# Constrained Surprise Search for Content Generation

Daniele Gravina
Institute of Digital Games
University of Malta
Msida 2080, Malta
daniele.gravina@um.edu.mt

Antonios Liapis
Institute of Digital Games
University of Malta
Msida 2080, Malta
antonios.liapis@um.edu.mt

Georgios N. Yannakakis
Institute of Digital Games
University of Malta
Msida 2080, Malta
georgios.yannakakis@um.edu.mt

***Abstract*—In procedural content generation, it is often desirable to create artifacts which not only fulfill certain playability constraints but are also able to surprise the player with unexpected potential uses. This paper applies a divergent evolutionary search method based on surprise to the constrained problem of generating balanced and efficient sets of weapons for the *Unreal Tournament III* shooter game. The proposed constrained surprise search algorithm ensures that pairs of weapons are sufficiently balanced and effective while also rewarding unexpected uses of these weapons during game simulations with artificial agents. Results in the paper demonstrate that searching for surprise can create functionally diverse weapons which require new gameplay patterns of weapon use in the game.**

## I. Introduction

Procedural content generation has been used since the 1980s in the game industry to quickly and computationally efficiently create elaborate structures such as the dungeons of *Rogue* (Toy and Wichman 1980) or the universe of *ELITE* (Acornsoft 1984). Commercially, procedural content generation (PCG) is used primarily for two reasons: (a) to cut down on development effort and time, and (b) to create unexpected, unique experiences every time the game is played, thus increasing its lifetime and replayability value. Due to the former, small teams of game developers have been able to (procedurally) create grandiose gameworlds such as those in *Minecraft* (Mojang 2011) and *No Man's Sky* (Hello Games 2016). Due to the latter, games such as *Civilization V* (Firaxis 2010) have been immensely successful in retaining a userbase engaged despite the lack of e.g. an overarching campaign.

Academic interest in PCG has often used search-based processes [1] such as evolutionary computation to create game content which optimize one or more game qualities deemed important by the designers. To a degree, such gameplay qualities are required from content in order to ensure the game being playable and fair between players (e.g. in a competitive game). On the other hand, a core motivation of commercial PCG is the element of surprise it can elicit from players. While the majority of PCG research focuses on creating one final artifact which exemplifies the desired properties of its type, there are several attempts at creating a diverse set of content using e.g. multi-objective optimization [2], multi-modal optimization [3] and novelty search [4].

This paper is inspired by earlier work on creating sets of diverse artifacts [5], and applies a recently introduced divergent search algorithm, namely *surprise search* [6], [7], on the task of procedural content generation. In particular, the goal is generating pairs of weapons for a competitive first-person shooter game: the two weapons must be usable and balanced between them, but also exhibit surprising behavioral properties (i.e. different weapon pairs would allow different types of gameplay or strategies to emerge). Towards that end, constraints on usability and balance are satisfied via a feasible-infeasible two-population approach (FI-2pop GA) which guides infeasible content towards feasibility [8]. In the feasible population, however, the weapon pairs evolve towards surprising behaviors, i.e. behaviors that were not predicted based on the previous generations. This *constrained surprise search* algorithm is shown to create more diverse content than objective-driven search. Moreover, its behavior and performance is shown to be different than randomly assigned fitness scores applied on the feasible population.

## II. Background

This section outlines the algorithm of *surprise search* which, in this paper, is framed within *constrained optimization* for the *procedural content generation* domain.

### A. Procedural Content Generation

Compared to the historical use of procedural content generation in games, academic interest in PCG from the perspective of artificial intelligence (AI) is relatively recent. PCG research focuses on expanding the generative algorithms, going beyond *constructive* approaches [1] which are carefully crafted scripts used in the game industry to produce a limited range of content which is however guaranteed to be playable. PCG research on the other hand has used many different sets of algorithms, often revolving around evolutionary computation and constraint satisfaction, among others. Broadly, evolutionary computation under the umbrella term *search-based PCG* [1] evolves a large population of artifacts towards a certain objective, usually pertaining to in-game quality. Constraint satisfaction, on the other hand, uses a carefully selected set of constraints to ensure that all of the generated content is playable [9].

### B. Constrained Optimization and PCG

While it would seem that constraint satisfaction and search-based PCG are incompatible in terms of design approach, there have been several attempts to integrate playability constraints to search-based PCG [10], [4]. Often, the simplest solution is

to assign a minimal fitness score and kill off the infeasible individual [3]. In highly constrained spaces, however, this is not a desirable strategy as most genotypical information is lost [11]. Indeed, if a population consists only of infeasible results then assigning a minimal fitness results in random search. Instead, constrained optimization often utilizes *penalty functions* [12] which reduce the fitness score of an infeasible individual. Designing a penalty function can become as challenging as the optimization problem itself, however, as very high penalties can kill off all infeasible results while very low penalties can lead to extraneous exploration of the infeasible search space. A more recent solution to constrained optimization is the feasible-infeasible two-population (FI-2pop) genetic algorithm [8], which evolves two separate populations towards optimizing a problem-dependent objective (in the feasible population) and minimizing the distance to feasibility (in the infeasible population). The feasible population contains only individuals which satisfy all constraints, while the infeasible population contains individuals which fail one or more constraints; feasible offspring of infeasible individuals migrate to the feasible population and vice versa. The benefit of the two-population approach is that (a) there is no competition between feasible and infeasible individuals, and (b) any search strategy can be applied to either the feasible or the infeasible population. Earlier research on game level generation has explored the use of novelty search in the feasible population or in both populations [4], in order to ensure that playable (due to the constraints) yet diverse (due to the divergent search) game levels were being produced. The current paper explores the use of surprise search [7], a recent but promising divergent search method, on the feasible population for the purposes of creating balanced but surprising weapons.

### C. Surprise search

Surprise search [6], [7] is a new algorithm for evolutionary divergent search which rewards unexpected — rather than unseen — behaviors. Surprise search uses a prediction model to construct the expected outcomes at the current stage of evolution; when evaluating the actual outcomes in the population, it rewards those which deviate from the expected [13]. This mimics a self-surprise process [14], where individuals who do not conform to the evolutionary trend are selected and ensuingly create their own trend which new individuals must again diverge from. The algorithm has been shown to outperform objective search in deceptive problems and to be more robust than novelty search in a maze navigation task [7]. Surprise search is composed by two main modules: a *predictive* model based on past behaviors and a *distance* formula to assess deviation from the expected outcomes. Surprise search uses the prediction model ($m$) to create a speculative 'current' population, based on $h$ previous generations; the model considers a degree of local (or global) behavioral information (expressed by $k$). The predictive model is described in eq. (1); more details about $m$, $h$ and $k$ are found in [7].

$$p = m(h, k) \tag{1}$$

The surprise score, used for selecting individual $i$ in the current population, is based on the distance of the closest $n$ prediction points obtained with the prediction model $m$:

$$s(i) = \frac{1}{n} \sum_{j=0}^{n} d_s(i, p_{i,j}) \tag{2}$$

where $d_s$ is the domain-dependent measure of behavioral difference between an individual $i$ and its expected behavior, $p_{i,j}$ is the $j$-closest prediction point (expected behavior) to individual $i$ and $n$ is the number of prediction points considered; $n$ is a problem-dependent parameter determined empirically.

## III. Methodology

The goal of the generative algorithms is the creation of pairs of usable and balanced weapons which exhibit surprising behavioral characteristics. The weapons are used in the commercially successful *Unreal Tournament III* (Epic Games 2007) game (UT3). Besides its commercial appeal, UT3 has well-designed game levels and AI modules which allow for simulations of game matches in order to derive behavioral properties of the weapons. Weapons in UT3 are already quite diverse, which allows the genetic algorithm to explore different sets of parameters such as bouncing bullets, grenades affected by gravity, or exploding projectiles.

### A. Representation & Genetic Operators

In the genotype, each weapon is represented by 11 parameters with different value ranges and in-game properties as shown in Table I. Since the generator evolves pairs of weapons (one per player in a deathmatch FPS game), the genotype therefore consists of 22 chromosomes, 11 for each weapon. Evolution is carried out by applying simulated binary crossover with a $60\%$ probability, and simulated binary mutation with a $5\%$ probability. These parameters have been chosen empirically via pre-experimentation conducted in [5]. Simulated binary crossover [15] applies a polynomial probability distribution (controlled by the maximum and minimum values of each parameter in Table I) to chromosomes; another parameter ($\eta = 20$ as suggested in [15]) controls how much the offspring will resemble their parents. This crossover strategy ensures that the weapon of each player will be a combination of parameters of weapons used by the same player (i.e. weapons cannot be assigned to a different player from generation to generation). Simulated binary mutation performs a similar modification with a chance of $5\%$ for each parameter in the gene. Using the same $\eta$ value, modifications via mutation depend on the value range of each weapon parameter (e.g. low $\eta$ values result in large mutations).

### B. Simulations

The two weapons evolved in this scenario are tested by two AI-controlled agents competing for the highest number of kills in a UT3 level. Experiments in this paper use the *Biohazard* UT3 level, which is small and thus ideal for one-versus-one matches; moreover, it consists of two separate floors which makes logging player positions easier. Each player is given

TABLE I
PARAMETERS OF EACH WEAPON WITH THEIR CORRESPONDING VALUE RANGE AND DESCRIPTION.

| Name | Value Range | Description |
|---|---|---|
| Rate of Fire (ROF) | [0, 4] | Number of bullets shot per second |
| Spread (Spr) | [0, 3] | This parameter affects the random deviation of the bullets trajectory: the higher the spread the less accurate the shooting. |
| ShotCost (SC) | [1, 9] | Number of bullets shot at once by the weapon. |
| Lifetime (L) | [0, 100] | Amount of time the bullets remain in game when shot. |
| Speed (Sp) | [0, 10000] | Speed of bullet when shot. |
| Damage (Dmg) | [0, 100] | The amount of damage that each shot deals when it hits an opponent. In case of SC $>$ 1, each bullet has Dmg/SC damage per bullet. |
| Collision Radius (CR) | [0, 100] | Radius of the collision sphere of bullets (for hitting enemies). |
| Gravity (Gr) | [-250, 250] | Gravity force applied to bullets: the larger the value, the stronger the $g$ acceleration applied to the bullet. For positive values, gravity is reversed (the bullet goes upwards). |
| Explosive (Exp) | [0, 300] | When a bullet hits a target (opponent, object or wall), it generates an explosion with radius equal to this parameter. All players within the radius of an explosion receive splash damage (a fraction of the weapon's damage depending on distance). |
| Ammo (A) | [1, 999] | Maximum amount of ammunition; all ammo packs increase ammo up to this value. |
| Bounce (B) | [0, 1] | Boolean value that says if the projectile will bounce when it hits a wall. |

one weapon and ammunition as defined in the genotype; if they pick up any weapon or ammo in the level then the ammo for their generated weapon increases by the ammo value in the genotype (i.e. players cannot pick up other weapons, including the other player's weapon). This ensures that each player tests only one weapon: these simulations allow for a comparison in terms of balance of the weapons, as well as for evaluating their effectiveness (if it kills the opponent often) and safety (if it does not result in the wielders shooting themselves). Moreover, simulations are used to create a map of the death locations of each player; these act as behavioral characteristics of the weapons and are used to assess unexpected behaviors in the surprise search algorithm. Simulations last up to a *time limit* of 1200 seconds or until a *score limit* of 20 is reached in terms of the total number of kills of the two players.

Since simulations require that AI agents use weapons of variable quality, the system provides suggestions to the AI behavior based on each weapon's parameters. For instance, if the weapon is a fast repeater (i.e. a rate of fire above 2) the AI is instructed to use it for long sequences of shots. If its bullets have a long lifetime, high speed and low spread, the AI is instructed to use it for long distance shots. Finally if the bullet has an explosive value above 50, the AI is instructed to treat it as a *splash weapon*, which relies less on accuracy.

### C. Constraints

There are certain playability requirements for the generated weapons: balance, effectiveness and safety. Each of these properties can be evaluated as a scalar value, via heuristics discussed below, based on simulations between AI controlled agents. A pair of weapons is considered playable (i.e. feasible) if each property is above a specific threshold. Moreover, for infeasible individuals the heuristics can be used to derive the distance from feasibility with regards to each constraint.

*Balance* is computed as the Shannon Entropy [16] of the kills obtained by the two agents:

$$f_b = \frac{1}{n}\sum_{j=0}^{n}\left(\frac{k_i}{K}log\left(\frac{k_i}{K}\right)\right) \tag{3}$$

where $K$ is the total number of kills obtained by the two agents in a simulation, $k_i$ are the kills obtained by $i$-th bot and $n$ is the number of players per simulation.

*Effectiveness* is calculated by dividing the total number of kills obtained in the simulation by the maximum *score limit*:

$$f_e = \frac{K}{S_{max}} \tag{4}$$

where $S_{max}$ is the score limit ($S_{max} = 20$ in this study) which must be attained for the level to be considered completed before the time limit expires.

*Safety* is introduced due to initial random weapons being dangerous to the wielder (due to high explosive values), and its goal is to make evaluations more robust against noise. Safety is computed in eq. (5) where the exponent is the number of suicides (i.e. deaths not scored as another player's kill):

$$f_s = 0.9^{D-K} \tag{5}$$

where $D$ is the total number of agents' deaths in a simulation.

The feasibility constraint is satisfied if $f_b \geq 0.9$; $f_e \geq 1$ (i.e. if exactly 20 kills are scored); $f_s \geq 0.9$ (i.e. if there's at most one suicide). The rationale for the strict thresholds for effectiveness and safety are to avoid creating sparse heatmaps of death locations (due to low effectiveness) or death locations originating from suicides (due to low safety).

### D. Constrained Surprise Search

Constrained surprise search fuses the properties of FI-2pop constrained optimization [8] with surprise search [7] evolving the feasible population. The proposed algorithm uses two populations which evolve towards different goals. The feasible population contains individuals which satisfy all constraints listed above, while the infeasible population contains individuals which have at least one of safety, balance and efficiency below the minimal threshold. The infeasible population assigns its members a fitness equal to $f_b + f_e + f_s$, regardless of whether some of the values of these properties are above the feasibility threshold. This favors individuals which satisfy more constraints to others which satisfy no constraints,

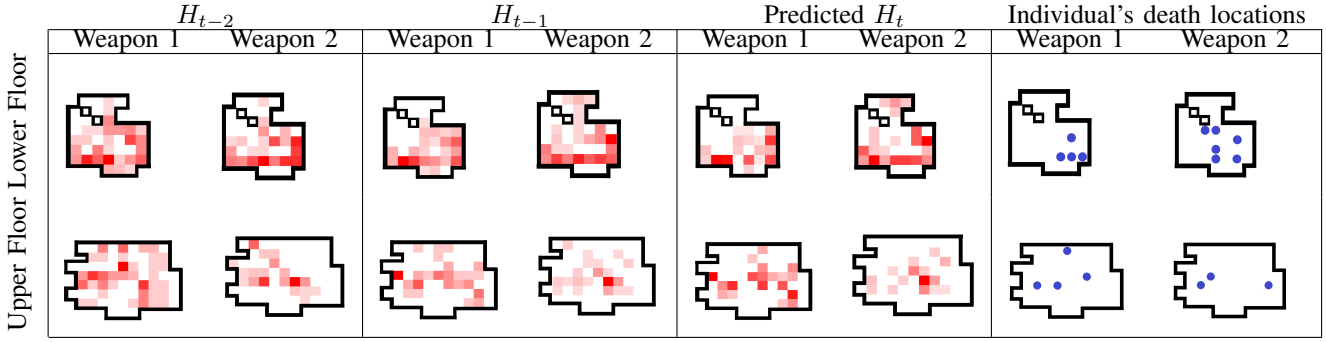



Fig. 1. An example of the prediction model of surprise search in this paper, at generation $t$. The first two sets of heatmaps are computed in the last two generations, $H_{t-2}$ and $H_{t-1}$; the death location density is always normalized per floor. Using linear interpolation, the difference $H_{t-1} - H_{t-2}$ is computed and applied to $H_{t-1}$ to derive the predicted current population's $H_t$ truncated to $[0, 1]$. An individual's death locations are mapped to $H_t$ to calculate the surprise score per eq. (6).

although some averaging artifacts may occur. Unlike traditional FI-2pop approaches, the infeasible population attempts to maximize this value, as the three properties act as objectives (with minimal value constraints).

Due to the highly constrained search space that evolution has to tackle, several steps are taken to make search in the feasible population more effective. Both populations select parents based on tournament selection (tournament size of 3), but the best individual of each population is copied as-is to the next (elitism of 1). This elitism ensures that at least one feasible individual will remain in the feasible population. Moreover, if the feasible population is smaller than the infeasible population, an *offspring boost* [4] is applied: the offspring boost forces the (larger) infeasible population to only produce offspring equal to 50% of the total population while the feasible population produces more offspring than it has parents, equal also to 50% of the total population.

In the feasible population, surprise search attempts to deviate from predicted behavioral trends of the current population. *Behavior* of a weapon is considered to be the playtraces of the player who wields it, and in particular the locations where their opponent died in this one-versus-one deathmatch game. Since the genotype contains two weapons and the *Biohazard* level consists of two floors, this creates a total of 4 heatmaps of death locations of each player. These *heatmaps* assign each death on a tile of a low-resolution grid (10 by 13 tiles per floor), incrementing the value (or heat) of that tile; example heatmaps are shown in Fig. 1. Note that heatmaps are normalized to a range of $[0, 1]$ based on the maximum heat value of each map (i.e. per floor and per player).

Surprise search attempts to deviate, therefore, from the expected heatmaps of this generation: i.e. have death locations which are unexpected based on the current evolutionary trends. Surprise search focuses on diverging from predictions $p$ (see eq. (1)) of the current population, calculated by observing the previous generations' behavioral changes. This paper uses only the populations of the last two generations ($h = 2$; eq. (1)) to predict the current population, applying a linear interpolation ($m$ is a linear regression model in eq. (1)). The model, $m$, considers the population as a whole ($k = 1$; eq. (1)). In short, when predicting the heatmaps $H_t$ for a population at generation $t$, the heatmaps of the population at $t - 2$ ($H_{t-2}$) is subtracted from those of the population at $t - 1$ ($H_{t-1}$) to calculate $\Delta H$. The prediction of $H_t$ is obtained by adding $\Delta H$ to $H_{t-1}$, ensuring that its values fall within $[0, 1]$ in all 4 heatmaps. Figure 1 illustrates this procedure.

In order to derive a surprise score for an individual $i$ (which the surprise search algorithm attempts to maximize), the locations of its agents' deaths are mapped to the appropriate cell of predicted heatmaps $H_t$ (depending on floor and player). The surprise score is calculated in eq. (6), as the complementary of the average cell values of $H_t$ where deaths occurred in individual $i$. This rewards individuals which diverge from the predicted consensus of the general population.

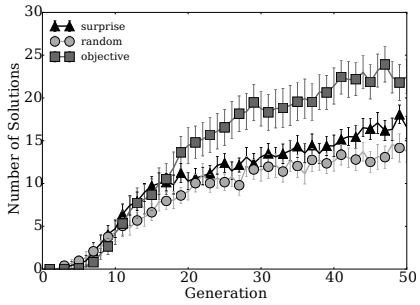
$$s(i) = 1 - \frac{1}{D} \sum_{d \in D} H_t(d) \tag{6}$$

where D is the number of deaths of all agents in individual $i$ and $H_t(d)$ the cell value of $H_t$ at the location of death $d$. This rewards individuals which diverge from the predicted consensus of the general population.

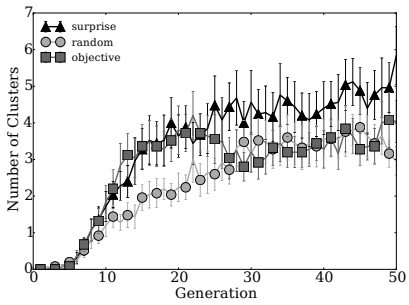
This evaluation of surprise is different from that of novelty used in novelty search [17], as the latter deviates from the actual population rather than its prediction. By using a prediction of the population, surprise search creates data (heatmaps in this case) which may never be attainable: diverging from such may push search in unexpected areas of the space.

## IV. Experiments

This paper aims to generate surprising weapons which have a modicum of balance, safety and efficiency. Towards evaluating constrained surprise search in terms of these different priorities, several tests are performed on the results of 25 independent optimization runs of constrained surprise search. The algorithm is compared with two baseline algorithms in terms of constraint satisfaction and diversity preservation; the most diverse solutions of constrained surprise search are then assessed in terms of their use by AI agents in game simulations; finally, a sample set of weapons evolved by constrained surprise search is presented in detail, showcasing how the different weapons are surprising yet balanced.




(a) Feasible individuals.




(b) Feasible clusters obtained with DBSCAN.

Fig. 2. Progress of two different performance metrics over the course of evolution averaged from 25 independent runs. Error bars depict standard error.

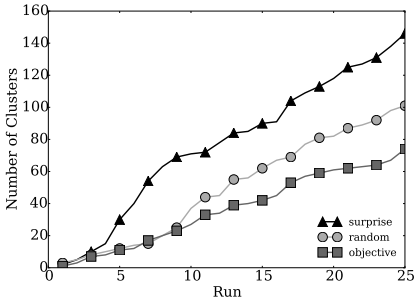



Fig. 3. Feasible clusters obtained with DBSCAN in the combined final populations of multiple evolutionary runs.

### A. Comparison with other methods

In order to assess the performance of constrained surprise search, its outcomes and overall optimization progress will be compared to two benchmarks: (a) objective search, which attempts to optimize the sum of balance, safety and efficiency ($f_b + f_s + f_e$), (b) constrained random search, which uses the FI-2pop paradigm but performs random search on the feasible population. The objective search does not need two populations, as it essentially amounts to the search in the infeasible population without minimal feasibility requirements. The constrained random search evolves the infeasible population to maximize $f_b + f_s + f_e$ while the feasible population assigns a random fitness within $[0, 1)$ to each of its members.

For the purposes of comparing the performance of the three algorithms, it is not straightforward which performance metrics are most appropriate for evaluating surprise or diversity. On one hand, it is relevant to evaluate how well-suited each algorithm is for constrained optimization: for that reason we use the *number of feasible individuals* as a measure of how the different search methods on the feasible space (in the case of surprise and random search) may create infeasible offspring from feasible parents. In order to evaluate diversity in the feasible population, we use the *pairwise genotypic distance of feasible individuals* to measure how different (at least in terms of weapon parameters) the genotypes are. The genotypes' values are normalized between [0,1] via min-max normalization based on each parameter's value range in Table I. However, it should be noted that the number of feasible individuals in the population may not be sufficiently diverse, and thus a smaller set of the most diverse results would be more appropriate both for in-game use and as a performance metric. In previous work [3], this set of "solutions" was discovered via $k$-medoids where $k$ was a property specified by the designer. In this paper, such solutions are obtained by DBSCAN [18] which can return a variable number of clusters (and their medoids) depending on the distribution of data. Therefore, it is possible to gauge the diversity of the population based on *the number of clusters found by DBSCAN*. DBSCAN is a density-based clustering technique [18] which groups individuals based on their nearest neighbor distance. DBSCAN depends on two parameters: a distance $\epsilon$,which is the distance from a randomly chosen point for considering its neighbors, and the minimum number of points within $\epsilon$ from a random point in order to be considered a cluster; in this paper $\epsilon$ is 0.2 and the minimum number of points is set to 1.

Reported results are collected from 25 independent optimization runs per approach; evolution lasts for 50 generations and is performed on a total population size of 50 individuals. Reported significance is obtained from two-tailed Mann-Whitney U-tests at a $5\%$ significance level.

Figure 2a shows the number of feasible individuals for each method, as evolution progresses. It is immediately obvious that at the start of evolution there are no feasible individuals, which indicates a highly constrained search space. Since all methods evolve infeasible individuals towards the same objective, it is not surprising that all methods discover the first feasible individual in approximately the same generation, i.e. generation 10 or so. More interestingly, the number of feasible individuals (in a total population of 50) keeps increasing throughout evolution as more and more infeasible individuals approach the border of feasibility. With the offspring boost, ideally the feasible individuals would be equal to the infeasible ones; however, feasible parents are more likely to create infeasible offspring than the reverse and thus feasible individuals are fewer. Objective-driven search tends to create more feasible results, while interestingly random search on the feasible population is not more destructive (in terms of feasible population size) than surprise search. Objective-driven search is expected to create more feasible individuals, primarily due to the fact that it uses a single population: therefore, feasible results are more likely to get selected (multiple times) and thus create more feasible offspring. Despite the feasible offspring boost in the FI-2pop approaches, this single population approach which continually tries to improve upon the constraints (even after all constraints are satisfied) is more efficient at creating feasible results.

DBSCAN is able to identify distinct clusters, so the number of clusters should be an indication of the population's diversity: essentially, DBSCAN plays the role of a designer choosing the most representative weapons (the clusters' medoids). The number of clusters found among feasible individuals (for a threshold of 0.2) is shown in Fig. 2b. While the

TABLE II
PERFORMANCE METRICS AT THE END OF 50 GENERATIONS. RESULTS ARE AVERAGED FROM 25 INDEPENDENT RUNS, WITH THE STANDARD ERROR SHOWN IN PARENTHESES. DIVERSITY REFERS TO THE AVERAGE PAIRWISE GENETIC DISTANCE.

| | Surprise | Objective | Random |
|---|---|---|---|
| Feasible Individuals | 16.6 (1.28) | 22.5 (1.98) | 13.8 (1.29) |
| Feasible Clusters | 5.84 (0.78) | 4.04 (0.58) | 2.96 (0.40) |
| Individuals' Diversity | 0.29 (0.03) | 0.26 (0.02) | 0.24 (0.02) |
| Medoids' Diversity | 0.33 (0.04) | 0.28 (0.03) | 0.26 (0.04) |

objective-based approach creates more feasible individuals, it is obvious that these individuals are genotypically similar leading to fewer clusters than the smaller feasible population of constrained surprise search. As the first feasible individuals appear in the population, both objective-driven search and surprise search start with a diverse population (and thus many feasible clusters). However, as objective-based selection prioritizes feasible individuals almost exclusively (i.e. when the number of feasible individuals increases after generation 20), objective-driven search converges to a few promising areas of the search space resulting in a drop in the number of clusters. By comparison, the behavioral surprise prioritized by surprise search manages to better preserve the genetic diversity of the initial feasible individuals. It should be noted that in the 25 runs performed for the reported results, there is a large deviation, on average, between the number of clusters for every approach, as can be gleaned from Fig. 2b.

Table II shows the final scores of the different performance metrics at the end of 50 generations. While constrained random search is able to maintain a sufficiently large feasible population, the number of distinct clusters found by DBSCAN is significantly lower ($p < 0.05$) than those of constrained surprise search. Objective search creates significantly more feasible individuals on average than constrained surprise search ($p < 0.05$), but they are not as diverse (based on the number of clusters); due to large deviations in the number of clusters, significance can not be established. As additional metrics, the average pairwise genotypic distance of all feasible individuals and of the cluster medoids is compared. Constrained surprise search tends to create more genotypically different medoids than both objective search and constrained random search.

As another measure of diversity, DBSCAN is applied on the final populations of multiple evolutionary runs and the number of discovered clusters is plotted in Fig. 3. Surprise search seems robust at finding clusters in the combined populations, and thus does not converge to the same local optima in every run. With few runs, the differences between random and objective search are minimal while surprise search finds far more clusters; the order of evolutionary runs being combined could have a minor effect in this. DBSCAN finds 146 feasible clusters in 415 feasible individuals collected from 25 runs of surprise search, versus 101 clusters in 563 feasible individuals of objective search. Indicatively, Fig. 3 shows that collecting 40 distinct weapon pairs (i.e. medoids) requires 6 runs of surprise search versus 11 and 14 runs of objective and random search respectively. Combined with a slightly higher medoid diversity, surprise search seems preferable for discovering more diverse weapons at a lower computational cost.

TABLE III
GAMEPLAY METRICS OF ALL CLUSTER MEDOIDS OF ALL 25 INDEPENDENT RUNS OF CONSTRAINED SURPRISE SEARCH. RESULTS ARE AVERAGED FROM 10 SIMULATIONS, WITH DEVIATION BETWEEN MEDOIDS IN PARENTHESES.

| | Both floors | 1st Floor | 2nd Floor |
|---|---|---|---|
| Total Kills | 15.14 (0.27) | 10.81 (0.18) | 4.29 (0.15) |
| Kills 1st | 7.83 (0.21) | 4.32 (0.22) | 3.51 (0.08) |
| Kills 2nd | 7.3 (0.21) | 5.11 (0.18) | 2.19 (0.08) |
| Balance | 0.87 (0.009) | 0.84 (0.011) | 0.71 (0.016) |
| Entropy 1st | 0.71 (0.002) | 0.63 (0.001) | 0.74 (0.003) |
| Entropy 2nd | 0.72 (0.003) | 0.64 (0.002) | 0.76 (0.004) |

### *B. Gameplay Qualities of Weapons*

In order to assess a modicum of the gameplay uses of the generated weapons, all cluster medoids discovered in the 25 surprise search runs were tested in 10 simulations each. Table III shows certain gameplay metrics of these weapons, averaged from 10 simulations: '1st' and '2nd' refers to the first and second player, while 'entropy' refers to Shannon's Entropy of the death locations' heatmaps on both floors.

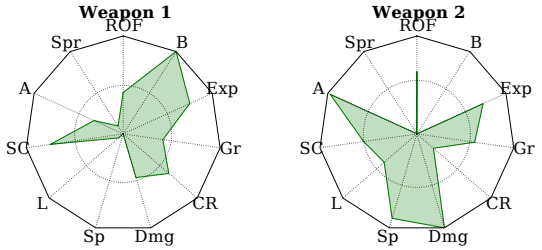
From Table III, it is clear that the number of total kills are on average below the minimal playability thresholds set in Section III-C; this points to a very noisy simulation-based evaluation. During optimization, it seems that a single simulation can decide that a weapon pair is playable in one generation and reject the same pair in the next. Based on the number of kills for each player, the calculated balance for the weapon pair is 0.87 on average, which is also slightly below the minimal threshold for $f_b$. Suicides on the other hand remain low at an average of 0.44 (standard error of 0.04), and therefore the safety constraint is always satisfied.

An interesting insight from Table III is the gameplay difference between the two floors: significantly more kills occur in the 1st floor for both weapons ($p < 0.05$), which should not be surprising since the second floor is not as accessible and offers a better vantage point to fire at enemies below. On the other hand, deaths in the second floor are significantly more dispersed spatially (based on Shannon's entropy) than those in the first floor ($p < 0.05$): this is also obvious from $H_{t-1}$ and $H_{t-2}$ of Fig. 1. This is can be traced to the fact that the second floor is in theory larger; moreover the second floor includes narrow bridges which partition the space and therefore players tend to die on opposite ends of that floor. The differences in both number of deaths and entropy of death locations could affect the performance of surprise search, however, which currently normalizes each heatmap individually; if the first floor has far more deaths than the second floor, treating them equally in terms of surprise places unnecessary impact on the second floor. A potential solution in future experiments could be to normalize heatmaps based on the total number of deaths rather than the number of deaths per floor and per player.

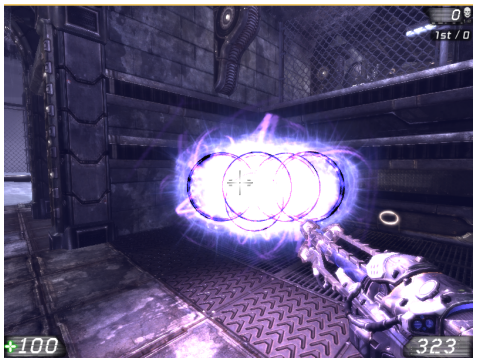
### C. Sample Weapons

In order to discuss a sample of the generated weapons in more detail, a weapon pair was chosen agnostically among the DBSCAN cluster medoids for surprise search; there is no assumption that this weapon pair exemplifies all generated content. For the sake of screenshots, weapons use the shock rifle weapon model and bullet effects in UT3.

Figure 4a shows a genotype collected after 50 generations of constrained surprise search. The first weapon fires very concentrated projectiles (maximum shot cost and very low spread) with no gravity and very low speed; the second weapon shoots very fast projectiles, with no spread and high damage. Trying these two weapons as a player, one quickly realizes that the first weapon creates 'mines' around the map (see Figure 4b): its bullets are extremely slow, with a large blast area (explosive, high collision radius) and they can also bounce on walls or the level's floor. Moreover, these 'mines' are fired in clusters (high shot cost) as seen in Fig. 4b, thus costing a lot of ammo (of which the weapon has little). This weapon seems over-powered, allowing its wielder to control the map; however as the bullets do not have a long lifetime, they are effective only if the opponent is nearby and running towards the wielder. Meanwhile, the second weapon is very similar to a rifle: high-damage fast bullets which shoot straight (trivial gravity effects) with a very low collision radius, thus requiring precise aiming. Unlike traditional rifles, however, the weapon's bullets have some explosive qualities. Obviously, a match-up between these two weapons requires a very different strategy from each player: the first weapon requires its wielder to move around the level, laying 'mines' in chokepoints when the other player is nearby. Meanwhile, the player with the second weapon does not need to move as much (also in order to avoid any mines) as she can fire her high-speed, precise and lethal bullets from a remote location.




(a) Sample weapon pair evolved via constrained surprise search.




(b) Weapon 1, evolved via constrained surprise search.

Fig. 4. Example of weapons evolved via constrained surprise search

## V. Discussion

The primary goal of this paper was to discover balanced and efficient, yet surprising weapons via constrained surprise search. Results indicate that constrained surprise search tends to evolve genotypically more diverse pairs of weapons, which have unexpected in-game uses. The FI-2pop paradigm also allows this method to discover feasible individuals quickly and consistently; preliminary experiments with surprise search or random search on a single population without constraints failed to find feasible results at any point of their evolution during 50 generations. Comparing constrained surprise search with constrained random search shows a (statistically) significant improvement of the former in terms of number of clusters: this indicates that surprise search is substantially different in terms of both process and results than random search. On the other hand, performing objective search on a single population finds significantly more feasible individuals; due to the selection pressure towards feasible individuals, however, much of the genetic diversity is eventually lost. While objective search has more feasible individuals, those are not necessarily as diverse or interesting as in surprise search.

There is an obvious limitation to the generalizability of the results reported in this paper, for several reasons: (a) the high deviation in the performance metrics from one run to the next makes assessing significance problematic, (b) there are many parameters which could be fine-tuned to improve both the algorithms and the assessment method. Regarding the deviation found in reported values, the need for simulations of full games (lasting up to 20 minutes) to assess each individual prevents extensive experiments for parameter tuning or for more runs to assess significance. Future work should explore how surprise search performs with different behavioral characteristics, as well as compare its performance against more methods including two-population objective search or two-population novelty search [4]. Finally, the choice of $\epsilon$ in DBSCAN also affects the results and conclusions; with a much lower $\epsilon$ the more numerous individuals of objective search create more clusters (e.g. with one individual per cluster), and with a much higher $\epsilon$ all methods create a single cluster.

Surprise search in this particular problem predicts the behavior of the population as a whole, based on the behavior of the previous two populations. In the general predictive model of eq. (1), the current approach uses the simplest predictive model $m$ (a linear model), the shortest history ($h = 2$) and there is no locality as the model aggregates the entire population ($k = 1$); for this reason the distance calculation in eq. (2) considers one neighbor ($n = 1$). Obviously there is a broad range of parameters to explore in order to improve the performance of surprise search, such as using a non-linear regression model or including a form of archive as in novelty search [17]. Another possible improvement could be choosing another behavioral characteristic to deviate from:

currently the system considers a "heatmap" of death locations; this heatmap is relatively sparse, also considering that the level has two floors. In many cases the differences between two such heatmaps is circumstantial, also due to the high stochasticity of combat; this was mitigated by using the lowest locality for surprise search and aggregating a heatmap for the entire population. Other behavioral characteristics could be considered, such as players' movements in the level or scalar gameplay values such as the entropy and ratio of kills in each floor or the distance between players at the time of death.

Finally, it should be noted that simulations with AI opponents are a necessity due to the numerous matches which need to be played per generation during evolution. The enemy behavior in UT3 is quite well-designed and competitive (at least for novice players), unlike many open-source games such as *Cube 2* used in other experiments [19]. However, the AI in UT3 is designed for the weapons included in the game which have specific parameters and uses; with generated weapons, unexpected combinations of weapon parameters such as high spread on a sniper rifle may result in less than ideal agent performance. Currently, this is somewhat mitigated when initializing simulations by instructing the AI that certain weapons with certain properties should be treated as UT3 weapons (e.g. high lifetime weapons use the sniper rifle AI module). However, completely unique weapons which do not have any AI module may be impossible to use: for instance, a mine weapon requires a very different navigational strategy, going through chokepoints and leaving mines. While the AI could be improved, it is perhaps more interesting to test the final weapons generated by each approach in a user survey with competing expert players. Observing emergent gameplay strategies, and how the meta-game is affected by unexpected uses of these weapons can offer a better insight of the usability and balance of the generated weapon pairs.

## VI. Conclusion

This paper introduced a constrained form of surprise search and applied it on a procedural content generation problem. The first goal of the generator is to create pairs of weapons for *Unreal Tournament III*, which have a balanced and efficient performance when played in simulated matches with AI agents. Towards that end, a feasible-infeasible two-population genetic algorithm was employed to maximize balance, efficiency and safety on an infeasible population until those values were above a required threshold. The second goal of the generator is to create pairs of weapons which have surprising properties and can result in interesting, unconventional gameplay. For this purpose, surprise search is applied on the feasible population, attempting to deviate from the expected behaviors of the AI agents (i.e. their death locations) which were predicted from past generations. Results in the paper show that the feasible-infeasible approach is able to find feasible individuals quickly and reliably, and that surprise search tends to create more diverse (if not always more) content. Since the reported experiment is small-scale, there is a broad range of future directions for improving surprise search and its parameters, testing it against more algorithms, and exploring other behavioral or gameplay characteristics other than death locations.

## Acknowledgment

This work has been supported, in part, by the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665) and the Horizon 2020 project CrossCult (project no: 693150).

## References

[1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-Based Procedural Content Generation: A Taxonomy and Survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, Sept 2011.

[2] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbck, G. N. Yannakakis, and C. Grappiolo, "Controllable procedural map generation via multiobjective evolution," *Genetic Programming and Evolvable Machines*, 2013.

[3] M. Preuss, A. Liapis, and J. Togelius, "Searching for good and diverse game levels," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2014.

[4] A. Liapis, G. N. Yannakakis, and J. Togelius, "Constrained novelty search: A study on game content generation," *Evolutionary Computation*, vol. 23, no. 1, pp. 101–129, 2015.

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

[6] G. N. Yannakakis and A. Liapis, "Searching for surprise," in *Proceedings of the International Conference on Computational Creativity*, 2016.

[7] D. Gravina, A. Liapis, and G. N. Yannakakis, "Surprise search: Beyond objectives and novelty," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2016.

[8] S. O. Kimbrough, G. J. Koehler, M. Lu, and D. H. Wood, "On a feasible-infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch," *European Journal of Operational Research*, vol. 190, no. 2, pp. 310–327, 2008.

[9] A. M. Smith, E. Butler, and Z. Popović, "Quantifying over play: Constraining undesirable solutions in puzzle design," in *Proceedings of ACM Conference on Foundations of Digital Games*, 2013.

[10] N. Sorenson, P. Pasquier, and S. DiPaola, "A generic approach to challenge modeling for the procedural creation of video game levels," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, 2011.

[11] Z. Michalewicz, "Do not kill unfeasible individuals," in *Proceedings of the Fourth Intelligent Information Systems Workshop*, 1995, pp. 110–123.

[12] C. A. Coello Coello, "Constraint-handling techniques used with evolutionary algorithms," in *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*. ACM, 2010, pp. 2603–2624.

[13] K. Grace, M. L. Maher, D. Fisher, and K. Brady, "Modeling expectation for evaluating surprise in design creativity," in *Design Computing and Cognition*, 2014.

[14] K. Grace and M. L. Maher, "What to expect when you're expecting: The role of unexpectedness in computationally evaluating creativity," in *Proceedings of the International Conference on Computational Creativity*, 2014.

[15] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," vol. 9, pp. 115–148, 1995.

[16] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[17] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, 2011.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[19] P. L. Lanzi, D. Loiacono, and R. Stucchi, "Evolving maps for match balancing in first person shooters," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2014.