Procedural Weapons Generation for Unreal Tournament III

Daniele Gravina
Dipartimento di Elettronica, Informazione e Bioingegneria
Politercnico di Milano
Milan, Italy
Email: daniele.gravina@mail.polimi.it

Daniele Loiacono
Dipartimento di Elettronica, Informazione e Bioingegneria
Politercnico di Milano
Milan, Italy
Email: daniele.loiacono@polimi.it

***Abstract*—Design a set of weapons for a competitive first person shooter (FPS) is not an easy task: it involves several challenges, such as balancing, diversity, and novelty. In this paper, we combine procedural content generation and evolutionary computation to solve this complex task. In particular, we introduce a novel approach to procedurally generate weapons for Unreal Tournament III, a popular commercial FPS. In our approach, each weapon is represented as a parameters vector and evaluated based on game statistics. We tested our approach on three different application scenarios that represents three typical design problems: (i) create a balanced set of weapons, (ii) improve a given weapon with the least possible changes, and (iii) create a set of weapons with some specific design goals. Finally, we also performed a preliminary user study to validate our work. Our results are promising and suggest that the proposed approach might be successfully used to support the weapon design process.**

I. Introduction

Nowadays, game industry is very competitive and to succeed it is very important to feature a large amount of game content with high quality and high degree of variety. Unfortunately, the design of such content is one of the most expensive and time consuming activities of the whole development process. In fact, the lack of time and resources to implement the pipeline required to produce high quality contents is often one of the major factor that set the AAA productions apart from the middle-sized ones. To make the problem even more challenging, different game genres require to produce very different kind of contents with their specific problems and peculiarities. In particular, for a competitive first person shooter (FPS), the weapons available in the game play a key role: with dozens of titles available on the market, developers always aim at designing a set of weapons that stands out with respect to the competitors.

In this work, we combine procedural content generation (PCG) and evolutionary computation to generate weapons for *Unreal Tournament III* (UT3), a rather popular competitive first person shooter. Our aim is to propose a novel approach to design a set of balanced weapons with a large amount of variety and suitable for different playing strategies. We extended UT3 so that a weapon can be represented with a vector of parameters and it can be tested in game. Accordingly, we propose several metrics to evaluate the weapons based on few game statistics that can be easily collected through a simple simulation. Therefore, we described three possible scenarios where an evolutionary algorithm can be applied to evolve weapons, using the proposed metrics as a fitness function. Finally, to assess the quality of the evolved weapons, we carried out a preliminary user study with more than 30 users. Overall, our results are promising and we believe that our approach could be useful to assist developers during the design of a set of weapons.

The paper is organized as follows. In Section II we provide an overview of the related work. Then, in Section III we briefly present UT3 and in Section IV we illustrate our approach. Section V, Section VI, and Section VII presents the three application scenarios and discusses the experimental results. Finally, in Section VIII we report the results of our user study and in Section IX we draw some conclusions.

II. Related Work

Back in the early days, procedural content generation (PCG) was originally used in games as *Beneath Apple* [1] and *Elite* [2] to deal with hardware constraints: due to the very limited memory available, a large amount of game content was procedurally generated on the fly. Nowadays, PCG is instead used either to support the generation of specific contents (e.g., terrains, trees, building, etc) or to provide the players with an *endless* gaming experience.

Recently, PCG attracted a lot of interest also in the game research community since the introduction of the *search-based procedural content generation* (SB-PCG) [1], which combines the procedural content generation methods with a search-based algorithms, such as the genetic algorithms, to automatically generate high-quality game content. So far, SB-PCG proved to be successful in several game genres such as platform games [2]–[4], racing games [5]–[8], RPG games [9], strategy games [10], and others [11], [12].

In this work, we focus on first person shooter (FPS) games, a very popular genre of game where the player controls a character from a first person subjective and has to shoot the opponents. An FPS features two major types of content: the maps where the fight happens and the weapons that players can use. In a previous work, Cardamone et al. [13] applied SB-PCG to evolve maps with an *interesting* gameplay for *Cube 2: Sauerbraten* [3], an open source first person shooter. Later, Lanzi et al. [14] extended the work of Cardamone et al. by applying SB-PCG to the problem of evolving *balanced* maps for FPS when players have different skill levels or are using different weapons. Concerning the weapons, McDuffee

[1] http://en.wikipedia.org/wiki/Beneath_Apple_Manor
[2] http://en.wikipedia.org/wiki/Elite_(video_game)
[3] http://sauerbraten.org

and Pantaleev [15] proposed to apply the SB-PCG to generate weapons in FPS. In particular, they developed *Team Blockhead Wars*, a rather simple multiplayer FPS that features fully customizable weapons through a set of parameters (e.g., damage, rate of fire, ammunition, etc.); thus, a genetic algorithm was applied to evolve new weapons based on the statistics collected from human players: the more a weapon is used by players (both in terms of times and kills performed) the highest would be its fitness.

The approach proposed in this paper is inspired to the work of McDuffee and Pantaleev [15] with three major differences: (i) instead of developing our own prototype of FPS, we extend commercial FPS to enable the procedural generation of weapons; (ii) we propose several metrics to evaluate the evolved weapons, based on game statistics collected through simulations that involve only computer-controlled characters, i.e., our approach does not require human players to evolve weapons; (iii) we test our approach in three different application scenarios.

III. Unreal Tournament III

Unreal Tournament III (UT3) is the last title of the popular series of first person shooters, *Unreal*, developed by Epic Games[4]. UT3 is a competitive multiplayer games but it also provides a sophisticated artificial intelligence to allow playing against computer controlled characters, called *bots*. The game also features several types of games, including *deathmatch*, where each player plays against all the others and *capture the flag*, where two teams fight each other to steal the flag of the opponents. UT3 was developed with the *Unreal Engine*[5], a very popular game engine that was used for developing several commercial games in recent years. The *Unreal Engine* offers several features, including the physics, the rendering, the cameras, etc., and it also allows to modify almost any element of the game logic through a dedicated scripting language called *Unreal Script*. In addition, UT3 is developed with a client-server architecture that allows to run server-side simulation of matches involving bots without the need of rendering the actual game.

IV. Procedural Weapon Generation

In this section we provide the three key elements of our approach. First, we illustrate how we represent the weapons to procedurally generate them. Then, we describe how the generated weapons are tested in UT3 on a simulation basis. Finally, we propose several metrics that can be used to evaluate the generated weapons.

A. Representation

Unreal Tournament III features several weapons that differ for their range, the damage they cause, the amount of ammunition, and the physics of their shots. In addition, most of the aspects that affect the behaviors of the weapons in Unreal Tournament III can be modified through one or more parameters in the game engine. Accordingly, we identified 10 key parameters to represent effectively a weapon in UT3 (see Table I for a detailed description of the parameters along with their range). As a result, we are able to represent a high variety of weapons — including almost all the kind of weapons typically available in FPSs — with a rather compact representation consisting of just a 10 real-valued parameters vector.

B. Simulation

To test the generated weapons, we followed a rather straightforward simulation-based approach. Using *Unreal Script*, we developed a custom UT3 server based on the *Deathmatch* game mode. Our custom server receives as input (i) the two weapons to test[6], each one defined by a parameters vector (according to Table I), (ii) a *score limit*, and (iii) a *time limit*. Thus, a match between two identical bots (with the highest skill level) is created; two UT3 weapons are procedurally generated using the parameters received as input and each one of them is assigned to one of the two bots; then, the match between the two bots starts and ends as soon as either the sum of the scores of the two bots reaches the *score limit* or the *time limit* is exceeded. At the end, the server return a large set of game statistics for each bot (hence for each weapon), which include the kills performed, the deaths, the longest kill streak (i.e., the longest sequence of consecutive kills between two deaths), and some statistics about the shooting dynamics (e.g., accuracy, average hit distance, average hit time, etc.).

It is important to stress that, as the simulation does not require any rendering, it can be performed at accelerated speed. In particular, the actual simulation speed is, in principle, limited by the power of the machine the server is running on. However, in practice, the simulation speed might affect the behavior of the bots as it might change the frequency with which the AI routines are called. Accordingly, we performed an experimental analysis to identify the highest simulation speed that still lead to reliable results and we decided to set the simulation speed to `8x` with respect to normal speed, i.e., simulation time tics 8 time faster than real time.

Finally, it is worthwhile to note that the artificial intelligence in Unreal Tournament III is able to dynamically adapt to custom weapons; in fact, when a new weapon is available, the artificial intelligence performs an analysis of its parameters to devise how it should be used. First of all, the weapon parameters, such as the *speed*, the *life span*, and the *gravity*, are used to adjust the aim of the bot and to compute the *weapon range*, i.e., the optimal distance from the opponent to use the weapon. In addition, based on the values of parameters some specific behaviors are associated to the weapon: if the *rate of fire* of the weapon is above a given threshold (i.e., 2 shots per second), the weapon is associated to the *fast repeater* behavior, i.e., the weapon can be used for long sequences of shots; if the computed *weapon range* is above a given threshold (i.e., 2000 UU), a *sniping* behavior is associated to the weapon, i.e., the weapon can be used as a sniper rifle.

C. Evaluation

When it comes at designing a set of weapons, the most important goal is to make them fun to play. Unfortunately,

[4]http://epicgames.com

[5]https://www.unrealengine.com

[6]In this work we focus on the generation of pairs of weapons, however the simulation-based approach described here could be easily extend to a larger number of weapons to test.

TABLE I. THE WEAPON'S PARAMETERS.

Name	Range	Description
Rate of Fire	$[0, 4]$	Shooting frequency of the weapon: number of bullets shot per second.
Spread	$[0, 3]$	This parameter affects the random deviation of the bullets trajectory: the higher the spread the less accurate would be the shooting.
Shot Cost	$[1, 9]$	Number of bullets shot at once by the weapon.
Life Span	$[0, 100]$	Amount of time the bullets remain in game when shot.
Speed	$[0, 1000]$	Speed of the bullets when shot.
Damage	$[0, 100]$	The amount of damage that each bullet deals when it hits an opponent; the highest damage, 100, also corresponds to the highest player's health value in UT3 (excluding temporary effect of power-ups).
Damage Radius	$[0, 100]$	Radius of the bullets shot by the weapon (in UT3, bullets are logically represented as spheres for computing damages and collisions).
Gravity	$[-250, 250]$	Gravity force applied to bullets shot by the weapon: the larger the value of this parameter, the stronger will be the gravity force that affects each bullet shot by the weapon; negative values means that gravity force will be upside-down, i.e., when shot bullets will go upward.
Explosive	$[0, 300]$	When a bullet hits a target, either an opponent or any object in the game, it generates an explosion; this parameter defines how big is the radius of this explosion; all the players that falls within the radius of such explosion would receive a *splash damage*, i.e., a fraction of the weapon's damage depending on the distance from the center of the explosion.
Ammo	$[0, 999]$	Maximum amount of ammunition for the weapon that a player can carry around.

players' fun might depend on several factors that are rather difficult to evaluate quantitatively, e.g., elements related to the aesthetics, humor, style, theme, and narrative of the game. Nevertheless, there are also several measurable gameplay features that are important to design a good set of weapons. Accordingly, in this section we propose and describe few metrics that can be used to evaluate a pair of procedurally generated weapons, based on the game statistics collected through a simulation.

Balance. To evaluate how balanced a pair of weapons is, we simply considered the distribution of the kills performed by the bots during the simulation. Accordingly, we measure balance as the entropy [16] of the kills distribution:

$$f_{balance} = -\sum_i \frac{k_i}{K} \cdot \log_2 \frac{k_i}{K}, \tag{1}$$

where k_i is the number of kills performed by the i-th bot and K is the overall number of kills performed by the two bots during the simulation. According to the definition above, the smallest value of $f_{balance}$ is 0 — all the kills are performed by one bot — and the largest one is 1 — the two bots performed the same number of kills.

Effectiveness. This metric aims at evaluating whether the evolved weapons can effectively kill the opponent and it is computed as,

$$f_{effectiveness} = \frac{K}{K^*}, \tag{2}$$

where K is the number of kills achieved by the two bots during the simulation while K^* is the highest number of kills that can be reached before the simulation ends; accordingly, $f_{effectiveness}$ is 1 when the bots perform K^* kills and linearly decreases along with the number of kills performed by the bots during the simulation.

Safety. As weapons in UT3 might cause damages also through the explosion of projectiles, it is possible for a player to kill himself by mistake. Therefore, this metric measures how *safe* the generated weapons are (i.e., how likely they can cause the self-deaths of the players) as follows:

$$f_{safety} = 0.9^{D-K}, \tag{3}$$

where D is the overall number of deaths during the simulation and K is the overall number of kills performed of the bots. As a result, f_{safety} is 1 when no self deaths happen during the simulation (i.e., D is equal to K) and then exponentially decreases as the number of self deaths increases.

Gameplay Goals. It is also possible to define several metrics that are based on specific game statistics collected during the simulation. As an example, in our work we considered three different design objectives:

- the average hit distance, $\overline{hit}_{i,dist}$, that is the average distance from the opponent computed from all the hits performed by i-th bot;
- he average hit time, $\overline{hit}_{i,time}$, that is the average time between a shot and the corresponding hit, computed from all the hits performed by i-th bot;
- the longest $killstreak$, $k_{i,MAX}$, that is longest sequence of kills performed by the i-th both between two deaths.

V. EVOLVING BALANCED WEAPONS

In this first application scenario, our aim is to show that SB-PCG can generate pairs of weapons that are balanced, fun to play, and with a good degree of variety. Indeed, balance is one of the most important design feature for weapons in FPS: ideally, each weapon (if used properly) should give the player the same chance to defeat the opponent. Concerning the fun, we cannot directly measure it, but we want to make sure at least that generated weapons are *effective* and *safe*.

Experimental design. We applied a single objective genetic algorithm to evolve a pair of weapons. In this problem, each individual consists of a vector of 20 parameters that encode a pair of weapons (Table I). We performed 10 runs with the following parameters settings: the number of generation was set to 50 and the size of population was 100; to select individuals we applied a tournament selection, with tournament size of 3; finally, we used the simulated binary crossover, with probability 0.6, and the simulated binary mutation, with probability 0.05. Individuals have been evaluated with a simulation on the *DM-Biohazard* map, a popular and pretty small map, involving two identical bots with the highest skill level; for the simulations, the *score limit* was set to 20 and *time limit* was set to 1200 seconds of simulated time.

To solve this problem, we designed the fitness function as follows:

$$F = f_{balance} + f_{effectiveness} + f_{safety}, \tag{4}$$

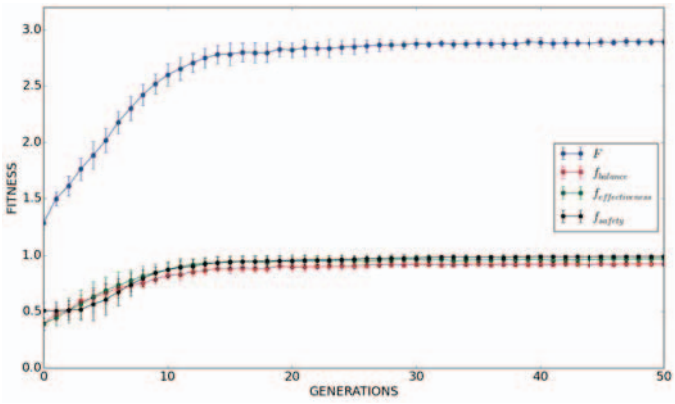

Fig. 1. Average fitness of the pairs of weapons evolved: blue line is the average fitness of the population for each generation; red line (*balance*), green line (*effectiveness*), and black line (*safety*) are the average value of the three components of the fitness function (see Equation 4) computed for each generation.

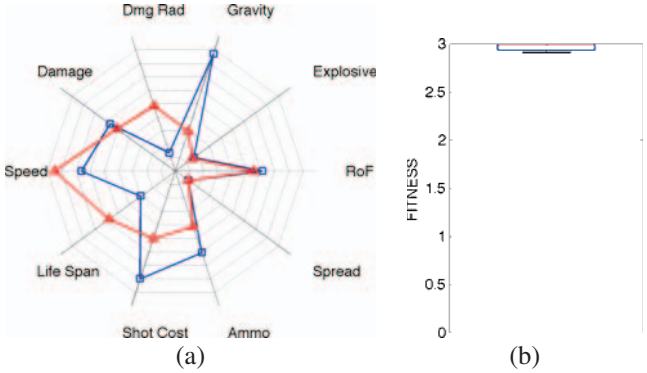

Fig. 2. An example of cluster discovered with DBSCAN applied to the pairs of weapons evolved in the experiment described in Section V: (a) is a radar plot that shows the average value of the weapons parameters in the cluster (red line depicts the parameters of the first weapon of the pair, while blue line depicts the parameters of the second one); (b) shows the distribution of the fitness function of the pair of weapons in the cluster.

where the range of F is between 0 and 3. Please notice that we did not included in F any term to encourage the diversity between the weapons in each pair evolved, as we expect such diversity to be an emergent result of our approach.

All the runs have been performed with DEAP [17], a python evolutionary library.

Results. Figure 1 shows the average fitness of the population during the generation (blue line in Fig. 1) as well as the averages of all the three components of the fitness function F, $f_{balance}$ (red line in 1), $f_{effectiveness}$ (green line in 1), and f_{safety} (black line in 1). In the early generations the average fitness of the population falls in the middle of the range, but it quickly increases after few generations and reaches almost the maximum after 30 generations. It is also interesting to see that the three components of the fitness function behaves in a very similar way, suggesting that none of the corresponding design problems is easier to solve than the others.

Analysis of evolved weapons. In order to get a better insight about the solutions discovered, we performed a clustering of all the final populations of the 10 runs performed. To this purpose, we used the *Density-Based Spatial Clustering of Application with Noise* (DBSCAN) [18], a well known clustering algorithm based on the density of points in the clustering space. Accordingly, we run the implementation of DBSCAN available in the Python *scikit-learn* library [19] with the following parameters setting: $\varepsilon = 0.2$ and $minPts = 5$, i.e., the lowest number of points required to identify a cluster is 5. The clustering algorithm discovered 29 different clusters, each one representing a different way to the design a pair of balanced and effective weapons. Figure 2 and Figure 3 illustrate two examples of the clusters identified with DBSCAN. The first example of cluster (Figure 2) includes six pairs of weapons. The fitness distribution of these six pairs of weapons (Figure 2b) suggests that they are all very well balanced with a fitness very close to the maximum value; nevertheless, when we look at the average values of the parameters (Figure 2a), the two weapons evolved in each pair are quite different among them. In particular, the first weapon (red line with triangles in Figure 2a) is a mid-range rifle: high *Speed* and decent *Life Span* allows to shoot the opponent from a medium/long distance; at the same time, low *Spread* and good *Rate of Fire* make the weapon quite effective. In contrast, the second weapon (blue line with boxes in Figure 2a) is a medium-low range weapon, as it has a moderate value of *Speed* and a very high value of *Gravity*; however, it is very powerful because it shots several bullets at once (high *Shot Cost*), it is very accurate (low *Spread*), and quite deadly (good *Damage*). Therefore, the two weapons require rather different playing strategies but they both have strengths and weaknesses, resulting in a quite good balanced match.

The second example of cluster (Figure 3) features two weapons completely different from the ones in the previous example. The first weapon (red line with triangles in Figure 3a) is a sort of sniper rifle with low *Shot Cost* and *Spread* but high *Damage*, *Speed*, and *Life Span*. The second weapon (blue line with boxes in Figure 3a) is a weapon that cannot be used from a long distance (medium *Speed* and *Life Span*); however, on medium-low range it might be an extremely powerful weapon, thanks to the low *Spread*, the very high value of *Shot Cost*, and moderate value of *Damage*. As in the previous example, the two weapons are very different but balanced as suggests the fitness distribution in Figure 3b. Overall, it is interesting to stress how the evolutionary design of the weapons was able to identify couple of weapons that are, at the same time, balanced and very different among them.

VI. Tuning Weapons

One of the most popular weapons of UT3 is the *Flak Cannon*, a close-combat weapon that fires several bullets at once. A well-known design problem of the *Flak Cannon* is that it is generally outperformed by the *Rocket Launcher* — the most popular weapon in UT3 — with few exceptions. As a second application scenario, we apply SB-PCG to generate weapons well balanced against the *Rocket Launcher* by making the least possible changes to the original *Flak Cannon*. Indeed, this is a very typical weapon design problem and we believe that SB-PCG can be a promising technique to deal with it.

Experimental design. To solve this problem, we designed it as a multi-objective problem, where we have to evolve a weapon that is (i) *effective*, *safe*, and *balanced* against the *Rocket Launcher*, and (ii) as much similar as possible to the *Flak*

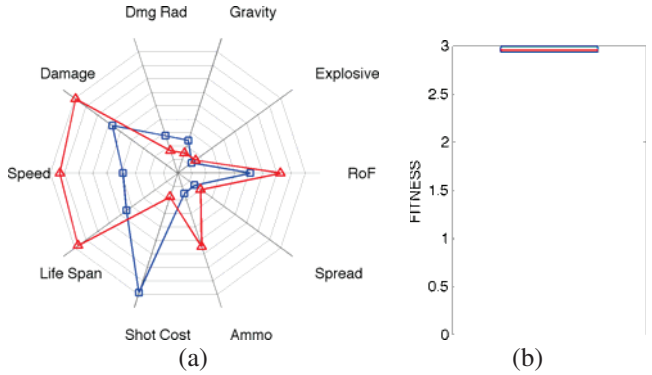

Fig. 3. An example of cluster discovered with DBSCAN applied to the pairs of weapons evolved in the experiment described in Section V: (a) is a radar plot that shows the average value of the weapons parameters in the cluster (red line depicts the parameters of the first weapon of the pair, while blue line depicts the parameters of the second one); (b) shows the distribution of the fitness function of the pair of weapons in the cluster.

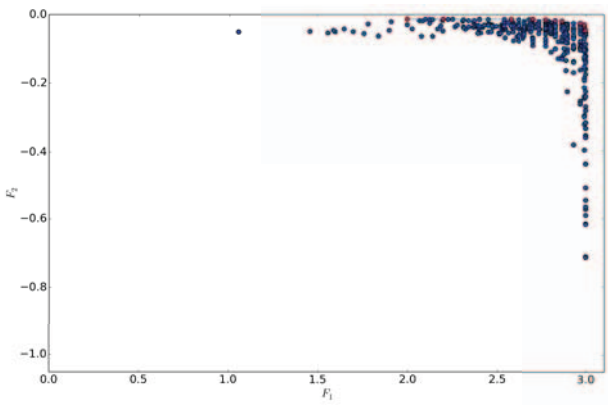

Fig. 4. Objective values of the weapons in the final populations of all the runs performed in the experiment described in Section VI. Red points identifies the weapons on the pareto-front.

Cannon. Accordingly, we used the *NSGA-II* [20], a well known multi-objective genetic algorithm, to perform 10 runs with the following parameters settings: the number of generation was set to 50 and the size of population was 50; each individual is a 10 parameters vector representing a single weapon; to select individuals we applied a tournament selection, with tournament size of 2; finally, we used the simulated binary crossover, with probability 0.9, and the simulated binary mutation, with probability 0.1. As in the previous experiment, we evaluated the individuals based on a simulated match in the *DM-Biohazard* map between two identical bots with the highest skill level; the *score limit* was set to 20 and *time limit* was set to 1200 seconds of simulated time. In such simulated match, one bot is using the *Rocket Launcher* and the other bot is using the evolved weapon to evaluate. The objectives of the evolved weapons are computed as:

$$F_1 = f_{balance} + f_{effectiveness} + f_{safety}, \tag{5a}$$

$$F_2 = -\sqrt{\sum_i (x_i - x_i^*)}, \tag{5b}$$

where x_i and x_i^* are respectively the parameters of the weapon to evaluate and of the *Flak Cannon* (both x_i and x_i^* are normalized, so that each parameter has the same weight).

Results. Figure 4 shows the final populations of the 10 runs performed; each weapon that is in the final population of a run is represented as a blue point in the space of the fitness objectives; the weapons that are in the pareto-front are represented with red points. The results show that evolution was able to generate several weapons that provides a very good trade-off between the two objectives. It is also worthwhile to note that the pareto-front has a very flat slope: the first objective reaches the maximum value with just a small decrease of the second objective. Accordingly, the results suggest that, at least in this case, it is possible to adjust the balancing of the *Flak Cannon* against the *Rocket Launcher* with just few changes to the parameters of the *Flak Cannon*.

Analysis of evolved weapons. As in the previous experiment, we clustered the evolved weapons in the final populations (Figure 4) of the 10 runs (consisting overall of 500 weapons) to have a better insight about the evolved solutions. We applied the *DBSCAN* algorithm with $\varepsilon = 0.05$[7] and $minPts = 5$. As a result, the clustering process identified 8 different clusters in the population. The largest cluster consisted of 374 individuals and included all the weapons that are very similar to the *Flak Cannon* (average value of F_2 is 0.05); on the other hand, they are not extremely balanced against *Rocket Launcher* (average value of F_1 is 2.69). Figure 5 shows a more interesting cluster of 6 weapons that despite being rather similar to the the *Flak Cannon* (see Figure 5a) are better balanced against *Rocket Launcher* as shown in Figure 5b. In particular, looking at the weapons parameters (Figure 5a) we can see that evolved weapons have a larger *Rate of Fire* and a slightly larger *Ammo*, but all the other parameters are basically the same of original *Flak Cannon*; nevertheless the new parameters allow for a better balancing (as shown by the distribution of F_1 in Figure 5b).

VII. Multi-Objective Design

In this last scenario, we apply SB-PCG to evolve weapons with some specific design objectives, such as encouraging long range combat or fast-paced action. To test our approach, we performed several experiments with different combinations of three design objectives: long range, kill streaks, and slow hit time. However, due to the lack of space, in this paper we present a single experiment that focuses on evolving a pair of weapons, so that using the first weapon would result in long distance kills, while using the second one would result in long killing streaks; in addition, the pair of evolved weapons should be well balanced, effective and safe to use.

Experimental design. We applied the *NSGA-II* [20] to evolve a pair of weapons using the following objectives:

$$F_1 = f_{balance} + f_{effectiveness} + f_{safety}, \tag{6a}$$

$$F_2 = \overline{hit}_{1,dist}, \tag{6b}$$

$$F_3 = k_{2,MAX}, \tag{6c}$$

[7] We had to reduce the value of ε from 0.2 to 0.05, because the similarity objective leads to several individuals very close in the parameters space.

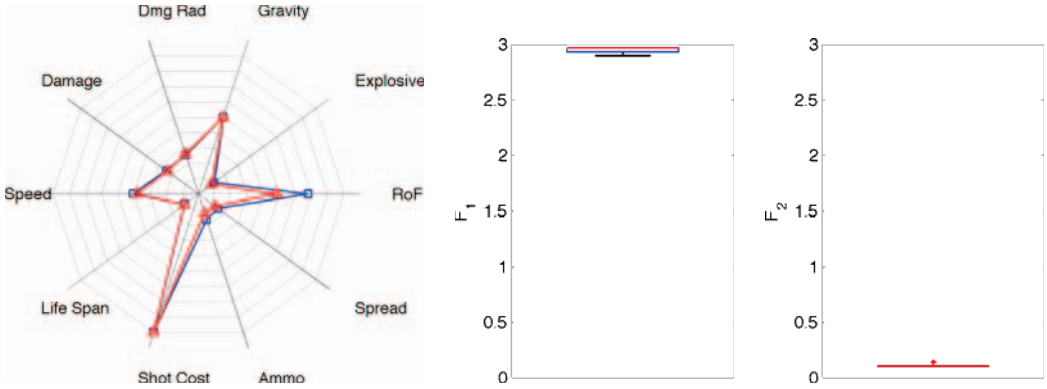

Fig. 5. An example of cluster discovered with DBSCAN applied to the weapons evolved in the experiment described in Section VI: (a) is a radar plot that shows the average value of the weapons parameters in the cluster (red line depicts the original parameters of the *Flak Cannon*, while blue line depicts the parameters of the evolved one); (b) and (c) shows the distribution of the two objectives of the weapons in the cluster.

where F_1 is the same objective used in previous experiments, F_2 is the average hit distance of the first weapon (see Section IV-C), and F_3 is the longest killing streak achieved using the second weapon (see Section IV-C).

The experiment consisted of 10 runs with the following parameters settings: the number of generation was set to 50 and the size of population was 100; each individual is a 20 parameters vector representing a pair of weapons; to select individuals we applied a tournament selection, with tournament size of 2; finally, we used the simulated binary crossover, with probability 0.9, and the simulated binary mutation, with probability 0.05. As in the previous experiments, we evaluated the individuals based on a simulated match in the *DM-Biohazard* map between two identical bots with the highest skill level; the *score limit* was set to 20 and *time limit* was set to 1200 seconds of simulated time.

Results. Figure 6 shows the objective values for all the pairs of weapons in the final population of the 10 runs performed. To investigate the trade-off between the three design objectives, the final population is represented in three different two-dimensional spaces, one for each possible pair of the three objectives; each pair of weapons is represented either as a blue point or as red point if it belongs to the pareto-front. In particular, Figure 6a shows clear trade-off between objective F_1, i.e., balancing, effectiveness and safety of the weapons, and objective F_2, i.e., the average hit distance of the first weapon: the longer is the distance from which the first weapon can be effectively used, the more difficult is to have a balanced pair of weapons; this is not surprising as it is very difficult to design a well balanced long-range weapon both in terms of pace and effectiveness. Figure 6b shows a similar result when it comes at considering the trade-off between the objective F_1 and objective F_3, i.e., the longest killing streak of the second weapon; in fact, although the pareto-front is less steep than the previous one, it appears that a longer killing streak leads to a worse balancing between the two weapons; again, this is not surprising as it is rather obvious that a long killing streak would necessary results in a clear advantage for a weapon — thus, in a worse balancing. Finally, Figure 6c shows that we have a trade-off also between the two design objectives F_2 and F_3: when evolving a pair of weapons, increasing the average hit distance of the first one leads to decreasing the longest killing streak achieved by the second one. In addition, results also suggests that, overall, designing a long-range weapon, i.e., maximizing F_2, is perhaps the most difficult optimization problem tackled by the evolutionary algorithm; in fact, Figure 6 shows that while evolution found several solutions with high values of objectives F_1 and F_3, there are only few solutions that have a high value of objective F_2.

Analysis of evolved weapons. As in the previous experiments, we clustered the 1000 pairs of weapons in the final populations of the 10 runs performed; the DBSCAN algorithm (with parameters $\varepsilon = 0.2$ and $minPts = 5$) identified 37 clusters. Figure 7 shows an example of the clusters identified, consisting of 8 pairs of weapons with the following average objective values, $F_1 = 2.53$, $F_2 = 1216.8$, and $F_3 = 6.2$ (see the distribution of the objective values in Figure 7b–d). In particular Figure 7a shows the average parameters value of the pair of weapons included in the cluster; the first weapon (red line with triangle) has larger values in *Speed* and *Life Span* parameters, resulting more suited than the second weapon when used from a long distance. In addition, it has a good *Rate of Fire*, a very small *Spread* and very high *Damage*; however, it also has a limited value of *Ammo*, forcing the player to find additional ammunition rather frequently. The second weapon, in contrast, appears less powerful than the first one in several respects, except for the very large *Damage Radius* and a better value of *Ammo*. Accordingly, we could expect the first weapon to behave much better in long range fights but, at the same time, the second weapon to be rather dangerous when used from short distances.

VIII. User Study

Finally, we performed a preliminary user study to assess the capabilities of our approach to evolve a couple of balanced and enjoyable weapons. The study was carried out during an Open Day at our university and involved 35 distinct users (18-22 years old student, mostly male). Each user was asked to play a match against a bot with a score limit of 10 points and time limit of 10 minutes. During the match, the player can use either a *red* or a *blue* weapons [8]; human player starts with

[8]To make the weapons easy to distinguish for the human players, we used blue and red textures to actually color the weapons in game.

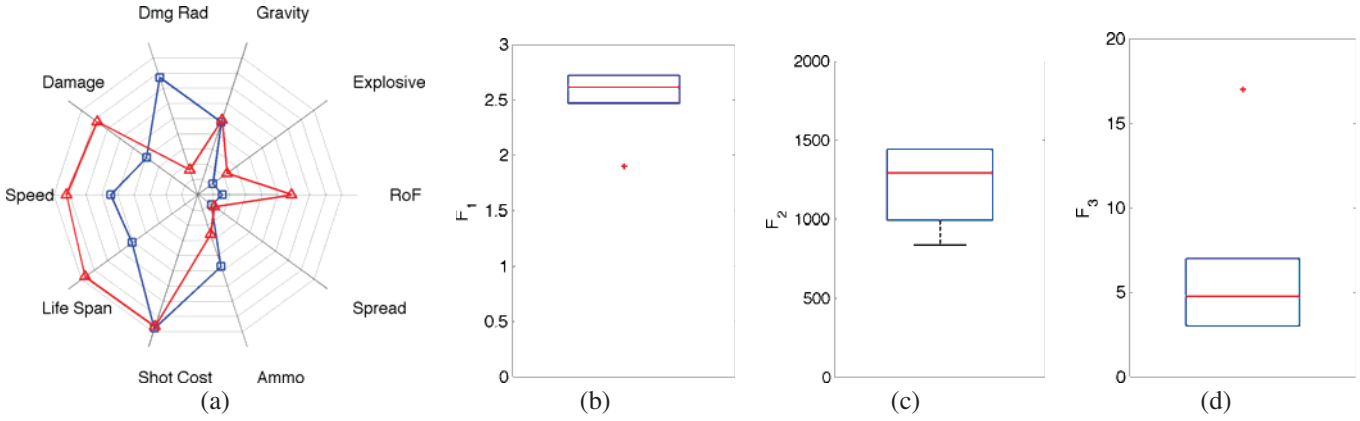

Fig. 7. An example of cluster discovered with DBSCAN applied to the pairs of weapons evolved in the experiment described in Section VII: (a) is a radar plot that shows the average value of the weapons parameters in the cluster (red line depicts the parameters of the first weapon of the pair, while blue line depicts the parameters of the second one); (b)–(d) shows the distribution of the three objectives of the pairs of the weapons included in the cluster.

the *blue* weapon, while the bot starts with the *red* one; during the match, the player can switch the weapon anytime; as soon as the player switches the weapon, the bot does the same, so that the player and the bot are never using the same weapon. The *red* and *blue* weapons were selected by alternating among two pairs of weapons from the best ones evolved for the first scenario presented in this paper (see Section V). At the end of the match, we collected some information about the users (e.g., age, gender, gaming experience, etc.) and asked them the following two questions for both the *red* and the *blue* weapons:

Q1. Rate your experience in terms of fun when using the *blue* [*red*] weapon (using a {Low, Medium, High} scale).

Q2. Rate how powerful is the *blue* [*red*] weapon with respect to the other one (using a {Under-Powered, Balanced, Over-Powered} scale).

Figure 8 shows the answers collected from the users. These results suggest that, in general, the users enjoyed playing the evolved weapons; in fact, overall only the 8.6% of the collected ratings were *Low*, while 51.4% were *Medium* and 40% were *High*. Concerning the balance, about half of the users evaluated the weapons balanced (51%), while the others found one weapon more or less powerful than the other one. We investigated better this issue and discovered that the red weapon in the second pair was perceived not as balanced as the other ones. In particular, the comments left by the users suggested that, when using this weapon, it was too difficult to hit the opponent. In fact, this weapon features a large *Gravity*, making it quite difficult for human player to find a proper trajectory to hit the opponents. Of course, this issue could not be discovered in our simulations as computing a proper trajectory is not a problem for the bots.

IX. Conclusion

In this paper, we applied search-based procedural content generation (SB-PCG) to evolve weapons for *Unreal Tournament III* (UT3), a popular commercial first person shooter. We represented each weapon as a vector of parameters and extended UT3, so that all the generated weapons can be actually used and tested in game. We described several metrics to evaluate the generated weapons and we also implemented a simulation-based methods to compute them. Then, we identified three interesting applications to test our approach: (i) we applied SB-PCG to evolve a pair of effective and balanced weapons; (ii) we applied SB-PCG to improve the balancing of an existing weapon; (iii) we applied SB-PCG to evolve a pair of weapons with different design objectives. Our results suggest that SB-PCG can be effectively applied to solve these design problems and might be a promising technique to assist developers with the weapons design process. Finally, we performed a test with more than 30 users to assess the quality of the weapons generated. The feedback received from the users show that SB-PCG can actually generate weapons that are interesting, fun to play and rather balanced. However, the simulation-based approach used to evaluate the evolved weapons might require some tweaks to take into account some issues that are specific to human players, such as the complexity of using a weapon.

Future works include improving the simulation-based evaluation of the weapons, testing the approach on different first person shooters, and performing additional studies with human players.

References

[1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proceedings of EvoApplications*, vol. 6024. Springer LNCS, 2010.

[2] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Optimization of platform game levels for player experience," in *Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2009, October 14-16, 2009, Stanford, California, USA*, C. Darken and G. M. Youngblood, Eds. The AAAI Press, 2009.

[3] K. Compton and M. Mateas, "Procedural level design for platform games," in *Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference, June 20-23, 2006, Marina del Rey, California*, J. E. Laird and J. Schaeffer, Eds. The AAAI Press, 2006, pp. 109–111.

[4] N. Shaker, G. N. Yannakakis, and J. Togelius, "Towards automatic personalized content generation for platform games," in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, October 11-13, 2010, Stanford, California, USA*, G. M. Youngblood and V. Bulitko, Eds. The AAAI Press, 2010.

[5] J. Togelius, R. De Nardi, and S. Lucas, "Towards automatic personalised content creation for racing games," in *Proc. IEEE Symposium on Computational Intelligence and Games CIG 2007*, 2007, pp. 252–259.

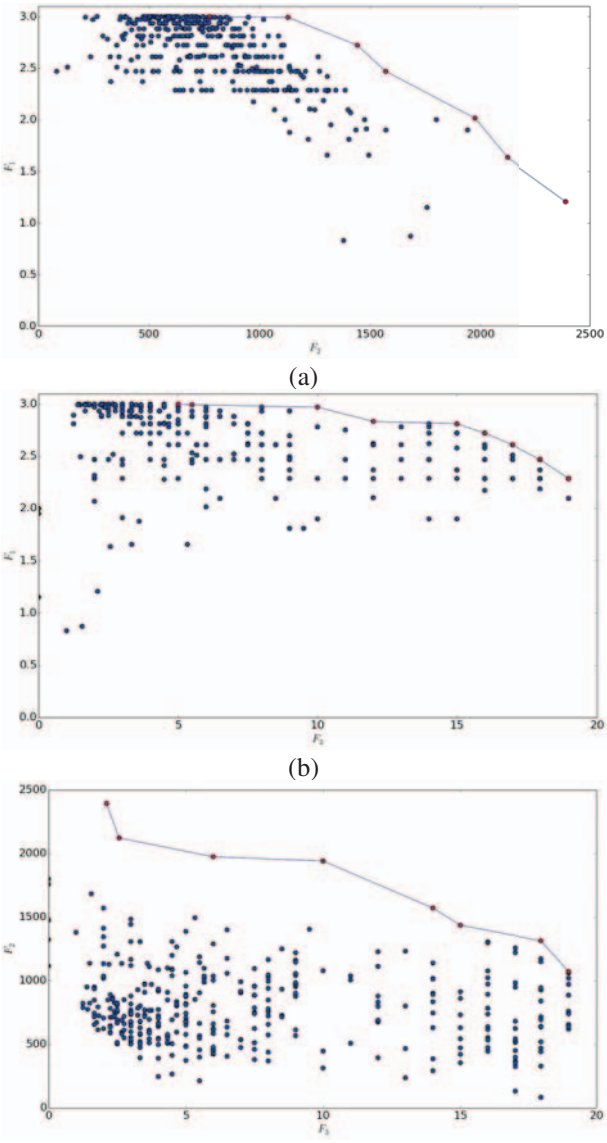

Fig. 6. Objective values of the pair of weapons in the final populations of all the runs performed in the experiment described in Section VII: (a) shows the first objective (y-axis) and the second objective (x-axis); (b) shows the first objective (y-axis) and the third objective (x-axis); (c) shows the second objective (y-axis) and the third objective (x-axis). Red points identifies the weapons on the pareto-front.

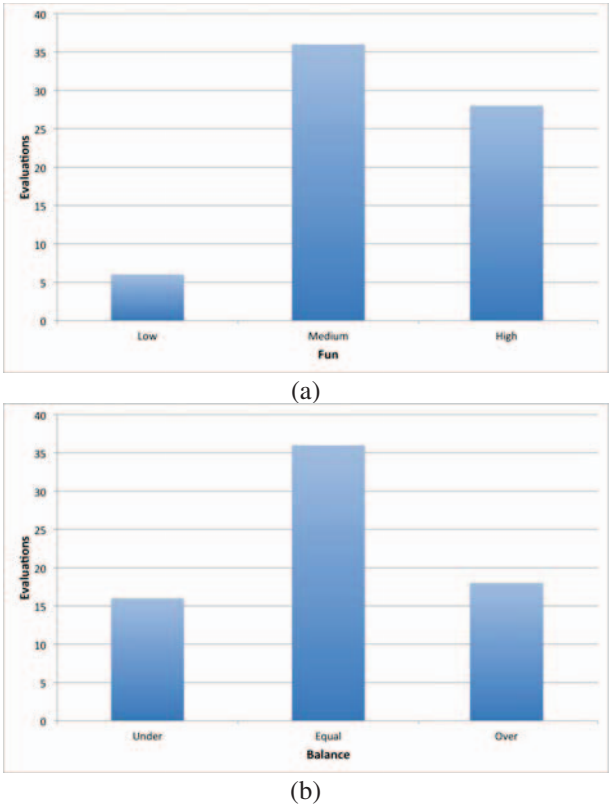

Fig. 8. Answers provided by the users to Q1 (a) and to Q2 (b).

[6] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Automatic track generation for high-end racing games using evolutionary computation," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 3, pp. 245 –259, sept. 2011.

[7] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Interactive evolution for the procedural generation of tracks in a high-end racing game," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ser. GECCO '11. New York, NY, USA: ACM, 2011, pp. 395–402.

[8] L. Cardamone, P. L. Lanzi, and D. Loiacono, "Trackgen: An interactive track generator for TORCS and speed-dreams," *Appl. Soft Comput.*, vol. 28, pp. 550–558, 2015.

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

[10] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the starcraft map space," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010, pp. 265–272.

[11] E. J. Hastings, R. K. Guha, , and K. O. Stanley, "Automatic content generation in the galactic arms race video game," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 245–263, 2009.

[12] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley, "Combining search-based procedural content generation and social gaming in the petalz video game," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, October 8-12, 2012*, M. Riedl and G. Sukthankar, Eds. The AAAI Press, 2012.

[13] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I*, ser. EvoApplications'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 63–72.

[14] P. L. Lanzi, D. Loiacono, and R. Stucchi, "Evolving maps for match balancing in first person shooters," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26-29, 2014*. IEEE, 2014, pp. 1–8.

[15] E. McDuffee and A. Pantaleev, "Team blockhead wars: Generating fps weapons in a multiplayer environment," in *FDG PCG workshop*, 2013.

[16] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal, The*, vol. 27, no. 3, pp. 379–423, July 1948.

[17] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[18] M. Ester, H. peter Kriegel, J. S, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." AAAI Press, 1996, pp. 226–231.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2000.