Heuristics for Placing the Spawn Points in Multiplayer First Person Shooters

Marco Ballabio
Dipartimento di Elettronica,
Informazione e Bioinformatica,
Politecnico di Milano,
Email: marco1.ballabio@mail.polimi.it

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

***Abstract*—Level design in first person shooters is a critical and complex process with many facets. Among them, the placement of the spawn points, i.e., the positions of the level where players starts the game after being killed, has a huge impact on the game experience. In this work, we propose a novel approach to this problem that combines a set of design rules with a graph-based analysis of the level. As a results, we design a set of heuristics that, based on the topological features of the graphs extracted from the levels, can be used for the placement of spawn point. Finally, we test our approach with a small user study on three different levels, comparing our heuristic placement with an uniform strategy. Although preliminary, our results are promising and suggest that the level design principles are effectively captured by our heuristics.**

I. Introduction

First person shooters (FPS) are a very successful genre of videogames, where players navigate the game environment from a first person perspective and fight against either computer controlled characters or other players. These games allow to experience a high level of immersion and, thus, require a close attention to level design. Furthermore, the ever-increasing popularity of competitive multiplayer FPS added a new level of complexity to this design process in order to support different game modes, playing styles and interactions among players. Despite the importance of this process, this field is still lacking a well established common ground or a set of standards, whereas level design is still considered as an art, based only on the human creativity and previous experiences. However, in the past few years this subject attracted a lot of researchers that proposed both theoretical frameworks for the analysis of the design patterns [8] and several approaches to the procedural generation of levels [11], [5], [14], [2], [15]. Unfortunately, only few works focused on the problem of resources placement, although it is of extreme importance. In particular, the position of the spawn points, i.e., the locations where players appear after being killed, can significantly affect the players experience.

In this work, we propose a novel approach to the placement of spawn points that is inspired by the design principles identified in [17] on the basis of an analysis of map design and game dynamics of *Quake 2*[1]. To use this knowledge in practice, we introduce a set of graph-based representations for the FPS levels and then we map the design principles to some topological features that can be computed using these graph-based representations. Accordingly, we design a set of heuristics that can be applied to procedurally place the spawn points in a FPS map, on the basis of the analysis of graphs extracted from the FPS levels. Furthermore, to test our heuristics, we compared our heuristics with a uniform placement strategy, by carrying out a small user study on three different levels that involved 27 users . Although preliminary, the results are promising and show that our heuristics seems indeed able to capture the design rules listed in the literature.

The remainder of the paper is organized as follows. In Section II we discuss the related work. The underlying design principles used to design our heuristics are described in details in Section V. In Section IV we provide some background about different kind of map representation and we introduce a novel set of graph-based representations. In Section VI we report and discuss our experimental analysis. Finally, in Section VII we draw our conclusions.

II. Related Work

A few works in the game research literature studied the level design principles and patterns used in first person shooters (either single player or multiplayer). Güttler et al.[7] studied the spatial design principle of levels, usually called also *maps*, in a FPS. In their work, they identified so called *points of collisions*, i.e., areas of the map where the majority of players interactions happen. Larsen[12] analyzed three really different multiplayer games, *Unreal Tournament 2004*[2], *Day of Defeat: Source*[3] and *Battlefield 1942*[4], identifying shared patterns, evaluating their impact on gameplay, and suggesting some guidelines on how to use them. Hullett and Whitehead[8] identified some patterns for single player games, many of whom can be applied to a multiplayer setting. In [9], Hullett also analyzed the cause-effect relationships between some of these patterns and gameplay data collected with real players.

[1]Id Software, 1997
[2]Epic Games, 2004.
[3]Valve, 2005.
[4]DICE, 2002.

978-1-7281-1884-0/19/$31.00

Furthermore, in the past few years several works have been published on the procedural generation of maps for first person shooters, although most of them focused more on the layout of the map rather than on the placement of game elements. In a seminal work, Cardamone et al.[3] proposed an approach to generate maps for multiplayer first person shooters that are *fun* to play: they introduced the concept of *fight time*, that was defined as the time span from the moment in which two players engage in a shooting to the death of one of them, and argued that maximizing it lead to maps that are more fun to play. Instead, Lanzi et al.[11] focused on the generation of maps that are balanced for two players that have either a different playing style (e.g., are using two different weapons) or different skill levels. The same idea, was later extended by Loiacono et al. by introducing a multiobjective approach to generate maps with several design objectives at the same time [14] and by generating maps that lead to an emerging fleeing behavior [15]. Olsted et al.[16] focused on the generation of maps for teams of players, suggesting that the previous approaches do not generate maps suitable for this kind of game experience. Therefore, inspired by the design used in *Counter Strike*[5] and in *Call of Duty*[6], they devised a novel procedural generation process: maps are generated starting from a grid of nodes, adding connections among them iteratively (based on a set of design rules), and finally populating them with all the resources, such as spawning points, weapons, etc. Anand and Wong[1], proposed a search-based procedural generation approach that is able to generate a multiplayer maps in a few seconds. Their approach is based on an analysis of the topological features of the maps, such as the points of collisions [7], the connectivity between the regions of the map, the positioning of the control and spawn points. Cachia et al.[2] took the approach introduced in [3] a step further with the generation of multi-level maps and the placement of resources and spawn points. Liapis [13] proposed an iterative approach to the design of a map by means of interlocking rooms generated separately. Karavolos et al. [10] showed how a surrogate model of the interrelations between different types of content in the same game can be used for level generation. Finally, Giacomello et al. [5] trained a Generative Adversarial Network [6] to generate levels for *DOOM*[7].

III. Design principles for competitive FPS maps

One of the key concept underlying the map design of first person shooters is the so-called *level flow*. In single player experiences, it basically refers to how players naturally move through the game environment to reach the end of the level. In this case, the flow is affected by the aesthetics of the environment (e.g., a notable example of this is the use of colors in *Mirror's Edge*[8]), by positioning power-ups and items along the path, or by means or visual/audio cues that can catch the player attention (e.g., the dynamic flock of birds used in *Half Life 2*[9] as described in [4]). Instead, in multiplayer experiences the level flow generally refers to how the players interact with each other and with the game environment. In fact, in this case the flow is affected by the layout of the map and the placement of the resources: the more an area of the map is easy to navigate and offers tactical advantage, such as covers, resources or high ground, the more players will be comfortable moving in it; in contrast, an area with a *bad* flow, such as an area very exposed and difficult to navigate, will be generally avoided by the players. Accordingly, the design of multiplayer map aims at balancing the flow through different areas, e.g., an area with a bad flow could allow players to get access to a very powerful resource, such as a powerful weapon.

In this work, we focus on the positioning of spawn points, despite the proposed approach can be easily adapted also for placing other items or resources. Our approach is based on the analysis of the work of of Tim Schäfer [17], who has performed an in depth analysis of multiplayer 1vs1 maps in Quake 2. The core idea introduced in [17] is that the analysis and the balancing of the map flow, cannot be isolated from the analysis of the game dynamics. In fact, as soon as a player is killed during the game, the balancing is significantly affected. The killed player, dubbed as *down-player*, loose all the weapons, the ammunition, and armor that he has previously collected, while the player who survived, dubbed *up-player*, generally has stronger weapons and equipment. Therefore, the goal of the *up-player* is to gain and hold the control of the key areas of the map, i.e., the one that gives a tactical advantage and access to powerful resources, to kill as soon as possible the *down-player* keeping the game out of balance. In contrast, the *down-player*, needs to get access to powerful weapons and resources, to re-balance the game before facing again the *up-player*. Accordingly, the designer of the map should take into account this game dynamics both for what concern the layout of the map itself and for the placement of the spawn points, the ammunition, the resources, etc. In particular, Tim Schäfer describe few rules of thumb for the placement of resources and spawn points [17]:

(i) spawn points should be positioned in areas that are of low interest for the up-player and that are easy to leave, i.e., central hubs and dead ends are a bad choice, whereas rooms with 2 or 3 exits are usually the best option;
(ii) health packs are placed in zones that are safe or not too dangerous;
(iii) armors are usually placed in spots that are aimed both at the *down-player* and at the *up-player*, such that a small quantity of armor should be easy to achieve, whereas power-ups that provide full armor should be placed in dangerous areas;
(iv) mid-power weapons are of high interest for the *down-player*, since he needs to get one of them as soon as possible if he wants to face the *up-player*, so should be

[5]Valve Software, 2000
[6]Infinity Ward, 2003
[7]Id Software, 1993
[8]Digital Illusions CE, 2008.
[9]Valve, 2004.

placed in areas that are easy to reach and the same goes for their ammunition;

(v) very strong weapons should be placed in areas that are strategically disadvantageous, like dead ends or vertically dominated areas, or difficult to reach;

(vi) special weapons and power-ups that grant temporary advantages to the player who collects them, like invisibility or increased damage, should be placed in locations difficult to reach and contextual to their effect.

IV. Maps Representation and Analysis

Although in this work we do not deal with the generation of the layout of the maps, in order to define heuristics for the spawn points, we need to represent and analyze the whole map. Accordingly, in this section we describe how maps can be represented using different levels of abstraction.

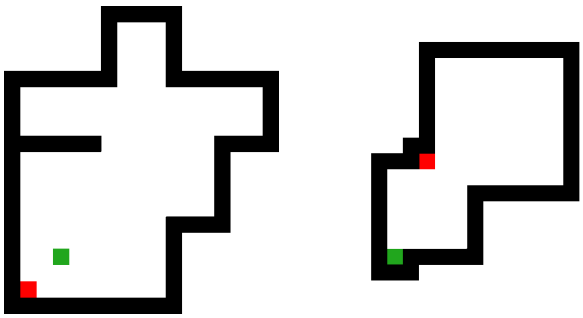
A. Tile-Based

Each first person shooter has its own internal map representation, that is usually the more convenient one for the engine of the game. As an example, *DOOM* basically employs a map representation based on vertices, sectors, and segments (see [5] for additional details). However, a quite standard representation that can suit the design of maps of several first person shooters, is the map representation based on *tiles*: the map space is organized as a regular grid of square tiles [10]; thus, each tile can be either empty, non-traversable (i.e. an obstacle, like a wall), or a game element (i.e., a tile containing a weapon, a power-up, a spawn point, etc.).

B. Indirect

An indirect representation is such that it represents the map in a more compact way and with a higher level of abstraction, i.e., it does not define the map *itself* but rather define how it can be generated. A representation of this kind widely used in the literature, is the *All-Black* representation introduced in [3]. It is built on a tile-based representation, but encodes only empty areas of an otherwise non-traversable map. The encoded areas consist of square rooms and corridors of fixed width. Rooms are defined by $\langle x, y, s\rangle$ triplets, where x and y define the coordinates of the center of the room and s defines its width. Corridors are rectangular areas with a fixed width and are defined by $\langle x, y, l\rangle$ triplets, where x and y define the point in which the corridor starts and l defines its length. In this work, we extended this representation as follows. For allowing the encoding of game objects (i.e., spawn points, weapons, etc.), we added a third kind of triplet, $\langle x, y, o\rangle$, that uses x and y to denote the coordinates of the tile that hosts the objects and o to denote the object itself, encoded as a character. In addition, to allow the representation of multi-level maps, we simply introduced a flight of stairs as game object and concatenated together the representation of each level of the map. Figure 1 shows two examples of maps along with their All-Black representation. We refer the interested users to the work of Cardamone et al. [3] for additional detail on the All-Black representation.

[10]Please notice that in games where vertical heights are very relevant for the map design, the tile-based representation can be easily generalized by organizing the map space as a regular 3D-grid of cubic tiles



(a) Map represented by $\langle 5,\ 5,\ 9\rangle\ \langle 10,\ 10,\ 7\rangle\ \langle 15,\ 25,\ 3\rangle\ |\ \langle 5,\ 15,\ 15\rangle\ \langle 11,\ 15,\ -7\rangle\ |\ \langle 5,\ 5,\ s\rangle\ \langle 7,\ 7,\ h\rangle$.

(b) Map represented by $\langle 1,\ 2,\ 5\rangle\ \langle 4,\ 6,\ 8\rangle\ |\ \langle 5,\ 6,\ -10\rangle\ \langle 10,\ 15,\ -6\rangle\ |\ \langle 3,\ 7,\ s\rangle\ \langle 1,\ 1,\ h\rangle$.

Fig. 1: Two examples of maps represented with the All-Black encoding: the character | is used to separate rooms, corridors, and game object triplets; s and h represent respectively a spawn point and an health pack.

C. Graph-Based

Another interesting approach to represent a map is by means of a graph. With respect to the previously described approaches, this representation is probably not very convenient to generate a map, but it represent the topological properties of the map more effectively. Accordingly, in this work we introduce several graph-based representations that can be used to represent and analyze different properties of the maps.

1) Outlines graph: This graph is extracted directly from the All-Black representation of the map and is obtained by associating a node to every vertex of every room and corridor and by connecting the non-adjacent ones that belong to the same outline. This graph has a single kind of node (*vertex node*) that contains the coordinates of the tile it represents, which are used to position the node when the graph is visualized. Figure 2b shows an example of this graph. This graph can be used to visualize the rooms which compose the map.

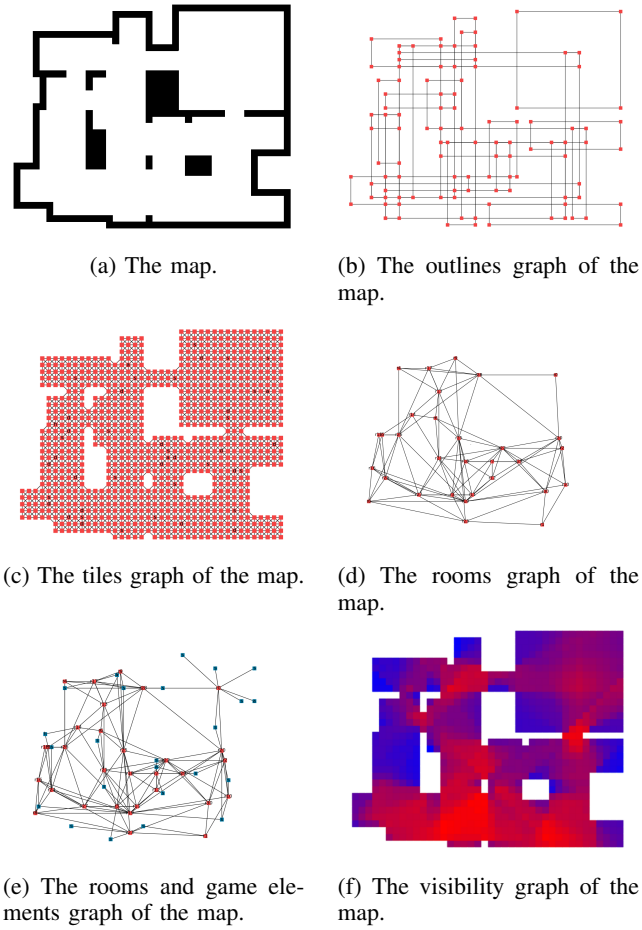
2) Tiles graph: This graph is extracted from the tile-based representation of the map and is obtained by associating a node to each empty tile and by connecting each node to its 8-neighbors. The horizontal and vertical edges have cost 1, whereas the diagonal ones have cost $\sqrt{2}$. This graph has a single kind of node (*tile node*) that contains the coordinates of the tile it represents, which are used to position the node when the graph is visualized. Figure 2c shows an example of this graph. This graph can be used to find the minimum distance that separates two cells, along with the shortest path that connects them.

3) Rooms graph: This graph is extracted from the All-Black representation of the map and is obtained by associating a node to each room and corridor and by connecting nodes

to corresponding rooms or corridors overlap, using as weight the Euclidean distance of their central tile. This graph has a single kind of node (*room node*) used to represent both rooms and corridors that contains the coordinates of the closest and furthest vertex of the room/corridor from the origin. When visualized, each node is positioned on the coordinates of the center tile of the room/corridor it represents. Figure 2d shows an example of this graph. This graph can be used to analyze the topology of a map, in order to find loops, choke points, central areas and other kind of structures.

4) Rooms and game elements graph: This is an extension of the rooms graph, which also includes game elements as nodes, that are connected to the nodes corresponding to the rooms and corridors which contain them. In addition to the room node inherited from the rooms graph, this graph has a node to represent game elements (*element node*) that contains the coordinates of the game element, which are used to visualize the node, and the character associated to it. Figure 2e shows an example of this graph.

5) Visibility graph: This graph is extracted from the tile-based representation of the map and is obtained by associating a node to each empty tile and by connecting each node to all the tiles that are visible from that node. For two tiles to be respectively visible, it must be possible to connect them with a line without crossing any non-traversable tile. This graph has a single kind of node (*visibility node*) that contains the coordinates of the tile it represents, which are used to position the node when the graph is visualized, and its *visibility*, which is computed as the *degree centrality*, i.e. the number of edges incident to that node. A tile with high visibility allows to control a wide portion of a map, but at the same time an entity standing on it is easy to spot. This graph can be visualized more conveniently by using an heatmap where a sequential colormap is used to encode the *visibility* attribute of each node, as shown in Figure 2f. This graph can be used to analyze which areas of the map are more exposed and which ones are more sheltered.



(a) The map.

(b) The outlines graph of the map.

(c) The tiles graph of the map.

(d) The rooms graph of the map.

(e) The rooms and game elements graph of the map.

(f) The visibility graph of the map.

Fig. 2: An example of map with all the different graph-based representations extracted from it.

D. Metrics

Once graph-based representations have been generated for a map, it is possible to compute several interesting metrics from them that provide useful information about the map, such as:

- *Degree centrality*: defined for a node, it is the number of edges that the node has. If the node represents a room, it measures how many entrance or exits the room has.
- *Closeness centrality*: defined for a node, it measures its centrality in the graph, computed as the sum of the lengths of the shortest paths between the node and all other nodes in the graph. If the node represents a room, it measures how central the room is.
- *Betweenness centrality*: defined for a node, it measures its centrality in the graph, computed as the number of shortest paths connecting the nodes that pass through the node. If the node represents a room, it measures how central the room is.
- *Connectivity*: defined for a graph, it is the minimum number of elements (nodes or edges) that need to be removed to disconnect the remaining nodes from each other. If the graph represents a map, it measures the existence of isolated areas.
- *Eccentricity*: defined for a node, it is the maximum distance from the node to all other nodes in the graph. If the node represents a room, it measured how isolated the room is.
- *Diameter*: defined for a graph, it is the maximum eccentricity of its nodes. If the graph represents a map, it measures the size of the map.
- *Radius*: defined for a graph, it is the minimum eccentricity of its nodes. If the graph represents a map, it measures how distanced the rooms are from each other.
- *Periphery*: defined for a graph, it is the set of nodes with eccentricity equal to the diameter. If the graph represents a map, it defines its peripheral areas.
- *Center*: defined for a graph, it is the set of nodes with eccentricity equal to the radius. If the graph represents a map, it defines its central areas.
- *Density*: defined for a graph, it ranges from 0 to 1, going

from a graph without edges to a complete graph. If the graph represents a map, it measures how complex it is.

V. Placement of Spawn Points

Following the guidelines described previously in Section III, we designed an heuristics for placing the spawning point in a map that is basically based on two principles: the *safety*, i.e., the spawn point should not be placed in a dangerous position for the *down-player*, and the *uniformity*, i.e., the spawn points should be uniformly distributed across the map. In particular we designed two distinct heuristics to be used in sequence to place a spawn point, the first is used to select a room inside the map and the second to select a tile inside a room. Overall, the placement of the spawn points is performed iteratively, i.e., one spawn point at once.

A. Room Selection Heuristics

This heuristic combines two terms that are designed to take into account the principles of safety and uniformity. Accordingly, given the rooms and game elements graph of a map G_{rr} (see Section IV) and the subset of nodes corresponding to the rooms $R \subseteq G_{rr}$, the spawn point is placed in a room (r^*) selected as follows:

$$r^* = \arg\max_{r \in R}(w_D \cdot D(r) + w_{H_e} \cdot H_e(r)), \tag{1}$$

where w_D and w_{H_e} (in the range $[0, 1]$) are the weights of terms $D(r)$ and $H_e(r)$, that are defined as follows:

$$D(r) = \begin{cases} 0 & \text{if } \deg(r) = 1 \\ 1 - \dfrac{\deg(r) - \min_{r'} \deg(r')}{\max_{r'} \deg(r') - \min_{r'} \deg(r')} & \text{if } \deg(r) \neq 1 \end{cases}, \tag{2}$$

$$H_e(r) = \min_{n \in G_{rr}} \begin{cases} 1 & \text{if } n \notin S \\ \dfrac{\mathrm{d_{sp}}(r, n)}{\mathrm{diam}(G_{rr})} & \text{if } n \in S \end{cases}, \tag{3}$$

where $\deg(n)$ is the connectivity degree of the node n in the room graph node, $\mathrm{diam}(G_{rr})$ the diameter of the graph G_{rr}, and $\mathrm{d_{sp}}(n, m)$ denotes the length of the shortest path that connects the two nodes n and m, found using Dijkstra's algorithm. Equation 2 promotes rooms with few passages but that are not dead ends, whereas equation 3 promotes rooms that are distant from the already placed game elements. Both terms are in the range $[0, 1]$. In all the experiments performed in this work, the weights w_D and w_{H_e} have been empirically respectively set to 1 and 0.5.

B. Tile Selection Heuristics

This heuristics combines three terms based on the principles of safety and uniformity. Given G_v the visibility graph and the subset of tiles ($T \subset G_v$) in the room r^* selected with the previous heuristics, the spawn point is placed in the tile t^* selected as:

$$t^* = \arg\max_{t \in T}(w_v \cdot v(t) + w_{h_w} \cdot h_w(t) + w_{h_e} \cdot h_e(t)), \tag{4}$$

where w_v, w_{h_w} and w_{h_e} (in the range $[0, 1]$) are the weights of terms $v(t)$, $h_w(t)$, and $h_e(t)$, that are defined as follows:

$$v(t) = 1 - \frac{\deg(t) - \min_{t' \in G_v} \deg(t')}{\max_{t' \in G_v} \deg(t') - \min_{t' \in G_v} \deg(t')}, \tag{5}$$

$$h_w(t) = \frac{\mathrm{d_{wall}}(t, r^*)}{\max_{t' \in T} \mathrm{d_{wall}}(t', r^*)}, \tag{6}$$

$$h_e(t) = \begin{cases} 0 & \text{if } |H| = 0 \\ \min_{h \in H} \dfrac{\mathrm{d}(t, h)}{l_d} & \text{if } |H| > 0 \end{cases}, \tag{7}$$

where $\mathrm{d_{wall}}(n, r)$ is the distance of the coordinates associated to the node n from the walls of the room r, computed as the sum of the minimum distances from the horizontal and vertical walls, l_d is the *diagonal* length of the room, and $\mathrm{d}(n, m)$ is the distance of the coordinates associated to the nodes n and m. Equation 5 promotes tiles with low visibility, equation 6 promotes tiles that are distant from the walls, whereas equation 7 promotes tiles that are distant from the already placed game elements inside the room. All three terms have values in the range $[0, 1]$. In all the experiments performed in this work, the weights w_v, w_{h_w} and w_{h_e} have been empirically respectively set to 1, 0.5, and 0.5

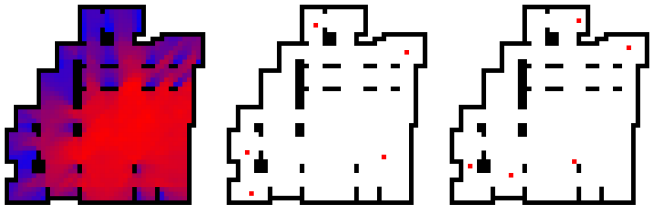
VI. Experimental Analysis

To test our heuristics we designed a simple experimental analysis that involve a few users that are asked to explore the maps while performing a simple task. To carry out this experimental analysis we developed a simple experimental framework with Unity [11], a very popular game engine. In the remainder of this section, we briefly describe our experimental framework, our experimental design, and our results.

A. Experimental Framework

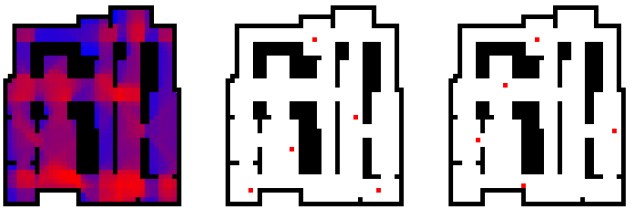
To perform our experimental analysis, we developed our own first person shooter environment using Unity. Despite being more simple than a proper first person shooter, such as *Cube 2: Sauerbraten* used in several previous works [3], [11], [14], it can be deployed using the WebGL technology and, thus, it can be played simply using a browser. This was extremely useful to organize the user study, because allowed to let users play online on their own machine. Although, our framework does not implement any bot to play against, it can load and generate maps with several representations, it also offers several weapons as well as several game modes, based on searching and destroying targets around the map. The framework also allows to collect a large amount of data

[11] https://unity.com/



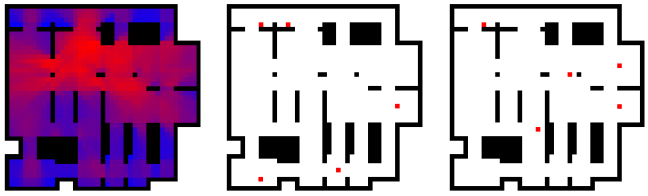
(a) Heatmap showing the visibility of the level. (b) Spawn points (in red) placed using the heuristic approach. (c) Spawn points (in red) placed using the uniform approach.

Fig. 3: *Arena* map used in the experiment.



(a) Heatmap showing the visibility of the level. (b) Spawn points (in red) placed using the heuristic approach. (c) Spawn points (in red) placed using the uniform approach.

Fig. 4: *Corridors* map used in the experiment.



(a) Heatmap showing the visibility of the level. (b) Spawn points (in red) placed using the heuristic approach. (c) Spawn points (in red) placed using the uniform approach.

Fig. 5: *Intense* map used in the experiment.

for later analysis. Collected data include the identifier of the map used, the identifier of the study, the game mode, a list and description of (i) all the spawn events, (ii) the shots, and (iii) the targets destroyed. In addition, the framework allows also to collect the answers of the users to a set of previously designed questions.

B. Experimental Design

To test our heuristics, we designed a task, called *Target Hunt*, as follows. The player needs to find and destroy as many targets as possible in a given amount of time. The targets spawn, one at once, in a location of the map chosen randomly among a set of spawn points generated in advance. Therefore, the aim of this task is to evaluate how easy the spawn points are spotted and reached by the players.

The study was performed using three different maps: *Arena*, *Corridors*, and *Intense*. For each map, two placement of the spawn points are compared: one using the heuristics described in Section V and one using a simple uniform distribution. Figure 3, Figure 4, and Figure 5 show the visibility of the three

TABLE I: An overview of collected data. Each value reported is an average of all collected samples.

		Total	Arena	Corridors	Intense
Samples		27	10	9	8
Targets	Heuristic	10.06	10.75	10.44	8.75
	Uniform	11.88	12.27	10.67	12.75
AvgSearchTime	Heuristic	20.24	21.09	18.25	21.40
	Uniform	16.06	15.50	17.46	15.18
AvgSearchDist	Heuristic	73.34	64.21	70.90	84.90
	Uniform	60.70	55.24	68.13	58.72

levels and the spawn points. It can be noted that the three maps have very different layouts. *Arena* consists of a wide arena, two sides of which are adjacent to parallel corridors with many openings; the central area allows to control most of the map, whereas the corridors offer some cover. *Corridors* is made of many small rooms connected by long corridors; there is no area that allows to control the others and the only points with high visibility are the corridors intersections. *Intense* is a mix of the previous ones, with both open areas and small rooms connected by corridors. Each user plays a *Target Hunt* for three minutes on one of the three maps two times: once with the heuristic placement of the spawn points and once with the uniform one. The two playing sessions are presented in random order and the map is flipped in one of two sessions.

Finally, the performance of the players are assessed computing the *AvgSearchTime*, i.e., the average time needed for the user to find a target, computed as the duration of the session divided by the number of targets found, and the *AvgSearchDist*, i.e., the average distance covered by the user to find a target. In addition, at the end of the two playing sessions, each user is asked to evaluate which one was more difficult (players are also allowed to answer that the two sessions were equally difficult).

C. Results

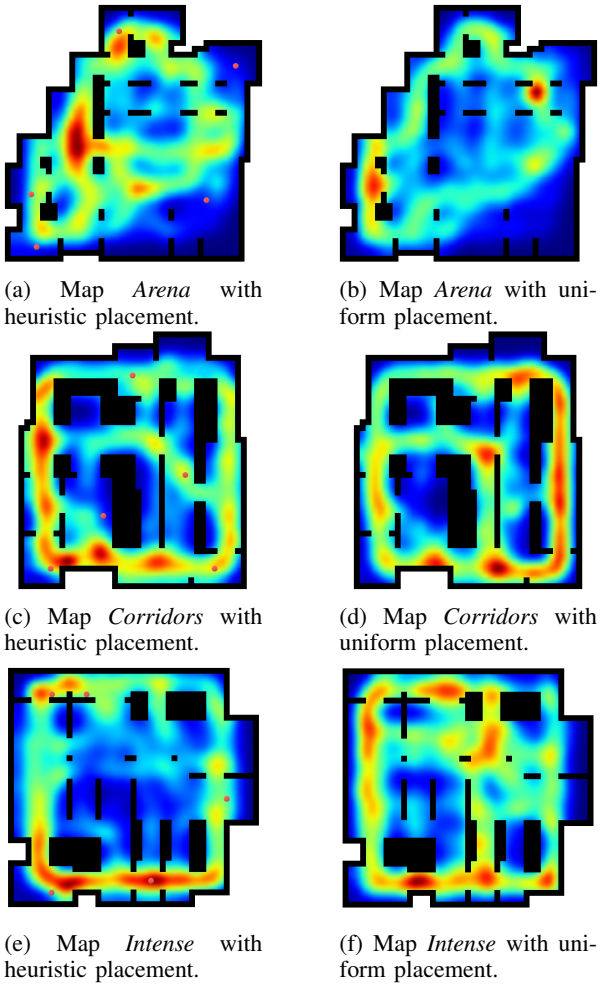
We collected data from 27 users, 10 users played in map *Arena*, 9 in map *Corridors* and 8 in map *Intense*. Table I compares the data collected during the experiments using the heuristic placement and the uniform one. The table reports the average number of targets destroyed (*Targets*), the average distance covered (*AvgSearchDist*) and the average time passed (*AvgTimeDist*) between the targets; data are reported for all the collected samples as well as grouped for each map. Results show that both *AvgSearchTime* and *AvgSearchDist* values are larger, thus the number of destroyed targets is smaller, when the heuristic placement is used with respect to the uniform placement. These results confirm what we expected: our heuristics is indeed able to place the spawn points in locations of the maps that are more difficult to spot and to reach following the normal flow of the map. Overall, the difference is rather large, corresponding to an increase of roughly 26% and 21% respectively of time required and distance covered by the players to find the next target. The analysis of data grouped for each map, shows that the differences are smaller

for the *Corridors* map and larger for the *Intense* map, due to their different layouts. In fact, the central area of *Arena* allows to control all of the surroundings, so the placement of spawn points in areas that are less visible has a very relevant effect. *Corridors*, instead, has a regular structure and the number of intersections between its corridors is almost always the same, so it is not so relevant where spawn points are placed, since all the rooms have similar features. Finally, the tangled structure of *Intense* offers to the heuristic approach a lot of interesting spots where to place spawn points and the presence of an area that allows a partial control of the map makes this choice even more meaningful. As a result, this analysis suggest the our heuristics has a more pronounced impact on maps with a non uniform layout.

To test the statistical significance of this result, we performed the non-parametric Wilcoxon signed-rank test by *Pratt*[12], comparing the *AvgSearchTime* and *AvgSearchDist* values collected when using heuristic placement and when using uniform placement. According to the test, the differences of both these values are statistically significant (with respectively a $p - value$ of 0.00203 and 0.01243).

The position of spawn points also influenced the way in which users moved across the maps. Figure 6 shows, for each map and for each placement strategy, the heatmaps of time spent by the players in the different areas of the maps. It is possible to notice that users, once understood the topology of the map, started to follow well defined *farming routes*[13]. These routes tend to be circular and to skirt the perimeter of the maps, with deviations that are influenced by the position of spawn points.

It is also interesting to notice how the difficulty in finding the targets was perceived by the users. As it can be seen in figure 7, in the survey the majority of the users subjectively evaluated more difficult the session where they performed worst (i.e., where they destroy less targets), but some of them evaluated such the one where they performed better.



(a) Map *Arena* with heuristic placement.

(b) Map *Arena* with uniform placement.

(c) Map *Corridors* with heuristic placement.

(d) Map *Corridors* with uniform placement.

(e) Map *Intense* with heuristic placement.

(f) Map *Intense* with uniform placement.

Fig. 6: Heat maps of the player positions for the three maps used in the experiment. The red circles represent spawn points.

VII. Conclusions

Few works in the past focused on the placement of spawn points in a first person shooter map. In this work, we proposed a novel approach to this problem that is inspired to the analysis of the *up-player* vs *down-player* dynamics described in [17]. To this purpose, we introduced a family of new map representations based on graphs that can be easily extracted from the usual tile-based representations used in several previous works. Then, we used such graph-based representations to design an heuristic strategy for placing the spawn points on the map according to the design principles suggested in [17]. Finally, we tested our approach with a rather small user study which involved 27 players that performed a simple search and destroy task on three different maps. The collected data suggested that our placement strategy proposed is actually able to identify areas of the map that are indeed less easy to spot and to reach, as prescribed by the design rules it was designed after. Although very preliminary, our results are promising and suggest that our graph-based representations allow to identify and design better topological design patterns.

In the future, we plan to the extend our experimental framework with AI controlled bots and with online multiplayer, that would allow to design experiment with a setting more similar to a real first person shooter. Future works also include the design of placement strategies for additional game elements, such as the weapons and the health packs, as well as an experimental comparison with other placement strategies, such as the one used in [2].

[12]With respect to the standard Wilcoxon test, the one by Pratt considers also the observations for which the difference of the elements in the pair is zero. We opted for this approach since some samples happen to have metrics with the same value for the two placements.

[13]In video games, *farming routes* are regular closed paths defined to maximize the collection of certain resources in a specific map.

References

[1] Anand Bhojan and Hong Wei Wong. Arena - dynamic run-time map generation for multiplayer shooters. In Yusuf Pisan, Nikitas M. Sgouros, and Tim Marsh, editors, *Entertainment Computing – ICEC 2014*, pages 149–158, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

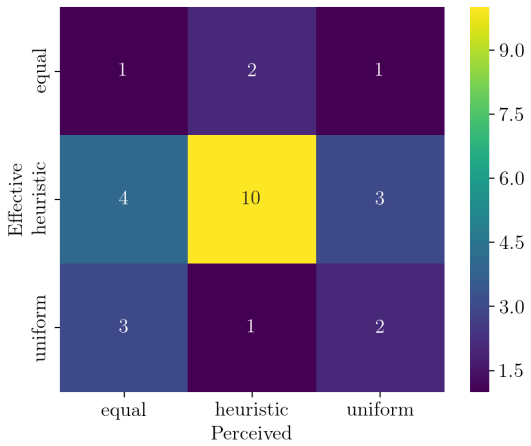


Fig. 7: Comparison of the placement strategy that resulted objectively more difficult for the player (reported on the rows) with respect to the one that was *perceived* as more difficult for the player (reported on the column). Each square reports the number of samples for each possible combination. The objective evaluation of the difficulty is based on the number of targets destroyed by the players, where *equal* is for ties.

[2] William Cachia, Antonios Liapis, and Georgios N. Yannakakis. Multi-level evolution of shooter levels. In *AIIDE*, 2015.

[3] 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.

[4] Matthew Gallant. Guiding the players eye. http://gangles.ca/2009/05/26/guiding-the-eye/, 2009.

[5] Edoardo Giacomello, Pier Luca Lanzi, and Daniele Loiacono. DOOM level generation using generative adversarial networks. In *IEEE Games, Entertainment, Media Conference, GEM 2018, Galway, Ireland, August 15-17, 2018*, pages 316–323. IEEE, 2018.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[7] Christian Güttler and Troels Deg 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.

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

[9] Kenneth M. Hullett. The science of level design: Design patterns and analysis of player behavior in First Person Shooters levels, 2012.

[10] D. Karavolos, A. Liapis, and G. N. Yannakakis. Using a surrogate model of gameplay for automated level design. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, Aug 2018.

[11] 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*, pages 1–8, Aug 2014.

[12] Simon Larsen. Level design patterns. http://simonlundlarsen.com/wpcontent/uploads/2015/06/Level-design-patterns.pdf/, 2006.

[13] Antonios Liapis. Piecemeal evolution of a first person shooter level. In Kevin Sim and Paul Kaufmann, editors, *Applications of Evolutionary Computation*, pages 275–291, Cham, 2018. Springer International Publishing.

[14] D. Loiacono and L. Arnaboldi. Multiobjective evolutionary map design for cube 2: Sauerbraten. *IEEE Transactions on Games*, 11(1):36–47, March 2019.

[15] Daniele Loiacono and Luca Arnaboldi. Fight or flight: Evolving maps for cube 2 to foster a fleeing behavior. In *IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22-25, 2017*, pages 199–206. IEEE, 2017.

[16] Peter Thorup Olsted, Benjamin Ma, and Sebastian Risi. Interactive evolution of levels for a competitive multiplayer FPS. In *IEEE Congress on Evolutionary Computation, CEC 2015, Sendai, Japan, May 25-28, 2015*, pages 1527–1534. IEEE, 2015.

[17] Tim Schäfer. Designing great 1vs1 FPS maps. https://dfspspirit.wordpress.com/2015/03/26/designing-great-1vs1-fps-maps-part-1/, 2015.