information (rather than individual channels used by the CNN). Each tile is represented as a tuple of integers describing the elevation (0 for ground floor, 1 and 2 for first and second floor) and contents (e.g. stairs, healthpack, etc.). The players' bases are identical to the original training set (P1 has $5 \times 5$ tiles at the bottom left corner, similarly for P2 at the top right corner) and evolution can not change those areas.

A new population is created by first selecting individuals to reproduce. The fittest 10% of the population is copied to the new population unchanged (elitism). The remaining 90% is chosen via tournament selection of size 5, and then recombined and mutated. For the purposes of mutation and recombination, the level (of $20 \times 20$ tiles) is divided into a $4 \times 4$ grid of cells. Each pair of individuals has a 20% chance of producing offspring via recombination: recombination is implemented by randomly picking a cell at that position from either parent. Offspring and any unselected parents are then susceptible to mutation. Every cell of every individual has a 10% chance of being mutated by one of the following variants:

- *Move Pickup:* If a cell contains one or more pickups, one of these pickups is moved to a neighboring cell.
- *Grow Cell:* One of the following operators is chosen at random: either all ground tiles that are adjacent to a 1st floor tile transform into 1st floor tiles, or all 1st floor tiles adjacent to a 2nd floor tile transform into 2nd floor tiles.
- *Erode Cell:* Opposite of *Grow Cell*, either 1st floor tiles adjacent to ground tiles are transformed into ground tiles or 2nd floor tiles adjacent to 1st floor tiles are transformed into 1st floor tiles.
- *Place Stairs:* Adds a stair to a random ground floor tile which is adjacent to only one first floor tile (remaining adjacent tiles must be ground floor tiles).
- *Place Block:* A $3 \times 3$ block of first floor tiles and a stair is created if there is enough space on the ground floor. Any pickups in this area are moved to the first floor.
- *Dig Hole:* Within a $5 \times 5$ block of first floor tiles, the central $3 \times 3$ first floor tiles are transformed into ground tiles with a stair directed inwards. Any pickups in this area are moved to the ground floor.

If a mutation is not applicable, (e.g. if a cell does not contain pickups for the first variant), another variant is attempted until either a mutation is applied or all mutation variants are tested.

After mutation and recombination, each map is analyzed in terms of traversability in order to prevent infeasible maps. The following constraints are enforced: (a) bases must always be reachable via ground floor tiles, (b) each pickup must be reachable, (c) each first floor tile must be connected to at least one stair, (d) there must be no holes in an area

TABLE I: Trade-offs between character classes in this paper.

| Class | High | Low |
|---|---|---|
| Heavy | health, rate of fire | speed, accuracy |
| Pyro | damage | accuracy |
| Scout | speed | health |
| Spy | accuracy | clip size |
| Sniper | damage, accuracy | rate of fire, clip size |

of first floor tiles without a stair to climb out of the hole and (e) a stair must always lead to a first floor. A naive constructive algorithm repairs unreachable areas and stair placements (without changing players' bases).

## VI. Experiment

Experiments in this paper target a set of gameplay outcomes for specific matchups between two character classes. Moreover, to show how the algorithm can be used as a design tool for human-authored designs, all evolutionary runs start from a well-formed map and attempt to improve it. Details of this seeding process are given in Section VI-A, while other seeds are tested in Section VI-D.

In order to demonstrate the algorithm, a set of five diverse classes was chosen from *Team Fortress 2*: Heavy, Pyro, Scout, Spy, Sniper. The major trade-offs in terms of their class parameters are shown in Table I. The Heavy, Pyro and Scout all carry short ranged weapons; the Sniper carries a long ranged weapon. The cloaking and knifing abilities of the spy are ignored, treating it as a regular, medium ranged class.

Matching all character classes against each other results in 25 matchup combinations (5 between avatars of the same class). Three target match durations were selected to cover the spectrum of possible game lengths: 200 seconds (short duration), 300 seconds (medium duration) and 600 seconds (long duration). Since the two agents have an equal skill level, the target kill ratio (for P1) was set to $0.5$ (i.e. balanced kills). As mentioned above, the fitness function for evolution is the Euclidean distance from both target gameplay outcomes. Significance tests reported in the paper use $\alpha = 5\%$.

### A. Starting Map

The initial population is seeded with the map shown in Fig.1a: its central area is symmetrical as opposed to the edges. The map has four healthpacks on a diagonal between the two bases and two damage boosts on the first floor at the center of the map. P1 (orange) spawns near a damage boost and P2 (purple) spawns near armor.

Based on simulations on the initial map (as part of the ground truth evaluations discussed in section VI-C), the matches on this map on average last for 266 seconds, but can be as short as 203 seconds (Heavy versus Sniper) and as long as 484 seconds (Heavy versus Heavy). The map is most suitable for Snipers and least suitable for Heavies, judging by their average kill ratio against other classes. This explains the short duration of the Sniper versus Heavy matchup, which ends in a hands-down defeat for the Heavy which manages a kill ratio of $0.2$ (the worst in all matchups). All matches seem to give a slight advantage to P1 regardless of their class.

### B. General Performance

Each class matchup (out of 25) and intended duration (out of 3) is provided as input and intended output respectively and the genetic algorithm attempts to improve the human-authored level in 20 independent runs. Results in this section examine the fittest evolved levels at the end of each evolutionary run, after a maximum of 100 generations (although early stopping is possible). In all cases, the initial population consists of 20 copies of the human authored map of Fig. 1. Unless explicitly noted, all metrics are calculated from the average of these 20 independent runs per matchup and intended duration.

Based on the surrogate model's predictions, the Euclidean distance from the target balance and playtime decrease over the course of evolution, dropping to an average of $0.19$ from $0.42$ in the initial map. As expected, evolution is more challenging for the longer durations which are rarely seen in the training set: there, the average distance using predicted values is $0.35$. The easiest duration to predict is the medium duration, which is very common among matches in the training set and the average distance using predicted values is $0.09$. However, these distances could be low because the model has been overfitting to a skewed training set towards 300 seconds. To verify whether the predictions are reflected in actual gameplay traces, the ground truth in terms of gameplay outcomes is calculated via simulations as described in Section VI-C.

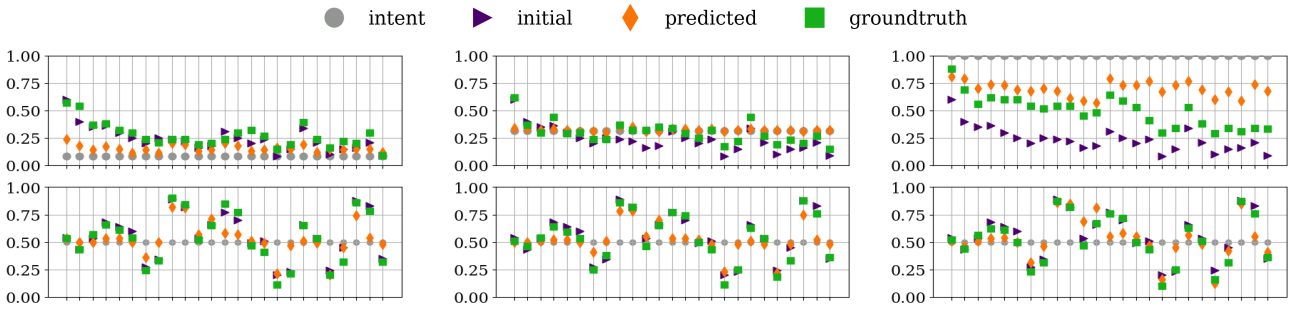
### C. Improvements over the initial map

For the purposes of grounding, the best evolved maps for each run (i.e. 20 maps per class pairing and intended duration) were simulated using the agents on which the model was trained. Each map was simulated 10 times to account for the stochasticity of the AI, and gameplay outcomes (kill ratio, duration) are averaged from those 10 simulations (*ground truth* or GT). The different ground truth, predicted, and initial values per matchup are shown in Fig. 3. Two key performance metrics will be used to compare the evolved maps to the initial map, and to assess the prediction accuracy of the model. We measure *prediction discrepancy* based on Eq. (1), i.e. the difference in the Euclidean distance between intended and predicted values and Euclidean distance between intended and GT values. We measure *improvements from the initial map* based on the difference in the Euclidean distance between the initial map's GT values and intended ones and the Euclidean distance between final map's GT values and intended ones, normalized to the former to give a ratio. This performance metric, formulated in Eq. (2), is positive if the evolved map actually approaches the intended gameplay properties in one or both dimensions compared to the initial map and negative if it actually is more distant than what the designer intended.

$$P(m) = |(dist(m_{pred}, m_i) - dist(m_{gt}, m_i)| \quad (1)$$

$$O(m) = \frac{dist(m_0, m_i) - dist(m_{gt}, m_i)}{dist(m_0, m_i)} \quad (2)$$

where $m$ is the generated map being assessed; $dist(x, y) = \sqrt{(KR(x) - KR(y))^2 + (d(x) - d(y))^2}$ is the Euclidean distance between parameter vectors $x$ and $y$; $KR(x)$ and $d(x)$ is




(b) Short duration (c) Medium duration (d) Long duration

Fig. 3: The feature values of the designer's intent, the initial map, the model's predictions, and the ground truth of the evolved maps. Top row: game duration, bottom row: kill ratio. The data is split along the 25 matchups, ordered alphabetically.

| Duration | $O$ | $P$ |
|---|---|---|
| Short | -10%±18% | 0.14±0.03 |
| Medium | 7%±7% | 0.13±0.03 |
| Long | 28%±4% | 0.23±0.05 |
| **Average** | 8%±7% | 0.16±0.02 |

TABLE II: Improvement over initial map and prediction discrepancy. Values are averaged across all 25 matchups, along with their 95% confidence interval.

the kill ratio and the duration (respectively) in the parameter vector $x$; $m_{gt}$ and $m_{pred}$ is the gameplay parameter vector based on GT and CNN prediction respectively; $m_0$ is the GT parameters for the initial map of Fig. 1; and $m_i$ is the intended gameplay parameter vector specified a priori by the designer.

Observing the improvement of maps evolved for medium and short duration (see Table II), we note some mixed results. On the one hand, maps evolved for medium duration show a minor improvement over the initial map, while maps evolved for short duration often have negative improvements (i.e. moving away from the desired values). It is important to note that for short durations specifically, more matchups have negative improvements (14) than positive (10). Many negative improvements were also observed for medium duration (7), but not more than positive (14). When evolving for long duration, on the other hand, evolution was far more successful in improving the maps compared to the initial state. A likely reason for this discrepancy between short and long durations is because the initial map was mostly favoring short or medium durations than long durations; improving towards short durations was therefore more challenging for evolution.

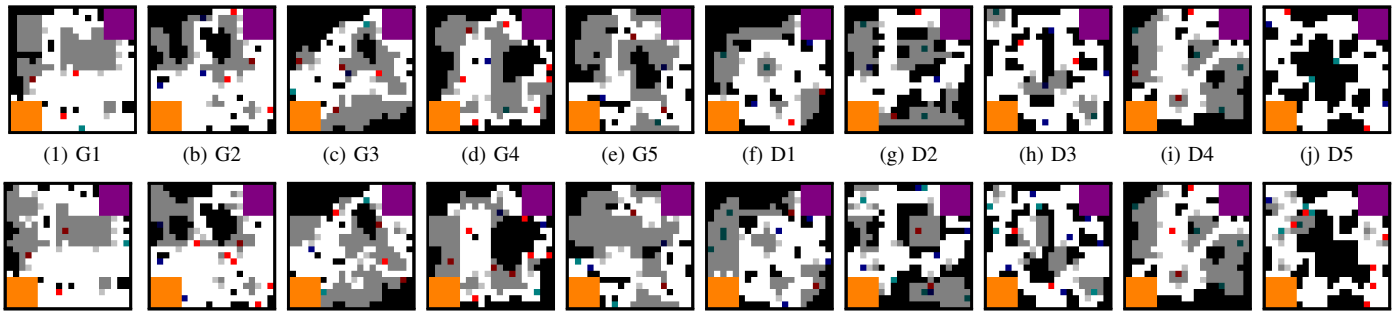
Although maps for long durations were always improved over the initial map (in all 25 matchups), the model overestimated the predicted improvements; the high $P$ value for long durations is primarily due to the fact that most matches in truth took less time than what was predicted. Similarly, for short durations the model overestimated how short the duration would be (see Fig. 3). This is somewhat expected from a regression model (especially one with a worse $R_t^2$ than $R_{KR}^2$), but it should be noted that in the majority of matchups the ground truth durations for maps evolved for short durations were shorter than those evolved for medium (16 of 25 matchups) and maps evolved for long durations were longer than those evolved for medium (25 out of 25 matchups).

It is interesting to note that different class matchups have different performances across intended durations. The matchup with the highest improvement was Scout versus Scout for short durations (42% improvement), Sniper versus Sniper for medium durations (38% improvement), and Heavy versus Heavy for long durations (63% improvement). It is worthwhile to note that the matchups with the highest improvements were predominantly symmetrical (both players have the same class).

### D. Performance with Different Initial Seeds

In order to test the generality of surrogate-based level generation, the same generation methods were applied using a broader range of levels as initial seeds. Unlike the in-depth assessment of the hand-crafted level of Fig. 1, a high-level analysis is provided using $P$ of Eq. (1) and $O$ of Eq. (2) as performance metrics. Ten game levels are used, each of which acts as an initial seed for 75 evolutionary runs: one run for each of 25 *Team Fortress 2* class matchups and each of three intended durations (short, medium, long). The ten levels are shown in Fig. 4: the first 5 levels are created by the same level generator used to produce the corpus, having similar but not identical patterns to levels on which the CNN model was trained on. The last 5 levels were created by a human designer, featuring a degree of symmetry and explicit level patterns such as arenas, choke points and flanking routes [8].

Results averaged across all maps of Fig. 4 for both performance metrics are shown in Table III. Results show some similar trends from Table II, and some differences. On the one hand, due to the large number of initial maps being tested, improvements were highly varied. Each initial map had a different duration and balance per class matchup and thus its improvements could be minor or negative. With generated levels as initial seeds, in particular, many matches had durations very close to the intended medium duration (causing the negative $O$ values). It is not surprising that generated levels generally had match durations close to 300 seconds, as evidenced by the distribution in Fig. 2a. Despite



(1) G1 (b) G2 (c) G3 (d) G4 (e) G5 (f) D1 (g) D2 (h) D3 (i) D4 (j) D5

(k) Scout vs Heavy (M), $O = 74\%$ (l) Heavy vs Heavy (M), $O = 89\%$ (m) Spy vs Spy (L), $O = 88\%$ (n) Spy vs Pyro (L), $O = 100\%$ (o) Heavy vs Scout (M), $O = 93\%$ (p) Heavy vs Spy (L), $O = 96\%$ (q) Scout vs Pyro (L), $O = 93\%$ (r) Pyro vs Scout (L), $O = 100\%$ (s) Scout vs Scout (M), $O = 81\%$ (t) Spy vs Spy (M), $O = 95\%$

Fig. 4: Additional levels used as initial seeds for testing the generality of the method (top) and best evolved maps in terms of improvement ($O$), based on each initial map (bottom). Intended durations are shown as M (medium) and L (long).

| Maps | Generated | Designed | All |
|---|---|---|---|
| Duration | $O$ (improvements from the initial map) | | |
| Short | 13%±6% | 24%±6% | 19%±5% |
| Medium | -31%±47% | 10%±24% | -11%±26% |
| Long | 37%±5% | 32%±6% | 35%±4% |
| **Average** | 6%±16% | 22%±8% | 14%±9% |
| Duration | $P$ (prediction discrepancy) | | |
| Short | 0.13 ±0.02 | 0.16 ±0.02 | 0.15 ±0.02 |
| Medium | 0.14 ±0.02 | 0.15 ±0.02 | 0.15 ±0.02 |
| Long | 0.13 ±0.02 | 0.18 ±0.03 | 0.16 ±0.02 |
| **Average** | 0.13 ±0.01 | 0.17 ±0.01 | 0.15 ±0.01 |

TABLE III: Performance metrics averaged across 5 or 10 maps of Fig. 4 and their 95% confidence intervals.

a few extremely negative $O$ values, in almost all cases the positive improvements were more than the negative ones per initial level and duration individually, and on average[2]. The only exception is for the 25 levels evolved for short duration based on G2 (9 positive versus 16 negative improvements). In terms of predictions, it is not surprising that using a designed level as an initial seed is more challenging than using a generated one (with slightly higher $P$ values across durations) since generated levels have patterns closer to those learned by the CNN model. However, when averaging $P$ values across the many evolutionary runs the differences between intended durations are less pronounced.

Fig. 4 also shows a sample of the evolved levels for a specific class pair and match duration. The evolved map with the best overall improvement ($O$) is shown, but it is interesting to note that not all maps are much changed from their initial states. In many cases, there are minor architectural differences while most changes are focused on the number and type of powerups (e.g. Fig. 4t and 4p). In other cases the powerups largely remain the same, but entire pathways are blocked off (Fig. 4o and 4q) or balconies or galleries introduced (Fig. 4o).

[2]There are 185 positive versus 63 negative in short matches, 173 versus 74 in medium matches, 222 versus 22 in long matches

## VII. Discussion

Based on the training results of Section IV-D, it is evident that the surrogate model is able to learn several interrelations between level and class parameters. However, the biased distribution of durations in the training corpus is evidently hampering the network's ability to accurately predict values outside the medium durations around 300 seconds; this is corroborated by the lower $R_t^2$ value. This lack of precision in predictions affects the performance of the genetic algorithm in non-medium durations, as evidenced by higher $P$ values in most instances of Table II and III. Results are still fairly consistent: most evolved maps have an actual shorter duration when evolving for a short duration than when evolving for a medium or long duration and vice versa. Future work should attempt to create a more fairly distributed dataset in terms of duration. Another notable improvement would be prematurely ending evolution when fitness consistently decreases compared to the initial map. In experiments of Section VI-D, the initial seed matched the intended balance and duration almost perfectly, and therefore evolution explored away from that (due to random mutations in 100 generations). Stopping evolution when maps can not be improved further would enhance results.

It is difficult to ascertain to which degree the inaccuracy of the model has led evolution away from more promising maps than the tested ones, i.e., whether it converged to a false optimum provided by the surrogate model [24]. An important strength of the surrogate model is its speed when it replaces simulation-based evaluations. Indicatively, one evolutionary run as described in Section VI lasts for 3.5 minutes on a 12-core Intel i7 processing unit; a single simulated match (optimized to run without graphics) lasts for 50 seconds on the same machine. If we use the mean gameplay metrics from 10 simulations for each class pairing to account for stochasticity (as used to calculate the ground truth in Section VI-C), one evolutionary run with 20 individuals evolving for 100 generations would last 12 days; even with a single simulation per individual one run lasts 1.2 days. This large computational overhead renders testing the surrogate model

against a pure simulation-based model unrealistic.

It should be noted that the convolutional network has been trained on synthetic playtraces, which may not always match human decision-making. Notable discrepancies when agents navigated the map of Fig. 1 was the fact that P2 rarely exploited the armor pickups of the right side of the map, which was intended as a safe hiding spot. Tactically, therefore, agents do not behave like human players do. While relying on human playthroughs for the vast data required to train deep learning models is not realistic, actual player traces from some of the promising levels can be used to fine-tune the current model.

The results of the experiments point to several possible directions for future work. First, discrepancy between prediction and ground truth values may require a more involved re-training process (with a better duration distribution and possibly human traces, as discussed above). Alternatively, we can re-introduce simulation-based evaluations when the predictive model deems that a map has sufficient quality that a more precise assessment of the gameplay outcomes (via simulations) is warranted. There are various strategies for combining simulation-based evaluations with a surrogate model, such as individual-based, generation-based or adaptive evolution control [7]. Finally, the evaluation could move away from the Euclidean distance which combines both gameplay outcomes (and treats them as equivalent) and use a multi-objective approach [25], treating distance from intended duration and distance from intended kill ratio as individual – possibly conflicting– objectives. The generator might leave this trade-off between fidelity to intended duration and fidelity to game balance up to the designer by presenting the Pareto front (and accompanying maps) to a user for manual selection.

## VIII. Conclusion

This paper demonstrated how deep learning and evolutionary computation could be combined for the purposes of a generative algorithm. The neural network can process game levels as an image and game properties as a parameter vector and predict two core gameplay outcomes. These gameplay predictions can then drive the evolutionary adaptation of game levels towards specific outcomes. Focusing on the domain of shooter games, the model is able to learn the majority of patterns between levels and character classes playing in them. The interrelations between levels, class parameters and gameplay outcomes are then exploited to generate new shooter levels which target a balanced gameplay of short, medium, or long duration for a specific pairing of character classes. Future work could exploit the learned mappings among dissimilar facets to generate content of different types (e.g. character classes), generate multiple types of content simultaneously or as a real-time co-creator for a human level designer.

## Acknowledgments

This research has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 693150.

## References

[1] J. Dormans and S. C. J. Bakkes, "Generating missions and spaces for adaptable play experiences," *IEEE Transactions on CI and AI in Games*, vol. 3, no. 3, pp. 216–228, 2011.

[2] A. Liapis, "Multi-segment evolution of dungeon game levels," in *Proc. of Genetic and Evolutionary Computation Conference*, 2017.

[3] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *IEEE Transactions on CI and AI in Games*, vol. 3, no. 3, pp. 187–200, 2011.

[4] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on CI and AI in Games*, vol. 3, no. 3, 2011.

[5] A. Liapis, G. Smith, and N. Shaker, "Mixed-initiative content creation," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2016, pp. 195–214.

[6] M. Cook, J. Gow, and S. Colton, "Danesh: Helping bridge the gap between procedural generators and their output," in *Proceedings of the IEEE Computational Intelligence and Games Conference*, 2016.

[7] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft computing*, vol. 9, no. 1, pp. 3–12, 2005.

[8] K. Hullet and J. Whitehead, "Design patterns in fps levels," in *Proceedings of the Foundations of Digital Games Conference*, 2010.

[9] B. Marlene, A. Aleksandr, A. Alexander, L. Christoph, N. Felix, W. Jonas, R. Marcel, R. Martin, P. Nicolas, P. Mike, and V. Vanessa, "An integrated process for game balancing," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2016.

[10] A. Liapis, G. N. Yannakakis, and J. Togelius, "Towards a generic method of evaluating game levels," in *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2013.

[11] A. Liapis, "Piecemeal evolution of a first person shooter level," in *Applications of Evolutionary Computation*. Springer, 2018.

[12] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *Proceedings of the Applications of evolutionary computation*, 2011.

[13] A. Zook, B. Harrison, and M. O. Riedl, "Monte-carlo tree search for simulation-based strategy analysis," in *Proceedings of the 10th Conference on the Foundations of Digital Games*, 2015.

[14] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, "Procedural personas as critics for dungeon generation," in *Applications of Evolutionary Computation*. Springer, 2015, vol. 9028, LNCS.

[15] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *arXiv preprint arXiv:1702.00539*, 2017.

[16] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," in *Proceedings of DiGRA & FDG*, 2016.

[17] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, "Predicting resource locations in game maps using deep convolutional neural networks," in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2016.

[18] R. Jain, A. Isaksen, C. Holmgard, and J. Togelius, "Autoencoders for level generation, repair, and recognition," in *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2016.

[19] M. Guzdial, N. Sturtevant, and B. Li, "Deep static and dynamic level analysis: A study on infinite mario," in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2016.

[20] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the International Conference on Computational Creativity*, 2014.

[21] D. Gravina and D. Loiacono, "Procedural weapons generation for unreal tournament III," in *Proceedings of the IEEE Conference on Games, Entertainment, Media*, 2015.

[22] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[23] D. Karavolos, A. Liapis, and G. N. Yannakakis, "Learning the patterns of balance in a multi-player shooter game," in *Proceedings of the FDG workshop on PCG in Games*, 2017.

[24] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[25] C. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007.