Evolving Maps for Match Balancing in First Person Shooters

Pier Luca Lanzi
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Email: pierluca.lanzi@polimi.it

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

Riccardo Stucchi
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Email: riccardo1.stucchi@mail.polimi.it

***Abstract*—Match balancing is one of the most important design issues in the development of an adversarial multiplayer shooter. Therefore, matchmaking algorithms are generally used to build teams, such that players can have fun with each other and enjoy the game experience. In this work we approach this problem from a completely different angle: we show that the design of the game content has a large impact on the match balancing and that procedural content generation might be a promising approach to improve it. In particular, we present a methodology to evolve maps for Cube 2: Saubertran, an open source first person shooter, and to improve the game balancing for specific combinations of players skills and strategies. We tested our approach on three different scenarios that involve players with different skill levels as well as players using completely different weapons. Our results are very promising and show that, in all the scenarios considered, our approach is able to evolve maps that result in a balanced game experience.**

I. Introduction

Nowadays, internet multiplayer is a standard feature in a commercial game and the core of the game experience in several genres, such as massive multiplayer online games, first person shooters, real-time strategy games, etc. In particular, in first person shooters (FPS), the adversarial multiplayer is often the primary component of success (notable examples are the EA's Battlefield [1] and the Activision's Call of Duty [2] series).

Among several technological and design issues that come with the development of an adversarial multiplayer shooter, *match balancing* is one of the most important. In fact, to engage players it is extremely important to provide them with the right amount of challenge [1]: a poor match balancing would easily result in either a boring or a frustrating game experience. A typical approach to this problem consists of implementing a matchmaking algorithm to select players with similar skills when building the teams for a match. Unfortunately, matchmaking algorithms have limited applications when (i) there is a limited or no control at all on the team building (e.g., when teams are decided by players) and (ii) players have very different playing strategies (e.g., when players use different weapons or items).

In this work, we approach the problem of match balancing from a different angle and propose a novel methodology based on the procedural content generation techniques. So far, procedural content generation combined with search-based algorithms [2] has been successfully applied to automatically design game content for several games, but its applications to game balancing have been very limited [3]. In this paper, we aim at showing that procedural content generation might be a promising approach to deal also with this challenging problem. In particular, we applied a genetic algorithm to evolve balanced maps for Cube 2: Saubertran, an open source first person shooter. We followed an approach similar to the previous work of Cardamone et al. [4], where search-based procedural content generation was applied for the first time to design maps for a first person shooter. However, while Cardamone et al. [4] focused on evolving maps that might result in a fast-paced action, in this work we focus on evolving maps that would improve the match balancing for specific combinations of players skills and strategies. We tested our approach on different scenarios that provide a challenge in terms of game balancing because they involve players that (i) have a very different skill level and (ii) are using completely different weapons. Our preliminary experimental results are very promising: in all the scenarios considered, the best maps evolved with our approach lead to rather large improvements in terms of match balancing.

The paper is organized as follows. In Section II we provide an overall picture of the related work and in Section III we briefly describe Cube 2. Then, in Section IV we introduce our approach to evolve maps for match balancing in Cube 2. Finally, experimental results are presented in Section V and we draw conclusions in Section VI.

II. Related Work

This work combines ideas and challenges from three different fields of research: matchmaking, game design patterns, and procedural content generation. In this section we provide an overview of previous works in those fields that are more relevant to our paper.

A. Matchmaking

Matchmaking is the most common approach used to solve the problem of match balancing in multiplayer adversarial games. It basically consists of building the teams of players such that the match would result as balanced as possible. Accordingly, matchmaking requires to assign a skill rating to each player on the basis of his profile and of previous match

[1] http://www.battlefield.com/
[2] http://www.callofduty.com/

978-1-4799-3547-5/14/1.00

results. A well known approach to rank players is the Elo rating system [5], officially used by the World Chess Federation and based on an efficient online update algorithm. Later, Herbrich et al. introduced TrueSkill™ [6], which extended the Elo rating system to deal with uncertainty and to estimate the skill of the single player from team results. More recently, several improvements and variants of TrueSkill™ have been introduced in the literature [7], [8], [9]. In a recent work, Delalleau et al. [10] proposed a novel approach for a strategical first person shooter: they argued that player skill alone is not enough for a successful matchmaking and better result can be achieved by taking advantage of the information available about the players behavior and preferences.

Rather than an alternative to the matchmaking systems, in this work we propose an approach that can be easily combined with a matchmaking system. In fact, our goal is to design the game maps such that the match would result as balanced as possible, independently from the players involved in the match. Accordingly, traditional matchmaking approaches might be extended to include the design of the game map as an additional decision outcome.

B. Game Design Patterns

Balancing a match between players with different playing strategies, is usually considered a design problem: it is up to the designers to make sure that the maps allow players to use effectively different strategies. Despite there are several books in the literature on this topic [11], [12], [13], there is still a limited formal understanding of the map design process. Several authors [14], [15], inspired by the good practices used in software engineering, suggested that identifying and applying game design patterns might improve the level design process. More recently, few works in the literature [16], [17] specifically focused on the study of game design patterns for first person shooters. In particular, Hullet et al. identified and analyzed some of the most important patterns in levels of single-player FPS [16]; later, Giusti et al. investigated the weapon design patterns in first person shooters [17].

The major challenge of this research direction lies on identifying relevant patterns and on understanding how they affect the game dynamics and the match balancing. Nevertheless, a *library* of relevant patterns could be effectively integrated in any approach based on the procedural generation of the game content.

C. Procedural Content Generation

Procedural content generation was originally introduced in early days of game development as a way to overcome the hardware capabilities: due to the very limited memory resources, game content that could not fit into the main memory was procedurally generated on the fly. In contrast, today procedural game content is mainly used either to speedup the content creation process or to design a specific game experience (e.g., non-deterministic level generation, endless games, etc.). Recently, Togelius et al. [2] introduced the *search-based procedural content generation* (SBPCG), 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, SBPCG has been applied to several game genres, including platform games [18], [19], [20], racing games [21], [22], [23], rpg games [24], [25],strategy games [26], and others [27], [28].

Fig. 1. An in-game screenshot of Cube 2: Sauerbraten.

In particular, this work is based on a previous work from Cardamone et al. [4] which applied SBPCG to evolve maps for Cube 2, an open source first person shooter. In particular, in this work we used one of the map encoding proposed in [4] and followed a similar approach to evolve maps. However, while the aim of Cardamone et al. was to evolve *interesting* maps, i.e., maps that allows the emergence of an interesting game dynamics, in this work we focus on evolving map that can improve the match balancing.

Finally, in a recent work Togelius et al. [3] introduced the idea of applying SBPCG to support the design of balanced maps for a strategy game. However, in their work Togelius et al. do not propose a direct approach to evolve balanced maps, but rather they propose a multi-objective approach to generate maps with the desired properties in terms of resource placement, navigation, safety, etc; accordingly, they leave to designer the task of deciding which are the properties of the map that would result in a balanced game experience.

III. CUBE 2

Cube 2: Sauerbraten [29] (Figure 1) is an open source first person shooter that supports multiplayer with several game modes, such as deathmatch, instagib, tactics, domination and capture of the flag. In the game there are seven different weapons available: chainsaw, pistol, machine gun, shotgun, rifle, grenade launcher, and rocket launcher. It is also possible to add computer controlled bots to the match, tuning the performance of each bot through a *skill parameter* that can be set to a value from 0 to 100 and that mainly affects the shooting accuracy of the bot. As commonly happens in first person shooters, the game is developed with a client-server architecture. This allows, when no human player is involved, to run only the server with accelerated speed (up to 100x) and without any visualization of the match. In addition, Cube 2 supports a scripting language for the customization of the user interface as well of many other elements of the game, allowing the creation of modified version of the game (*mods*). Finally, an open source and rather powerful map editor is provided with the game, allowing users to create custom maps.

IV. Evolving Maps for Match Balancing in Cube 2

The maps in Cube 2 are built combining together several structures called *octrees*: each octree is basically a cube that is recursively composed by 8 smaller octrees. The map editor provided with Cube 2 allows to build a map by positioning and combining together several octrees. For each octree positioned on the map, it is possible to choose the size (within a range), the material, the texture; once added to the map, each octree can be also edited either removing any of the 8 octrees it is composed of or moving any vertex of the octrees inward, making it possible to build more complex shapes (e.g., it is possible to create a ramp with the desired slope). In addition, the editor also allows to define the position of other game elements on the map, such as the spawning points, i.e., locations where players will reappear after being killed by an opponent, weapon pickups, i.e., locations where players can pickup a specific type of weapon during the game, and add-ons pickup, i.e., locations where player can find add-ons to boost either armor, ammo for a specific weapon, or health.

In this section we first describe the map encoding we used to evolve maps for Cube 2 and how do we compute the fitness function for the evolved maps.

A. Map Encoding

Despite Cube 2 allows to build maps that contain rather complex structures and that consist of several levels connected with either stairs, ramps or jumps, in this work we follow the approach of Cardamone et al. [4] and focus on a much simpler class of maps. Our maps consist of a single level (or floor) and such level can be ideally represented as a 64×64 matrix of square tiles. Each tile can be either empty or contain a wall of fixed height (built with a suitable composition of octrees). The empty tiles are basically the area of the map the can be navigated by the players, i.e., the rooms, the arenas and the corridors of the maps, and can be also populated with the spawning points, the weapon pick-ups, and the add-ons. The walls cannot be traversed by players (their height is designed such that it is not possible to jump over them) and, thus, allow to organize the maps in separated areas. Despite looking extremely simple, such representation would still lead to a rather large design space consisting of more than 4000 binary variables. Accordingly, we used one of the encoding proposed by Cardamone in [4], where all the tiles of the map are by default a wall and only the empty tiles are encoded, resulting in a much smaller design space. In particular, the genotype encodes the arenas and the corridors in the map as follows: each arena is encoded as a triplet $< x, y, s >$, where x and y define the center of the arena and s is the size of the arena; similarly, each corridor is encoded as a triplet $< x, y, l >$, where x and y define the center of the corridor and l encodes both the length and the direction of the corridor (positive length is used to encode horizontal corridors, while a negative length is used to encode a vertical corridor); all the arenas are squared and corridor have a fixed width of three tiles. As a result, the genotype is a vector of triplets that encodes n_a arenas and n_c corridors (in all the experiments reported in this work $n_a = 15$ and $n_c = 15$). In order to create a complete map of Cube 2 from the genotype described above, two additional steps are necessary. First, we remove all the empty tiles that are not reachable from the center of the map, i.e., we implement a sort of simple repair mechanism to make sure that the resulting map is fully connected. Second, we add spawning points, weapon pick-ups, and add-ons to the map with a simple heuristics that attempts to distribute uniformly them on the map.

B. Fitness Function

Evolved maps are evaluated on the basis of statistics collected from a two-players [3] match using the following fitness function:

$$f = -\sum_{i=1}^{2} \frac{S_i}{S_1 + S_2} \log_2 \left(\frac{S_i}{S_1 + S_2} \right), \tag{1}$$

where S_i is the score of the $\texttt{player}_i$ in the match, that is basically the number of kills it performed subtracted by the number of self-deaths[4]. Therefore $S_i/(S_1 + S_2)$ represents the *score ratio* of $\texttt{player}_i$ ($\texttt{SR}_i$ in brief), i.e., how many points the $\texttt{player}_i$ scored with the respect to the overall points scored by both the players during the match: when $\texttt{SR}_i$ is 0, $\texttt{player}_i$ did not score any point in the match; when $\texttt{SR}_i$ is 0.5, the match is perfectly balanced, i.e., $\texttt{player}_1$ and $\texttt{player}_2$ scored exactly the same; when $\texttt{SR}_i$ is 1, $\texttt{player}_i$ scored all the points of the match, i.e., its opponent did not score any point. Therefore the entropy of the *score ratio distribution* computed by Equation 1 is a measure of the match balancing: the more $\texttt{SR}_1$ and $\texttt{SR}_2$ are close to 0.5, the higher (the more close to 1) will be the fitness f.

In general, it is rather tricky to define a threshold of the fitness function above which it is possible to consider the match balanced. However, we can still identify a desirable range for $\texttt{SR}_i$: when both $\texttt{SR}_1$ and $\texttt{SR}_2$ are in a range comprised between 34% and 66% (roughly corresponding to $f > 0.92$), the most skilled player would perform a number of kills that is more than twice the number of kills of the other player. Accordingly, this range guarantees that the least skilled player would not die more than twice for every kills he performs and, conversely, the more skilled player would die at least once every two kills. As a result, the least skilled player would still feel being part of the action and the most skilled one would not feel invincible.

As done by Cardamone et al. [4], we run a simulated match with bots to compute the fitness function. Specifically, in all the experiments reported in this work, the fitness function is computed applying the Equation 1 to the statistics collected from a 10 minutes match between two bots on the map to evaluate. Using bots to compute the fitness function allows a fast evaluation of the maps (simulated times can be accelerated up to 100X) and several possibility of customization: we can evolve maps for a specific balancing scenario, by choosing the skill values of the bots as well as the weapons they can use during the match. On the other hand, computing the fitness function using bots results in an approximation of the complex behaviors of human players.

[3] In this work we focused on balancing two-players matches. However the approach and the fitness function can be easily adapted to a larger number of players.

[4] When using some weapons, such as grenade launcher or rocket launcher, a player can cause its own death due to the resulting explosions.

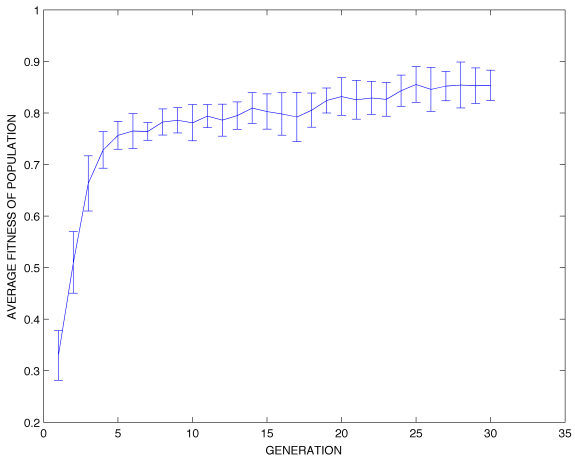


Fig. 2. Evolving maps for balancing the match between $\texttt{bot}_1$ with skill 80 and $\texttt{bot}_2$ with skill 35 (both using a rifle). Average fitness of the map evolved during the generations. Curves are averages over 10 runs (bars represent the standard deviation).

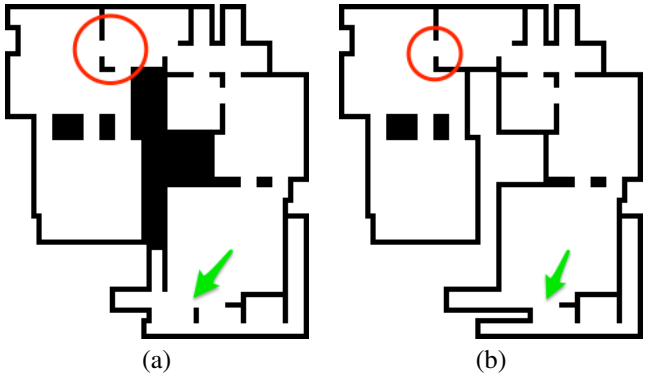


Fig. 4. Two examples of the maps evolved for balancing the match between two bots using the rifle: (a) $\texttt{map}_A$ ($f = 0.92$) and (b) $\texttt{map}_B$ ($f = 0.96$); the red circles and green arrows are used to highlight the main differences between the maps.

V. Experimental Results

To test our approach, we designed three set of experiments. In the first set of experiments we evolved maps for balancing the match between two bots with different skill values but using the same weapon. In the second set of experiments, we focused on balancing the match between two bots with different skill levels *and* using different weapons.

All the experiments reported in this paper has been performed using the C++ Genetic Algorithm Toolbox [30].

A. Scenario 1: Rifle vs Rifle

The aim of the first set of experiments is to test our approach to improve the balancing of a match between two bots with a different skill levels but using the same weapon. Specifically, we focused on bots that use the *rifle* and have a skill value of respectively 80 and 35. We chose the rifle because its performance as a weapon is heavily affected by the skill value of the bots. The skill values of the bots (80 and 35), have been selected on the basis of a preliminary analysis

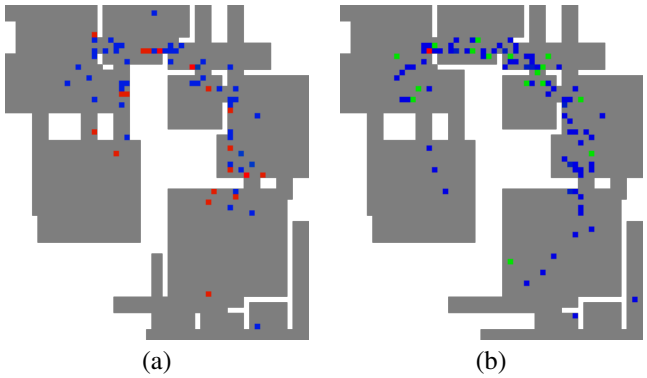


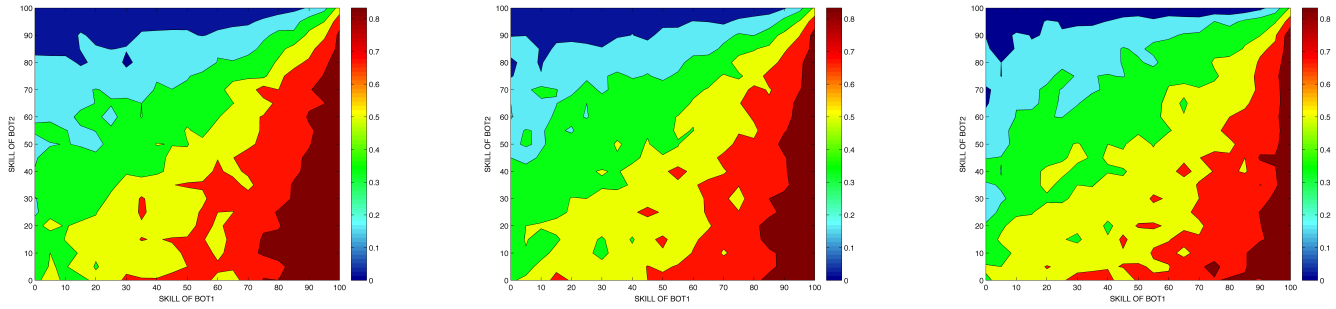
Fig. 5. Distribution of the more frequent positions from which $\texttt{bot}_2$ kills $\texttt{bot}_1$ on (a) $\texttt{map}_A$ (Figure 4a) and (b) $\texttt{map}_B$ (Figure 4b). The frequency of kills from each position is coded by the color from blue, the least frequent positions, to red, the most frequent positions; areas in gray are the positions from which there are no kills. Statistics are collected from 5 matches of 10 minutes each.

(not reported here due to the lack of space) so that the resulting balancing problem would be challenging but feasible.

To search for the best maps in the design space, we performed 10 runs of a simple genetic algorithm with the following parameter settings. Each run of the GA consisted of 30 generations; the population size was set to 100; we used tournament selection with tournament size 2; uniform mutation is applied with a probability 1/n, where n is the length of the genome; matrix crossover is applied with probability 0.2; for each candidate map, the fitness function was computed according to the Equation 1, running a 10 minutes match involving two bots: (i) $\texttt{bot}_1$ with skill value of 80 and using the rifle; (ii) $\texttt{bot}_2$ with skill value of 35 and using the rifle.

Figure 2 shows the average fitness of the evolved maps during the generations. In the early generations, the average fitness of the maps is rather low: a player highly skilled would easy dominate a match even against a slightly less skilled opponent. However, the average fitness quickly improves with generations and finally reach a final value around 0.85, i.e., in the average maps evolved in the final populations, the less skilled bot scores approximately the 28% of the total points. However, the best maps evolved in each run have a fitness that is as high as 0.95, resulting in a score ratio for the least skilled bot higher than 36%. Therefore, the best maps evolved allows for a rather balanced outcome even with a large difference between the skill level of the bots.

To provide a better insight of the results just presented, we analyzed how the match balancing changes in the evolved maps with generations. Figure 3 shows the outcome, reported as the score ratio of $\texttt{bot}_1$ ($\texttt{SR}_1$), of a match between two bots using the rifle for all the possible combinations of the skill values of the bots. In particular, Figure 3a shows the balancing on a map evolved in the early stages of the evolutionary process; Figure 3b and Figure 3c show the same, on a map evolved respectively in the middle stages and in the final stages of the evolutionary process. This analysis shows that the game balancing appears qualitatively the same in all the three maps. However, the reader can notice that during the evolution, the areas corresponding to rather balanced outcomes (the yellow and green areas in Figure 3) increase, while the



(a) (b) (c)

Fig. 3. Evolving maps for balancing the match between two bots using a rifle; the *score ratio* of $\texttt{bot}_1$ on a typical map evolved in (a) early stages, (b) in the middle, (c) in the final stages of the evolutionary process. All the data points are computed as an average of 5 matches of 10 minutes each.

areas corresponding to clearly unbalanced outcomes (the dark blue and dark red areas in Figure 3) reduce.

Finally, we see an example of how evolution affects the design of the maps. Figure 4a shows $\texttt{map}_A$ with fitness $f = 0.92$ and Figure 4b shows $\texttt{map}_B$ with fitness $f = 0.96$. The reader can notice that two maps are very similar, however even small changes can have a clear impact on the game dynamics. In fact, there are only two differences between the two maps: first, in $\texttt{map}_A$ the upper-left large arena is connected to the rest of the map with two openings (see the red circle in Figure 4a), while the same connection is replaced in $\texttt{map}_B$ with a single opening (see the red circle in Figure 4b); second, the large arena in the lower-right area of the map is rather open at the bottom (see the green arrow in Figure 4a), while in $\texttt{map}_B$ the bottom of the arena presents a narrow opening into a long L-shaped corridor (see the green arrow in Figure 4b). Figure 5 shows the distribution of the positions of the kills performed by $\texttt{bot}_2$ (the least skilled one) on both the maps: while in $\texttt{map}_{A1}$ the kills are scattered all over the map (Figure 5a), in $\texttt{map}_B$ (Figure 5b) we can see a *choke* point becomes strategical as it is rather easy to defend and allows the $\texttt{bot}_2$ to perform several kills from that position and from the positions nearby.

B. Scenario 2: Rifle vs Chainsaw

In the second set of experiments we focused on improving the balancing of a match between two bots with different skill values: $\texttt{bot}_1$ has a skill value of 20 and is using the rifle, $\texttt{bot}_2$ has a skill value of 80 and is using the chainsaw. We selected these weapons as they are completely different, i.e., a medium/long range weapon and a melee weapon, leading to a completely different dynamics in game. Concerning the different skill values, we set to 20 the skill value of the bot with rifle and to 80 the skill value of the others. Our preliminary analysis showed that in this scenario the outcome is rather unbalanced in favor of the bot using the chainsaw; in fact, rifle can be extremely powerful and outperform the chainsaw, but it requires a high level of skill to be used effectively; accordingly, when the bot with the rifle has a rather low skill level, it is unable to compete against a bot with a chainsaw on a standard map.

As in the previous set of experiments, we performed 10 runs of a simple genetic algorithm using the same parameters setting. Figure 6 shows the average fitness of the evolved

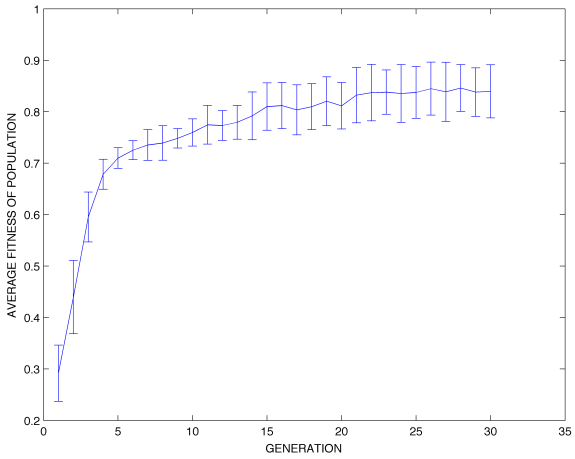
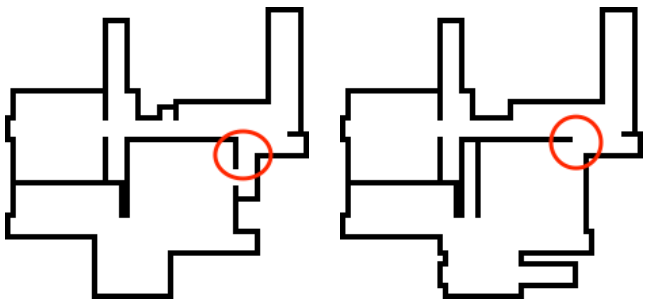
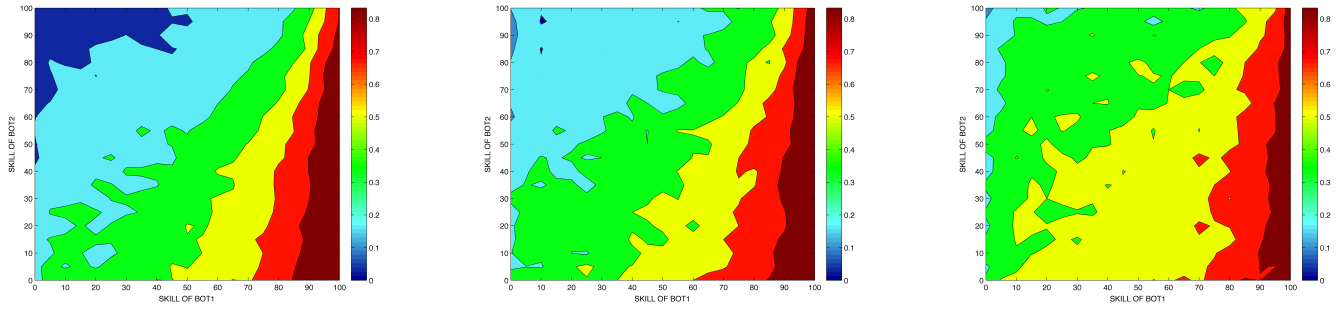


Fig. 6. Evolving maps for balancing the match between $\texttt{bot}_1$ using a rifle with skill 20 and $\texttt{bot}_2$ with skill 80 using a chainsaw. Average fitness of the map evolved during the generations. Curves are averages over 10 runs (bars represent the standard deviation).



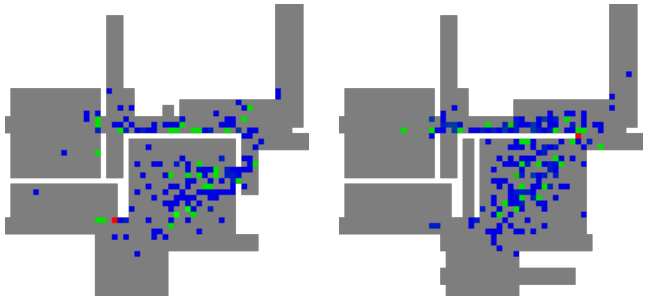
(a) (b)

Fig. 8. Two examples of the maps evolved for balancing the match between two bots using respectively the rifle and the chainsaw: (a) $\texttt{map}_C$ ($f = 0.89$) and (b) $\texttt{map}_D$ ($f = 0.92$); the red circles are used to highlight the major difference between the maps.



(a) (b) (c)

Fig. 7. Evolving maps for balancing the match between a bot using a rifle ($\texttt{bot}_1$) and a bot using a chainsaw ($\texttt{bot}_2$): the *score ratio* of $\texttt{bot}_1$ on a typical map evolved in (a) early stages, (b) in the middle, (c) in the final stages of the evolutionary process. All the data points are computed as an average of 5 matches of 10 minutes each.



(a) (b)

Fig. 9. Distribution of the more frequent positions from which $\texttt{bot}_1$ kills $\texttt{bot}_2$ on (a) $\texttt{map}_C$ (Figure 8a) and (b) $\texttt{map}_D$ (Figure 8b). The frequency of kills from each position is coded by the color from blue, the least frequent positions, to red, the most frequent positions; areas in gray are the positions from which there are no kills. Statistics are collected from 5 matches of 10 minutes each.

map over the generations. The results show that in the first generations, the average fitness of the maps is pretty low, but it quickly increases as the evolution proceeds. In the final generations, the average fitness of the maps in the population is above 0.83, that is the score ratio of the least skilled bot (the bot with the rifle gun) is around 26%, just below our target range to consider the match balanced. However, when we consider the best individuals evolved, we can find that the best maps have a fitness that is up to 0.96, that is a score ratio of the least skilled bot of 38%, that is within our target range.

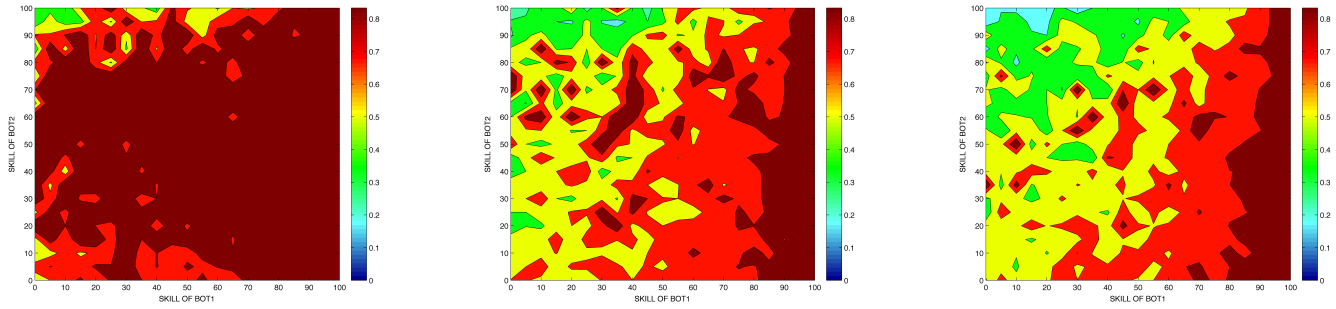
To have a better insight of how the match balancing changes during the evolution, we analyzed how the score ratio of the $\texttt{bot}_1$ ($\texttt{SR}_1$) changes with the skill values of the bots on (i) a map evolved in the early stages of the evolutionary process (Figure 7a), (ii) a map evolved in the middle stages of the process (Figure 7b), and (iii) a map evolved in the final stages (Figure 7c).

As expected, Figure 7a shows that at the beginning of the evolutionary process, the bot with the rifle gun is not able to win against the one with the chainsaw until its skill level is high enough. As the evolution proceeds, the evolved maps (see Figure 7b) start to change the balancing of the map in the area of interest: the bot with the rifle gun start to be much more competitive even with lower skill levels. At the end of the evolutionary process, as Figure 7c shows, the match balancing is clearly changed in favor of the bot with the rifle: the bot with the chainsaw is able to win only when it has a very high skill level and the opponent have a very low skill; in particular, in our scenario, the outcome of the match appears now much more balanced.

Finally, we discuss some examples to give a better insight on the underlying evolutionary process. Figure 8 shows two examples of evolved maps: $\texttt{map}_C$ and $\texttt{map}_D$; $\texttt{map}_C$ has fitness $f = 0.89$, i.e., the bot with the rifle gun has an average score ratio of approximately 30%; $\texttt{map}_D$ has a fitness $f = 0.92$, i.e., the bot with the rifle has an average score ratio of approximately 33%. The two maps are very similar, but such small differences can still have an impact on the balancing of the match. In particular, in this specific example, the major difference involves the connection between the uppermost part of the map with the bottom part: in $\texttt{map}_C$ a rather short corridor opens in a rather large room that is connected to the rest of the map; in $\texttt{map}_D$ the room is even larger and it is directly connected whit the upper part of the map. This small change results in a new strategical position, that allows to control all the movements in the map from the lower to the the upper part (and the opposite), especially using a long range weapon as a rifle. To confirm this analysis, Figure 9 shows the positions of the bot during the kills performed during the matches: it appears that in $\texttt{map}_D$ the new strategical position became the most effective position for the $\texttt{bot}_1$ and allows to kill the opponent not only in the room below but also to have a very good shoot on the upper long corridor.

C. Scenario 3: Rifle vs Grenade Launcher

In the last set of experiments, we focused on balancing the match between $\texttt{bot}_1$, that uses the rifle and has a skill value of 20, and $\texttt{bot}_2$, that uses the grenade launcher and has a skill value of 80. Similarly to the previous scenario, we selected two weapon completely different: the rifle, a weapon with an *instant-hit* mechanics, and the grenade launcher, an explosive weapon with a *delayed-hit* mechanics. Therefore, this results in two different playing strategies that provide additional challenges in terms of match balancing. Concerning the skill values selected, grenade launcher is a weapon very difficult to use and starts to become effective against the rifle only when used by highly skilled bots. According to our preliminary analysis, this scenario is heavily unbalanced in favor of the bot using the rifle, despite being much less skilled.



(a) (b) (c)

Fig. 11. Evolving maps for balancing the match between a bot using rifle ($\texttt{bot}_1$) and a bot using grenade launcher ($\texttt{bot}_2$): the *score ratio* of $\texttt{bot}_1$ on a typical map evolved in (a) early stages, (b) in the middle, (c) in the final stages of the evolutionary process. All the data points are computed as an average of 5 matches of 10 minutes each.

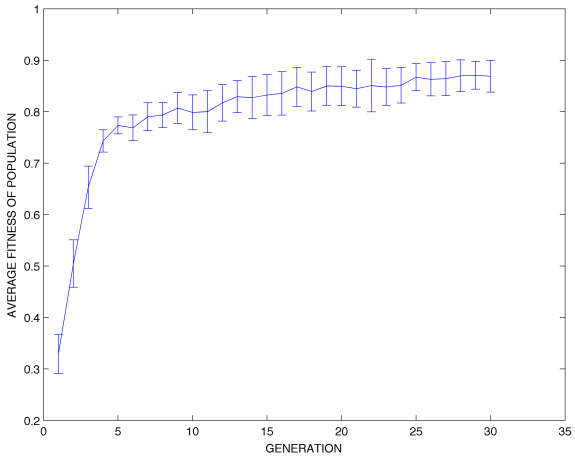
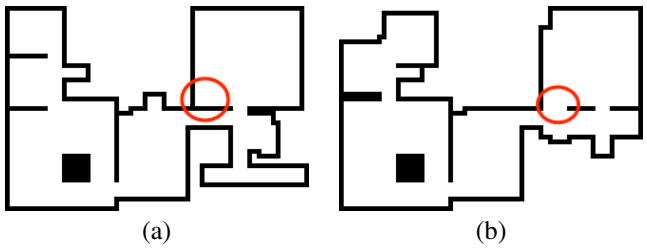


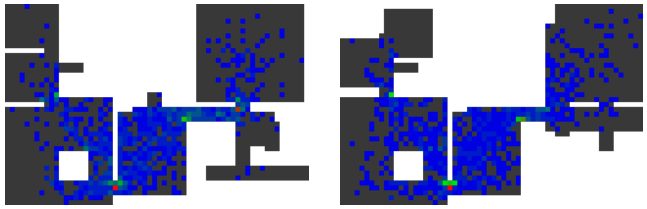
Fig. 10. Evolving maps for balancing the match between $\texttt{bot}_1$ using a rifle with skill 20 and $\texttt{bot}_2$ with skill 80 using a grenade launcher. Figure shows the average fitness of the map evolved during the generations. Curves are averages over 10 runs (bars represent the standard deviation).



(a) (b)

Fig. 12. Two examples of the maps evolved for balancing the match between two bots using respectively the rifle and the grenade launcher: (a) $\texttt{map}_E$ ($f = 0.93$) and (b) $\texttt{map}_F$ ($f = 0.98$); the red circles are used to highlight the main differences between the maps.

To evolve the maps, we performed 10 runs of a simple genetic algorithm with the same parameters settings of the previous experiments. Figure 10 shows the average fitness of the maps evolved during the generations. In the first generations, the average fitness of the maps in the population is lower than 0.4, i.e., the bot with the grenade launcher has a score ratio lower than 7%; as the generations proceed, the average fitness of the population increases and reaches a value around



(a) (b)

Fig. 13. Distribution of the more frequent positions from which $\texttt{bot}_1$ kills $\texttt{bot}_2$ on (a) $\texttt{map}_E$ (Figure 12a) and (b) $\texttt{map}_F$ (Figure 12b). The frequency of kills from each position is coded by the color from blue, the least frequent positions, to red, the most frequent positions; areas in gray are the positions from which there are no kills. Statistics are collected from 5 matches of 10 minutes each.

0.87, i.e., a score ratio around 29%, just below the target range. However, the best maps evolved maps have a fitness value up to 0.97, i.e., a score ratio larger than 39%.

Figure 11 shows how the evolutionary process affects the balancing of the maps. Figure 11a shows the match balancing on a typical map evolved in early stages of the evolutionary process, where the bot using the rifle ($\texttt{bot}_1$) appears almost always largely superior to its opponent. Later in the evolutionary process, the map evolved are slightly more balanced (see Figure 11b): despite the bot with the rifle almost always scores more than its opponent, the outcome appears more balanced when the skill value of the bot with grenade launcher ($\texttt{bot}_2$) is larger than the skill of the opponent. Then, in the final stages of evolution, the map evolved allows a balanced outcome of the match whenever the skill of the bot using the rifle ($\texttt{bot}_1$) is not too high.

Finally, Figure 12 shows two examples of the maps evolved. The first map, $\texttt{map}_E$ (Figure 12a), has a fitness of 0.93, i.e., score ratio of the looser bot is above 34%. The second map, $\texttt{map}_F$ (Figure 12b), has a fitness of 0.98, i.e., a score ratio above 41%. Despite they are similar, the reader can notice that in $\texttt{map}_E$ the room in the upper right area is connected to the rest of the map by a single opening in the middle of a corridor (see red circle in Figure 12a); instead, in $\texttt{map}_F$ the same room is connected to the rest of the maps by two separate openings, one of them pretty close to the central room (see red circle in Figure 12b). Such an apparently small difference actually affects the game dynamics: Figure 13 shows

the distribution of the positions from which $\texttt{bot}_1$ killed the opponent on $\texttt{map}_E$ and on $\texttt{map}_F$; while in $\texttt{map}_E$ there are two sort of choke points that provide ideal spots to control the map (see Figure 13a), in $\texttt{map}_F$ there is a single choke point to exploit for $\texttt{bot}_1$ (see Figure 13b). In addition, the second opening is more close to the central room improving the chance for $\texttt{bot}_2$ to shoot the opponent without being killed.

VI. Conclusion

We proposed a novel methodology to deal with match balancing in Cube 2, an open source mutliplayer FPS. Our approach is based on procedural content generation and consists of evolving game maps that are specifically designed to improve the match balancing. Accordingly, evolved maps are evaluated using a fitness function computed on the basis of the statistics collected from a simulated match between bots. We tested our approach on three different challenging scenarios: (i) balancing the match between two players with different skill levels but using the same weapon (the rifle), (ii) balancing the match between a skilled player using the chainsaw and a less skilled bot using the rifle, and (iii) balancing the match between a skilled player using the grenade launcher and a less skilled players using the rifle. Our experimental results suggest that our approach can successfully evolve maps for Cube 2 that improve the match balancing in each one of the three scenarios tested. Future works will include a validation of our approach with human players as well as extending our approach to other first person shooters, to more complex scenarios (e.g., match between teams) and to a more sophisticated map design.

References

[1] R. Koster, *Theory of Fun for Game Design*. PARAGLYPH PRESS, 2005.

[2] 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.

[3] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, G. N. Yannakakis, and C. Grappiolo, "Controllable procedural map generation via multiobjective evolution," *Genetic Programming and Evolvable Machines*, vol. 14, no. 2, pp. 245–277, 2013.

[4] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008402.2008411

[5] A. E. Elo, *The rating of chessplayers, past and present*. Batsford London, 1978, vol. 3.

[6] R. Herbrich, T. Minka, and T. Graepel, "Trueskill™: A bayesian skill rating system," in *Advances in Neural Information Processing Systems*, 2006, pp. 569–576.

[7] R. C. Weng and C.-J. Lin, "A bayesian approximation method for online ranking," *The Journal of Machine Learning Research*, vol. 12, pp. 267–300, 2011.

[8] S. Nikolenko and A. Sirotkin, "A new bayesian rating system for team competitions," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 601–608.

[9] J. C. Huang and B. J. Frey, "Cumulative distribution networks and the derivative-sum-product algorithm: Models and inference for cumulative distribution functions on graphs," *The Journal of Machine Learning Research*, vol. 12, pp. 301–348, 2011.

[10] O. Delalleau, E. Contal, E. Thibodeau-Laufer, R. C. Ferrari, Y. Bengio, and F. Zhang, "Beyond skill rating: Advanced matchmaking in ghost recon online," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 3, pp. 167–177, 2012.

[11] J. Feil and M. Scattergood, *Beginning game level design*. Course Technology PTR, 2005.

[12] A. C. Clayton, *Introduction to Level Design for PC Games*. Charles River Media, Inc., 2003.

[13] E. Byrne, *Game level design*. Cengage Learning, 2005.

[14] S. Bjork and J. Holopainen, *Patterns in game design*. Cengage Learning, 2005.

[15] B. Kreimeier, "The case for game design patterns," 2002. [Online]. Available: http://www.gamasutra.com/view/feature/132649

[16] K. Hullett and J. Whitehead, "Design patterns in fps levels," in *proceedings of the Fifth International Conference on the Foundations of Digital Games*. ACM, 2010, pp. 78–85.

[17] R. Giusti, K. Hullett, and J. Whitehead, "Weapon design patterns in shooter games," in *Proceedings of the First Workshop on Design Patterns in Games*. ACM, 2012, p. 3.

[18] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Optimization of platform game levels for player experience," in *AIIDE*, C. Darken and G. M. Youngblood, Eds. The AAAI Press, 2009.

[19] K. Compton and M. Mateas, "Procedural level design for platform games," in *AIIDE*, J. E. Laird and J. Schaeffer, Eds. The AAAI Press, 2006, pp. 109–111.

[20] N. Shaker, G. N. Yannakakis, and J. Togelius, "Towards automatic personalized content generation for platform games," in *AIIDE*, G. M. Youngblood and V. Bulitko, Eds. The AAAI Press, 2010.

[21] 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.

[22] 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.

[23] 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. [Online]. Available: http://doi.acm.org/10.1145/2001576.2001631

[24] 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.

[25] J. Dormans, "Adventures in level design: generating missions and spaces for action adventure games," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 1.

[26] 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.

[27] 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.

[28] 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 *AIIDE*, M. Riedl and G. Sukthankar, Eds. The AAAI Press, 2012.

[29] "Cube 2: Sauerbraten," http://sauerbraten.org/.

[30] K. Sastry, "Single and multiobjective genetic algorithm toolbox for matlab in C++," Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S.