Fight or Flight: Evolving Maps for Cube 2 to Foster a Fleeing Behavior

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

Luca Arnaboldi
Dipartimento di Elettronica,
Informazione e Bioinformatica,
Politecnico di Milano,
Email: luca1.arnaboldi@mail.polimi.it

***Abstract*—The fight-or-flight response is a typical physiological reaction to an imminent threat, which prepares the body to fight or flee. Such a mechanism can be also exploited to some extent in game design to elicit strong emotions in players. However, in first person shooters, fleeing is often a not viable option, making fight inevitable. In this work, we suggest that map design can be effectively used to deal with this issue. In particular, we presented a procedural content generation approach, based on evolutionary computation, to evolve maps for Cube 2, an open source first person shooter. Our results show that the design of evolved maps is effectively able to foster the emergence of a fleeing behavior, even in AI controlled characters specifically designed to fight and chase the opponents.**

I. Introduction

Fight-or-flight is a typical physiological response to an attack or to a perceived threat [2]. Such a mechanism can be also used by game designers to elicit strong reactions in players, such as stress, fear, anxiety or excitement. This idea is often exploited in survival horror games [22], while the design of first person shooters, being mainly focused on providing a frantic game experience, often makes the fight inevitable and might desensitize the perception of danger in players. Accordingly, our aim is to investigate how to deal with this issue with an ad hoc level design for first person shooters.

Today, game developers generally exploit standard tools and pipelines to speedup the whole development process. Unfortunately, level design is a trial and error process that still heavily relies on custom tools as well as on human expertise. Nevertheless, level design is often one of the main challenge for game developers and it is often the key to a successful game. In particular, in mutliplayer first person shooters the design of *maps*, i.e., the environment where the game takes place, plays a major role to provide players with an exciting and frantic experience: in fact, the design of maps affects the pacing, the balancing and ultimately how the the game is played.

Recent works in the literature [19], [21], [16], [3], [6], [24] suggest that procedural content generation [25] is a promising approach to develop tools that could improve the level design process by either assisting or, in some specific tasks, replacing the work of designers. In this paper, we present a procedural content generation approach to design maps for *Cube 2: Sauerbraten*, an open source first person shooter. In particular, our work is based on and inspired by previous works on the procedural generation of maps for first person shooters. However, while previous works focused on survival time [4], balancing [14], team-play [18], or resources distribution [1], we focus on evolving maps with a design that makes fleeing a more viable strategy for players.

We performed an experimental analysis to test our approach. The results are encouraging and show that, even in simulated matches between bots designed to fight and chase the opponents, a specific map design leads to the emergence of a fleeing behavior.

This paper is organized as follows. In Section II we provide an overview of the previous works in the literature on the procedural generation of maps for first person shooters. Then, in Section III we briefly describe *Cube 2: Sauerbraten* and in Section IV the approach used in this work to evolve the maps. Finally in Section V we present our experimental results and in Section VI we draw some conclusions.

II. Related Work

A. Map Design in First Person Shooters

Despite being a widely studied subject, the map design in first person shooters is not yet a standardized process and it still heavily relies on human expertise. Accordingly, some authors [12], [8], [9] tried to identify the most frequent design patterns that appear in first person shooters maps. In particular, Hullet et al. described the major patterns used for level design in single-player shooters [11]. Instead, Liapis et al. [15] studied general patterns, usable across different game genres (including first person shooters [1]). The major challenge of this research direction lies on identifying relevant patterns and on understanding how they affect the game dynamics and how they interact with game mechanics.

B. Search-Based Procedural Content Generation

So far, *search-based procedural content generation* (SB-PCG) [25] has been widely used to automatically generate high-quality game contents and proved to be effective in several game genres, including platform games [19], [5], [21], racing games [23], [16], [3], RPG games [6], strategy games [24], and others [10], [20]. This work is based on previous works [4], [14] which applied SB-PCG to evolve

Fig. 1. A screenshot of *Cube 2: Sauerbraten*.

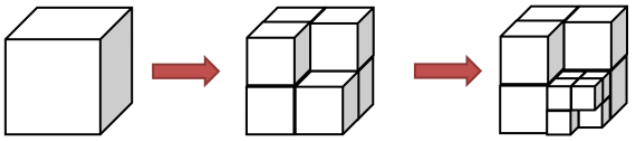


Fig. 2. The recursive structure of an *octree*.

maps for *Cube 2*. In particular, Cardamone et al. [4] proposed four different map encodings and applied a genetic algorithm to evolve *interesting* maps; in their work, the fitness function used to evaluate the maps was based both on the size of the map and on the average length of the fights (i.e., how long players do survive after starting to fight an opponent). Later, Lanzi et al. [14] extended the work of Cardamone et al. by focusing on the problem of evolving *balanced* maps for FPS when players either have different skill levels or are using different weapons. More recently, Ølsted et al. [18], focused on evolving maps for multiplayer FPS involving teams, introducing a new map encoding and evaluating the maps through user interaction. Finally, the work of Cachia et al. [1] further extended the previous works to evolve *Cube 2* maps with two floors; to evaluate the maps they combined the metrics used in [4], [14] with general heuristics for level design introduced in [15].

III. Cube 2: Sauerbraten

Cube 2: Sauerbraten (*Cube 2* in brief), is a rather popular open source arena first person shooter (see a screenshot of the game in Figure 1). It includes several game modes, such as *deathmatch*, *capture the flag*, *domination*, and others; it also features seven different weapons, ranging from melee weapons to long ranged ones, i.e., chainsaw, pistol, machine gun, shotgun, rifle, grenade launcher, and rocket launcher. In addition, *Cube 2* was used in several works in the literature [4], [14], [1] as it allows the creation of custom maps and offers AI controlled characters.

A. Maps

The maps of *Cube 2* are based on a recursive structure, called *octree*; indeed, each *octree* is a sort of cube recursively

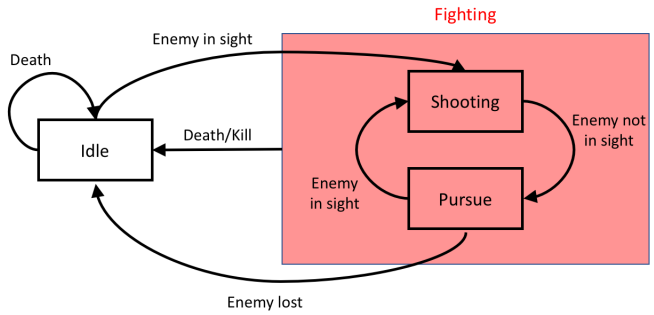


Fig. 3. The behavior of bots in *Cube 2*.

composed by 8 smaller *octrees* (see Figure 2). The map editor provided with the game allows to build complex 3D structures by editing and combining together several *octrees*. In fact, for each *octree* it is possible either to change its size or to move inward any of its vertexes; moreover, it is also possible to recursively edit or remove the *octrees* contained in a bigger *octree*. Finally, the editor allows also to place the spawning points (i.e., where players appear after being killed), ammunition, weapons, and power-ups.

B. AI

Cube 2 provides AI controlled characters, generally called *bots*, that can play against or with players. In general, the behavior of the bots in *Cube 2* can be effectively described with the finite state machine represented in Figure 3. When a match starts, the bot is in an *Idle* state, where it explores and patrols the map searching for enemies. As soon as an enemy is in sight, the bot enters in a *Fighting* state, that can be decomposed in a *Shooting* state and in a *Pursue* state: in the *Shooting* state, the bot has the enemy in sight and is either aiming or shooting at it; whenever the bot looses temporarily the sight of the enemy, it enters in the *Pursue* state, where it navigates the map searching for the enemy. If the enemy is not visible for a given amount of time, the bot re-enters in the *Idle* state. The bot re-enters the *Idle* state also from *Fighting* state when it kills the enemy or it is killed by him (this could also happen in the *Idle* state, if a not visible enemy kills the bot).

It is possible to adjust the skill level of the bots by setting a skill parameter which ranges from 0 to 100. Unfortunately, this skill parameter affects all bot behaviors and it is not possible to adjust only specific skills (e.g., it is not possible to increase or decrease the navigation skills of the bot without affecting also its aiming skills). Accordingly, we extended the AI routines of *Cube 2* to allow for the independent adjustment of individual bot skills. In particular, we identified three major skills and made them independent from the *general* skill level of the bot: (i) *aiming* skill, that affects how accurate the bot is when shooting; (ii) *positioning* skill, that affects how good the bot is at keeping an appropriate combat distance to better exploit the weapon used; (iii) *camping* skill, that affects how good the

Fig. 4. A screenshot of some maps available in *Cube 2*.

bot is at moving and aiming at the same time. All these skills have a value that ranges from 0 to 100.

IV. Our approach

In this section we describe the map encoding and the fitness function we used to evolve maps for *Cube 2*; in addition, we describe also some additional metrics computed to analyze the design of evolved maps.

A. Encoding

Cube 2 provides a map editor that allows users to create custom maps visually appealing and with a rather complex structure (see some examples in Figure 4). However, in this work we focus only on the map topology and, more specifically, on a kind of maps that are quite easy to generate. Accordingly, as in [4], [14], our maps have a single floor represented as a 64×64 matrix of square tiles. Each tile can be either empty or filled with a block of fixed height (each block is built using *octrees*). The empty tiles represent the map area that can be navigated by players, e.g., rooms, arenas and corridors of the maps. The blocks cannot be traversed by players and their height is specifically designed to make it not possible for players to jump over them; hence, blocks are basically used to separate the open areas of the map. Being a direct encoding of the 64×64 matrix of tiles not very effective, we used an encoding proposed by Cardamone in [4], defined as follows. All the tiles of the map are by default filled with a block and only empty tiles are encoded. 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 $n_a = 20$ and $n_c = 60$). In order to generate a map usable in *Cube 2* with this encoding, two additional steps are required: (i) all the empty tiles that are not reachable from the center of the map are removed, i.e., we make the maps fully connected; (ii) we uniformly distribute spawning points, weapon pick-ups, and add-ons over the empty tiles of the map.

B. Fitness

To evaluate how much fleeing is a viable option in a given map, we designed a fitness function based on the gameplay statistics we collect from a simulated match between bots on the map to evaluate. Despite the fact that an approach based on simulation (also followed in [4], [14]) is rather time consuming and relies on the game AI, it allows us to evaluate better how the map design *actually* affects the game dynamics. Instead, a fitness function based only on a *static* analysis of the maps (i.e., the topology, the placement of resources, etc.) might either fail to capture entirely the relationship between map design and game dynamics or lead to unnecessary design constraints (e.g., symmetry, fair distribution of resources, etc.). Accordingly, to evaluate the fitness of a map we simulate a match of 30 minutes (this value was set based on an empirical analysis on randomly generated maps); simulations are performed with accelerated game time such that a 30 minutes match takes only few seconds to be simulated. Based on the statistics collected during this simulated match, we compute the fitness of the maps as follows:

$$f = \frac{n_{lost}}{n_{fight}}, \tag{1}$$

where n_{fight} is the number of times bots enter in *Fighting* state (see Figure 3), n_{lost} is the number of times bots go from *Pursue* state to *Idle* state (see Figure 3), because they lost contact with the enemy during a fight. Therefore, this fitness function basically measures the fraction of fights that do not end with either killing the opponent or being killed.

C. Design Metrics

In the following, we present additional metrics useful to better understand the design of evolved maps and how it affects the most important game dynamics.

Balancing. In a FPS, *balancing* is very important to make sure players have an enjoyable game experience [14], [13]. Following the definition introduced in [14], we evaluated the balancing of a map by computing the entropy of the kills distributions among bots:

$$B = -\sum_i \frac{k_i}{K} \log_N \left(\frac{k_i}{K} \right), \tag{2}$$

where k_i are the kills performed by the i-th bot in the simulated match, N is the number of bots in the match, and $K = \sum_i k_i$. As a result, B is always between 0 and 1; in particular, it is close to 1 when all the bots perform almost the same number of kills, while it is close to 0 when a single bot performs almost all the kills. Figure 5 provides an insight of the value

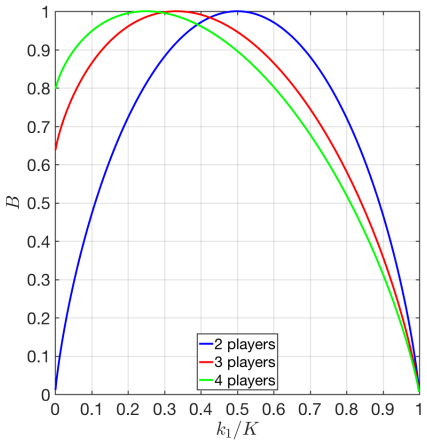


Fig. 5. The value of B computed according Equation 2 with two bots (blue line), three bots (red line), and four bots (green line); on the x-axis is reported the fraction of kill performed by the first bot (we assume that the other kills are evenly shared among other bots).

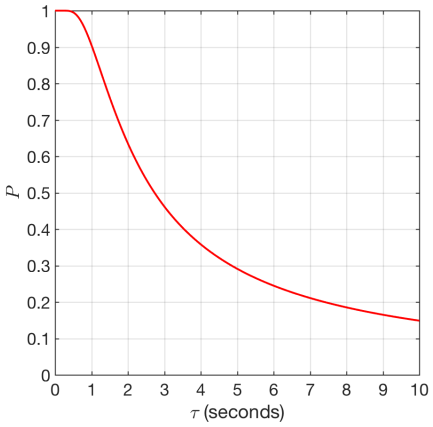


Fig. 6. The value of P computed according to Equation 3; on the x-axis is reported $\tau = t_{idle}/n_{fight}$, i.e., the average time (in seconds) between two consecutive fights.

of B computed by Equation 2, with a number of bots that goes from 2 to 4.

Pacing. In a FPS, *pacing* generally refers to the frequency with which players are engaged in fights. It tends to significantly affect game experience: a too high pace usually leads to a stressful experience, but a too low pace typically leads to a boring experience. To evaluate how the design of the map affects the pacing, we defined the following metric:

$$P = \frac{2}{1 + e^{-3/\tau}} - 1, \tag{3}$$

where $\tau = t_{idle}/n_{fight}$ represents the average time the bots spend between two fights (t_{idle} is the time spent in *Idle* state and n_{fight} is the number of times bots enter *Fighting* state). As a result, the value of P is computed with a sigmoidal function (see Figure 6) that is almost 1 when τ is very small and then quickly decreases toward 0.

Kill Streaks. A kill streak is the number of kills a player is able to perform without dying. Completing a long streaks is generally a very exciting experience for a player and several popular shooters, e.g., *Call of Duty* [1] and *Halo* [2], reward players when they achieve a long kill streak. As map design could affect the average length of kill streaks, we compute also the average length of the kill streaks performed during the match:

$$L_{strk} = \frac{\sum_i streak_i}{N_{strk}}, \tag{4}$$

where $streak_i$ is the length of i-th kill streaks achieved during the game and N_{strk} is the number of streaks during the whole match.

Fighting Time. In a FPS, players basically spend time either exploring the map (i.e., searching for an opponent or for resources) or fighting (i.e., either shooting or chasing). The map design might have a significant impact on the time balance between these two activities. Accordingly, to keep track of this, we compute the following metric:

$$T_f = \frac{t_{fight}}{t_{fight} + t_{idle}}, \tag{5}$$

where t_{fight} and t_{idle} are respectively the time spent by all the bots in respectively the *Fighting* and in the *Idle* states (see Figure 3). Therefore, T_f, that ranges from 0 to 1, computes the fraction of the match time that bots spend fighting.

Bots Statistics. Finally, we keep also track of some bot statistics that might be affected by the map design. In particular, we collect (i) the number of kills performed by each bot during the match (k_i), (ii) the total number of kills performed during the match (K), and (iii) the shooting accuracy of each bot (acc_i), computed as the number of hits over the number of shots performed.

V. Experimental Analysis

We applied a genetic algorithm to evolve maps for *Cube 2* that make it possible for players to flee. In our experimental analysis, maps are evolved for two players with slightly different playing styles. The first playing style, named *Berserker*, is quite an aggressive and very dynamic playing style: it is the typical style of players with poor aiming skills, more effective in close combats situated in open spaces (where they are very good at moving and aiming at the same time). The second playing style, named *Gunner*, exhibits a more balanced and static playing style: typical of players with average aiming skills, that need more time to shoot accurately and are more effective in closed spaces. To model these different playing styles with *Cube 2* bots we used the following AI parameters. The *Berserker* bot has general skill equal to 75, aiming skill equal to 40, positioning skill equal to 50, and camping skill

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

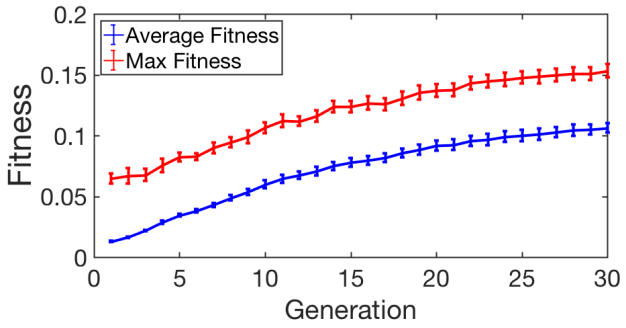


Fig. 7. Average (blue line) and maximum (red line) fitness values of the maps evolved for each generation. Curves are computed as the average of 12 runs (bars represent the standard errors).

equal to 90. The *Gunner* bot has general skill equal to 75, aiming skill equal to 75, positioning skill equal to 75, and camping skill equal to 20.

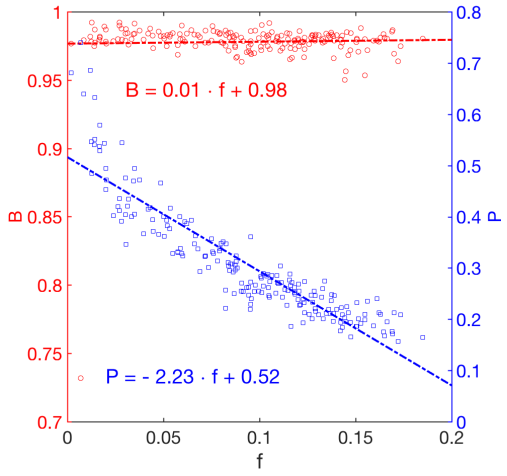
The setup of the evolutionary process is similar to the one used in [14]. We performed 12 runs [3], each one consisting of 30 generations with a population of 75 maps; uniform mutation is applied with probability 0.3; matrix crossover [17] is applied with probability 0.3. The fitness of each map is evaluated running a 30 minutes match with a *Berserker* bot against a *Gunner* bot, both using a rocket launcher.

Figure 7 shows that the genetic algorithm is effectively able to steadily improve both the average and the maximum fitness of the population as the generations progress. As the results show, the average fitness of the population is initially almost close to zero and, after 30 generations, the average fitness is around 0.1, while the maximum value of the fitness grows from a value around 0.05 to a value close to 0.15. The average and maximum fitness of the evolved maps might seem quite small, but they are actually quite difficult to achieve without a specific design of the maps. In fact, despite the overall size of the maps is quite limited [4] and the bots AI is primarily designed to pursue the opponent, our results show that the number of fights that do not end with a death goes from less than 1% on average (with a maximum of 6%) in the initial population, towards to almost 11% on average (with a maximum of 15%) in the final population.

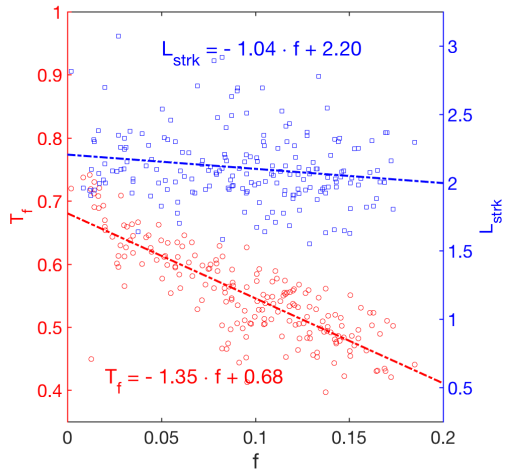
In order to get a better insight about the design of maps evolved by the genetic algorithm, we computed and reported additional metrics. Figure 8 shows the analysis of general gameplay metrics, i.e., the balancing, the pacing, the average length of kill streaks, and the fraction of time spent in fights. Figure 9 shows statistics about bots performance on the map, such as the accuracy of the bots, the fraction of kills performed by bots, and the total number of kills. Data show that evolving maps to increase the fitness value does not affect the balancing (reported in red in Figure 8a), the

[3]Experiments were performed on a machine with 4 CPU cores, that made it easy to run 4 simulations in parallel.

[4]All the maps evolved are based on a grid of 64×64 tiles as described in Section IV.



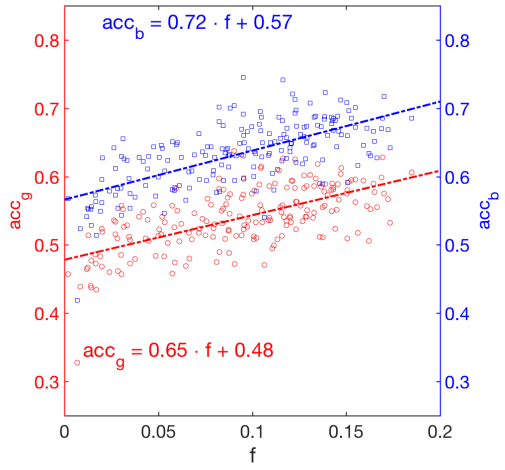
(a)



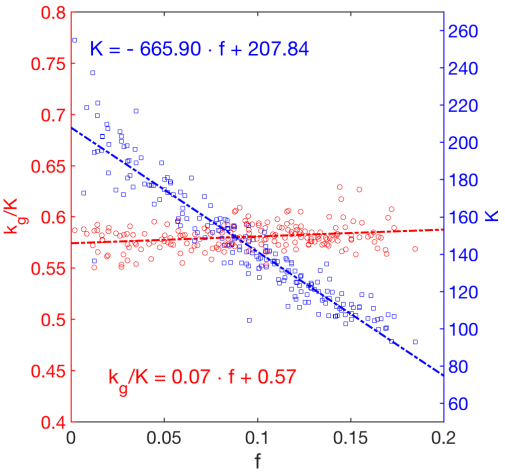
(b)

Fig. 8. Analysis of gameplay metrics in the evolved maps with respect to the fitness, f: (a) balancing in red and pacing in blue; (b) fraction of time spent fighting in red and average length of kill streaks in blue. The values of the metrics are plotted for each map with a circle and fitted with linear regression; the resulting trend-lines are plotted (dashed lines) along with their equations.

distribution of kills performed by the bots (reported in red in Figure 9b), and has just a minor effect on the average length of kill streaks (reported in blue in Figure 8b). Instead, the design of maps with high fitness values affects more significantly the pacing (reported in blue in Figure 8a), the amount of time spent fighting (reported in red in Figure 8b), and the total number of kills performed by the bots (reported in blue Figure 9b). As expected, the pacing, P, of the maps decreases from a value around 0.5 (when f is almost 0) to 0.2 (when f is greater than 0.15), which means that average time between fights increases from less than 3 seconds to almost 8 seconds (according to Equation 3); the same happens to the amount of time spent fighting, i.e., T_f, that decreases from almost 70% (when the fitness is small) to approximately 50% (when the fitness is large) of the whole match time; also the total number



(a)



(b)

Fig. 9. Analysis of bots statistics on the evolved maps with respect to the fitness, f: (a) accuracy of *Gunner* bot, acc_g, in red and *Berserker* bot, acc_b, in blue; (b) fraction of kills performed by the *Gunner* bot, k_g, in red and total number of kills, K, in blue. The values of the statistics are plotted for each map with a circle and fitted with linear regression; the resulting trend-lines are plotted (dashed lines) along with their equations.

of kills performed by the bots drops from more than 200 to approximately 100. These results suggest that maps that foster a fleeing behavior generally require a more complex structure that also makes it more difficult for the players to find each other, reducing the pace and the number of fights the bots are engaged in. Finally, Figure 9a shows that the accuracy of *Gunner* and *Berserker* bots slightly improves with the fitness value. This is probably due to the fact that maps with higher fitness values feature more close spaces, where it is easier to shoot more accurately.

Figure 10 summarizes all the changes in the map design from early generations, i.e., first 5 generations, to late generations, i.e., last 5 generations. All the changes of the metrics were tested with a Wilcoxon rank sum test [7], with p-values $p < 0.05$ considered statistically significant. The results of this

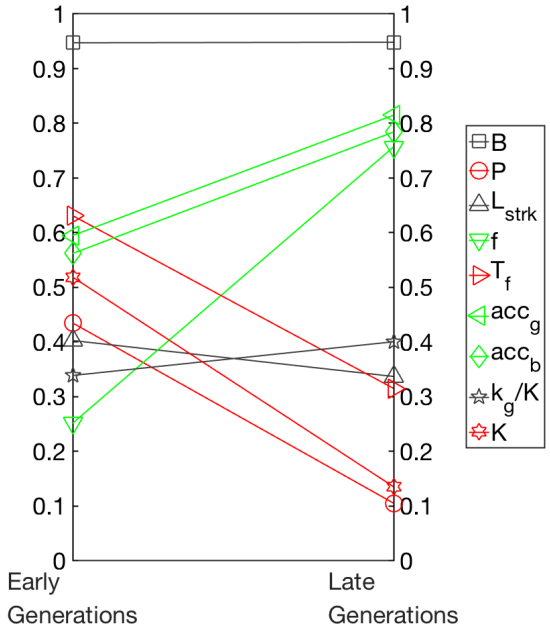


Fig. 10. A summary of the design differences between maps in the early generations (i.e., maps evolved in the first 5 generations) and maps in late generations (i.e., maps evolved in the last 5 generations). For each metric is reported the average value computed on maps from early and late generations. The line is reported in: (i) green, if the metric increases and the difference is statistically significant; (ii) red, if the metric decreases and the difference is statistically significant; (iii) gray, if the change is not statistically significant. All the values of metrics are normalized between 0 and 1, to make it easier to plot them together.

analysis are consistent with the previous discussion.

Figure 11 shows six examples of maps evolved in the 12 runs performed along with their fitness values. Maps in the left column have low fitness values and are typically evolved in early generations; maps in the middle column are examples of maps evolved halfway through the process; maps in the right column are examples from late generations with high values of fitness. Despite maps on the left seem to have a slightly more simple structure than maps on the right column, overall they look quite similar. Accordingly, to get a better insight we looked into the game dynamics that take place on these maps. Figure 12 shows the distributions of the deaths of bots on these six examples of maps. The distributions show that in maps with low fitness values the action takes place in a very limited area of the map, i.e., these maps feature a sort of central *hub* connected to all the peripheral areas. In contrast, in maps with high fitness values the action is distributed all over the map and they can generally be navigated following a closed circular path; in particular, it is interesting to note that adding even a single shortcut to such a navigation path (as in both maps in the middle column of Figure 12) leads to significantly reducing the emergence of a fleeing behavior, i.e., a lower fitness value.

Fig. 11. Examples of the maps evolved reported with their fitness values.

VI. Conclusions

In this work, we presented a procedural content generation approach to design FPS maps that allow to avoid fights more easily. In particular, we used a genetic algorithm to evolve maps for *Cube 2*, an open source FPS. We encoded the maps following the same approach introduced in [4] and we evaluated them on the basis of statistics collected from simulated matches between bots. To get additional insight about the design of the evolved maps we also analyzed additional metrics of the maps, such as the balancing, the pacing, the average length of kill streaks, the average fraction of match time spent in a fight, the shooting accuracy, and the total number of kills. Our results showed that the genetic algorithm is actually able to evolve maps that foster the emergence of a fleeing behavior, even in simulated matches involving bots that are specifically designed to always fight and chase the opponents. The analysis of evolved maps shows that this result is achieved with a design that does not include any central hub area but, instead, features a quite long closed path that allows to navigate almost the whole map. In addition, we noted that while such a design pattern does not have any significant effect on the balancing and on the average length of kill streaks, it significantly reduces the pacing, the time spent in a fight and the total number of kills performed during the match. As future work, it might be interesting to validate our results with human players and investigate whether the evolved maps are actually able to elicit some sort of fight-or-flight response. To improve the evaluation of the evolved maps, we might consider to introduce an *explicit* fleeing behavior that bots can select to avoid fights. We also plan to investigate how the distribution of weapons, ammunition and power-ups affects the emergence of fleeing behaviors. Finally, we want to explore in depth the trade-off between the pacing and the emergence of fleeing behavior that we found in this work.

References

[1] William Cachia, Antonios Liapis, and Georgios N Yannakakis. Multi-level evolution of shooter levels. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[2] Walter B Cannon. The wisdom of the body. *The American Journal of the Medical Sciences*, 184(6):864, 1932.

[3] Luigi Cardamone, Daniele Loiacono, and Pier Luca 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*, GECCO '11, pages 395–402, New York, NY, USA, 2011. ACM.

[4] Luigi Cardamone, Georgios N. Yannakakis, Julian Togelius, and Pier Luca Lanzi. Evolving interesting maps for a first person shooter. In *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I*, EvoApplications'11, pages 63–72, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Kate Compton and Michael Mateas. Procedural level design for platform games. In John E. Laird and Jonathan Schaeffer, editors, *AIIDE*, pages 109–111. The AAAI Press, 2006.

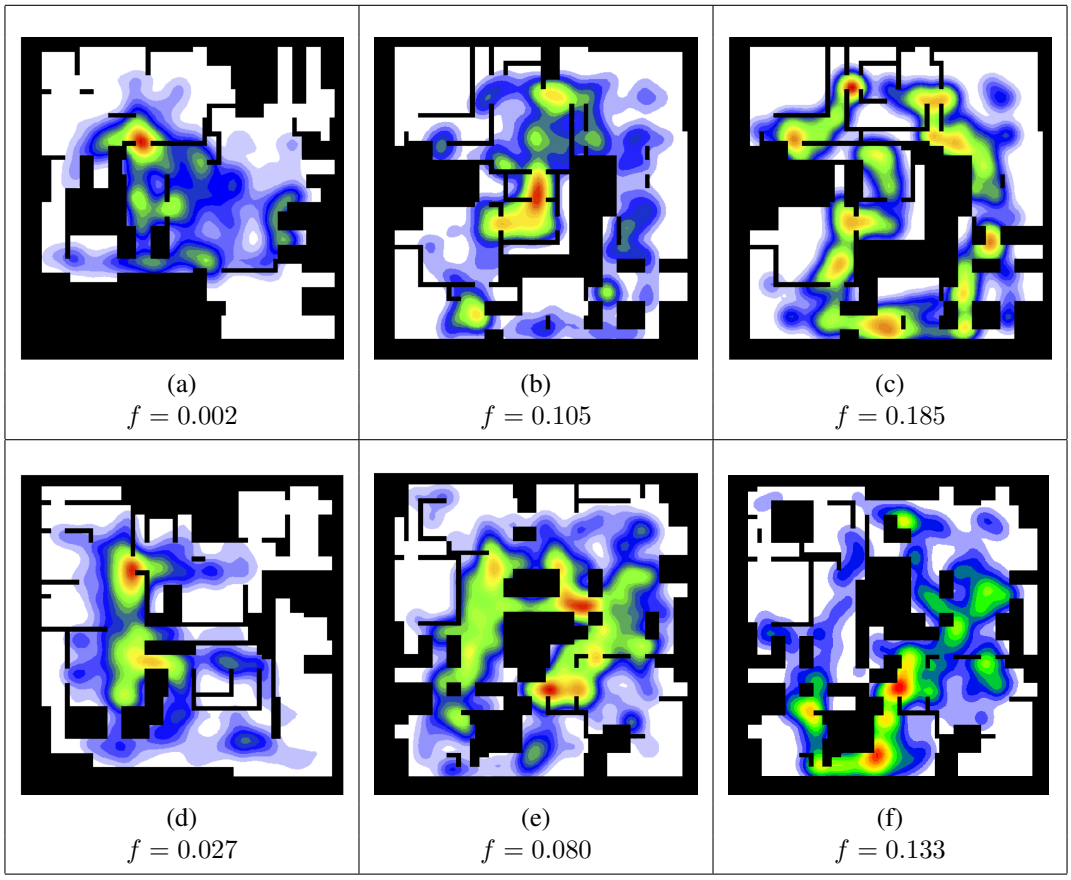


Fig. 12. Heat maps of deaths distribution over the six maps depicted in Figure 11. Color encodes the density of deaths: red areas corresponds to the highest density while blue areas to the lowest one.

[6] Joris Dormans and Sander Bakkes. Generating missions and spaces for adaptable play experiences. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):216–228, 2011.

[7] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric Statistical Inference*. CRC Press, 2010.

[8] Robert Giusti, Kenneth Hullett, and Jim Whitehead. Weapon design patterns in shooter games. In *Proceedings of the First Workshop on Design Patterns in Games*, page 3. ACM, 2012.

[9] Christian Güttler and Troels Degn Johansson. Spatial principles of level-design in multi-player first-person shooters. In *Proceedings of the 2Nd Workshop on Network and System Support for Games*, NetGames '03, pages 158–170, New York, NY, USA, 2003. ACM.

[10] Erin J. Hastings, Ratan K. Guha, , and Kenneth O. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):245–263, 2009.

[11] Kenneth Hullett and Jim Whitehead. Design patterns in fps levels. In *FDG '10: Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 78–85, New York, NY, USA, 2010. ACM.

[12] Kenneth M. Hullett. *The Science of Level Design: Design Patterns and Analysis of Player Behavior in First-person Shooter Levels*. PhD thesis, Santa Cruz, CA, USA, 2012. AAI3540841.

[13] Raph Koster. *Theory of Fun for Game Design*. PARAGLYPH PRESS, 2005.

[14] Pier Luca Lanzi, Daniele Loiacono, and Riccardo 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*, pages 1–8. IEEE, 2014.

[15] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Towards a generic method of evaluating game levels. In *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'13, pages 30–36. AAAI Press, 2014.

[16] 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*, 3(3):245 –259, sept. 2011.

[17] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[18] P. T. Ølsted, B. Ma, and S. Risi. Interactive evolution of levels for a competitive multiplayer fps. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1527–1534, May 2015.

[19] Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Optimization of platform game levels for player experience. In Christian Darken and G. Michael Youngblood, editors, *AIIDE*. The AAAI Press, 2009.

[20] Sebastian Risi, Joel Lehman, David B. D'Ambrosio, Ryan Hall, and Kenneth O. Stanley. Combining search-based procedural content generation and social gaming in the petalz video game. In Mark Riedl and Gita Sukthankar, editors, *AIIDE*. The AAAI Press, 2012.

[21] Noor Shaker, Georgios N. Yannakakis, and Julian Togelius. Towards automatic personalized content generation for platform games. In G. Michael Youngblood and Vadim Bulitko, editors, *AIIDE*. The AAAI Press, 2010.

[22] Maral Tajerian. Fight or flight: The neuroscience of survival horror. http://www.gamasutra.com/view/feature/172168/, 2012.

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

[24] Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis. Multiobjective exploration of the starcraft map space. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 265–272, 2010.

[25] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation. In *Proceedings of EvoApplications*, volume 6024. Springer LNCS, 2010.