CITATION: Kruse, J., Connor, A.M. & Marks, S. (2021) An Interactive Multi-Agent System for Game Design. Computer Game Journal. https://doi.org/10.1007/s40869-020-00119-z

Abstract

This paper presents a novel approach to procedural generation of game maps for multi-player, competitive video games. A multi-agent evolutionary system is employed to place streets, buildings and other items, resulting in a playable video game map. The system utilises computational agents that act in conjunction with the human designer to produce maps that exhibit desirable characteristics. This paper compares the impact that the additional agents have in terms of the quality of candidate solutions. The results indicate that the use of the agents produces higher quality solutions in comparison to a traditional interactive genetic algorithm.

Keywords: Interactive Genetic Algorithms, Procedural Content Generation, Game Design.

1 Introduction

First Person Shooter (FPS) games, particularly those that would be considered action games, have been one of the most successful and fastest growing genres in the computer games industry (Cifaldi 2006; Predescu and Mocanu 2020). Titles such as *Battlefield*, *Insurgency*, *Halo*, *Call of Duty* and many others have emerged as successful franchise that sell millions of copies every year. Creating maps for FPS games that exhibit characteristics that support the emergence of particular gameplays is a challenging and laborious task (Hullett and Whitehead 2010). This task is usually achieved manually which ensures that the map is both playable and matches the intent of the designer. Urban environments are particularly challenging as they are characterized by high complexity. The large number of individual items such as houses, streets, cars or similar obstacles and smaller props pose huge challenges for game designers. This paper outlines a novel semi-automated approach to urban map design, which draws on interactive evolutionary computation and autonomous agents to produce fully playable FPS game maps. The approach has the potential to discover novel map designs that may not have emerged when relying on the creativity of the human game designer alone.

2 Background and Related Work

Procedural content generation (PCG) is a popular application for creative computational systems (Liapis et al. 2014) and as a process it involved the generation of games content automatically using algorithms (Yannakakis and Togelius 2018). PCG has been used since the 1980s to produce such algorithmically generated content for video games, and the game *Akalabeth* is often cited as the first game where the game world was generated procedurally (Amato 2017). Since then, a wide variety of commercial games have successfully utilised PCG methods, and the techniques have gained new momentum in recent years due to the availability of computational resources in form of faster CPUs and general-purpose computing on GPUs (Mark et al. 2015).

Several studies have already reviewed the history of PCG in both analogue and digital games (Amato 2017; De Carli et al. 2011; Hendrikx et al. 2013) and so a comprehensive historical review need not be repeated. In general, PCG is often used when there is a need to produce large volumes of game content (Short and Adams 2017). PCG has been used in developing a wide range of game content, such as maps, adventures, characters, weapons, planets, plants and histories (De Carli et al. 2011). Such content can be generated through many different algorithms and some of them are built on methods from artificial intelligence (AI) or computational intelligence (CI), for example, evolutionary computation and constraint satisfaction (Togelius et al. 2013; Yannakakis and Togelius 2011). Such approaches can be considered as evolutionary procedural content generation, which is a subset of the search based approach to PCG (Togelius, Yannakakis, et al. 2010).

2.1 Evolutionary Procedural Content Generation

In evolutionary PCG, an evolutionary algorithm such as a genetic algorithm is used to search for content with desirable qualities. In general, these approaches are iterative and refine candidate solutions over time until they eventually arrive at a suitable solution (Togelius et al. 2011). Evolutionary PCG has been used to generate different types of necessary content such as puzzles (Smith et al. 2012), tracks (Loiacono et al. 2011), maps (Togelius, Preuss, et al. 2010) and terrains (Doran and Parberry 2010) to name just a few applications.

Evolutionary algorithms have extremely wide-ranging applicability and have been used to generate different types of game content, where a suitable fitness function can be determined to assess the quality of a solution candidate.

In general, three approaches are used for formulating a fitness function, namely an explicit calculation of fitness, simulation based fitness functions, or interactive functions (Yannakakis and Togelius 2011).

When the fitness is explicitly calculated or determined through simulation, the intention is to replace the human game designer and fully automate the design process. Whilst a worthy goal, reliance on fully automated approaches have been shown in some cases to produce games that are less engaging than manually designed content (Connor et al. 2017). Therefore, whilst automated approaches offer benefits typical of applying computational intelligence approaches to complex cognitive tasks, such as faster completion times when compared to a human map designer, they are not able to capture the human creativity that produces the most interesting map designs, and they are usually very specific to their particular domain (Mawhorter and Mateas 2010). For this reason, there is considerable interest in semi-automated solutions that work in conjunction with the human designer in such a way that it combines the benefits of both approaches.

2.2 Interactive Genetic Algorithms

The Interactive Genetic Algorithm (Takagi 2001) approach is a variation of a normal evolutionary approach that is often applied where there is difficulty in defining a fitness function. In this approach, the human user performs the selection of candidate solutions that are then evolved using genetic operators. This concept has been extended by (Kosorukoff 2001) in an approach known as a human based genetic algorithm (HBGA), where the genetic operators are also implemented by human agents rather than algorithmically. Central to this approach is a decentralised reconceptualisation of Genetic Algorithms using a multi-agent implementation that can potentially include both human and computational agents. This poses a paradigm shift, where selection and recombination are separated and performed by different entities (Kruse 2019).

Such implementations have been used to successfully counteract user fatigue, one of the implicit problems of Interactive Evolutionary Systems when applied to creative design tasks (Kruse and Connor 2015). A machine learning classifier is trained using the designer's input over a number of generations to gauge the aesthetic preference and capture the design intent. An agent based on this classifier provides additional (computational) selections, which reduces the selection runs for the human in the HBGA. This general approach is extended in the work described in this paper.

2.3 Evolving First Person Shooter Game Maps

In this paper, procedural or generative design denotes the automatic or semi-automatic generation of game maps, including elements such as the terrain, position of buildings, streets and other items. Instead of manually placing game assets into the map, significant parts of this task are automated. Evolving game maps for First Person Shooter game, also often called FPS levels, is not an entirely novel idea. For example, the work by Cardamone et al. (2001) incorporated map generation using evolutionary algorithms, with the main difference that their work used a reasonably simple mathematical fitness function based on the average fighting time of the player plus the free space on the map. The fitness function is not only purely theory-driven rather than modelled on data-driven player capabilities and preferences, but it also reflects a purely player-centric approach, which excludes the game designer's intent. Here the computational agent is the single designing entity.

Another recent example for procedurally generated FPS maps is work conducted by Lanzi et al. (2014) which also utilizes an evolutionary algorithm and hence falls under the umbrella of search-based procedural content generation (Togelius, Yannakakis, et al. 2010). Their work aims to evolve game maps that provide basic match balancing properties based on player skill and strategies.

While there are some similar works which have been conducted, for instance by Cardamone et al. (2001) in context of FPS content generation, and also by Cook and Colton (2014) for general procedural 3D game design, this approach differs in that it heavily considers the (human) game designers intent and feeds this deliberate choice back into the automated selection process of the evolutionary algorithm.

This goal has been addressed to some extent by work in other application areas. For example, Cardamone et al. (2011) have utilized an Interactive Genetic Algorithm (IGA) in the design of race tracks for a racing game. Similar

approaches have been adopted for terrain generation (Walsh and Gade 2010), game textures (Yoon and Kim 2012) and specific game assets (Provano et al. 2015). To date there has been little focus on utilizing an IGA in the development of game maps for multiplayer FPS games, though early iterations of the research described in this paper would fall into this category (Kruse et al. 2016).

As a result of high design and development costs, FPS games offer only a few hours of single-player gameplay and oftentimes just a small number of multiplayer maps (Cardamone et al. 2001). The research outlined in this paper offers a semi-automated approach that should allow map designers to produce a higher number of maps to significantly reduce costs, which has the potential to boost the popularity of a new title as a means to commercial success.

3 Multi-Agent System for Game Map Design

This paper presents a multi-agent system approach for designing maps to be utilized in multiplayer FPS games. This approach is based on an interactive genetic algorithm that not only integrates a human agent into the candidate selection process, but also a number of computational agents that are intended to improve the design process. The details of the game and the design prototype that incorporates the implementation of the multi-agent system are outlined in the following sections.

3.1 Game Overview

For this research project, the maps are implemented in a similar vein to a number of commercial games, such as *Counter-Strike: Global Offensive*, *Insurgency* and *Insurgency: Sandstorm.* The game is set in a deserted urban environment without any non-player characters (NPCs). It is not fully post-apocalyptic but rather deserted in a state that suggests the occupants of these places had only recently left. The buildings are intact, and the streets are only slightly cluttered from the aftermath of whatever caused the population to leave. The mode of game play is a simplified conquest-type game mode with a single flag point.

The maps produced by the design system consist of a few core game design elements that follow conventional FPS map designs, and which relate directly to game elements as introduced by Järvinen (2009). These elements are considered important to gameplay and game mechanics, as their design choice has a direct impact on how the players interact in the game environment. The elements relevant to the game and a description of their implementation in both the game and corresponding design tool are given in Table 1.

Table 1 Game Design Elements

Game Design Element	Description
Terrain	The terrain in the final maps was simplified and reduced to a flat playing field. Generally, terrain with small and large elevations provides cover by breaking line of sight between players. While this would have been an option for the prototype, in terms of game engine capability and design approaches found in a number of commercial FPS games, elevated terrain seemed to pose an obstacle to what this study seeks to understand. In order to present game designers with a quick-to-understand, simplified view of an entire game map, with tens or hundreds of corners and possible angles for line of sight and cover, a top down approach was chosen for the design tool.
Streets	Streets are an element that can be considered purely cosmetic, as these elements do not provide any cover or obstacles to players. Parts of the terrain that are not occupied by buildings, containers, or street elements are still fully walkable. However, streets are still included as they act as landmarks. In the case of this prototype, the street was a single path, essentially defining the overall shape of the playable map.
Buildings	Buildings were simplified to a square in the design prototype and in the resulting maps for a single reason: they served as semantic elements for obstacles. In order to focus gameplay at the street level, and not the building level, the implementation did not include walkable buildings (with an interior).

Flag pole	The prototype maps only comprised a single game mode, i.e. a variant of the 'conquest mode', with only one flag to capture. One team has to occupy the vicinity of the flag pole to defend it, while the opposing team attempts to get close to it for a specified time of 20 seconds in order to capture it. The area, which either blocks the capture or initiates the capture, is roughly one street element, or one block wide.
Shipping containers	Shipping containers provide small cover in street areas. Stylistically justified by the post-apocalyptic theme of the maps, where random obstacles clutter the streets of an urban area, they spawn primarily on street elements.

Figure 1 shows a top-down view of a single candidate solution in the map design system that is representative of the type of maps that can be produced. The large area that appears empty is simple terrain into which street elements, buildings, containers, the flag, and spawn points can be placed. In this case, the candidate solution has a relatively short path length, as well as the very close proximity of spawn points to one another and the flag pole. As a result, the entire play area only occupies a small part of the available terrain

The green and red borders indicate the two team spawn points and the flagpole is visualised by the round red circle in the middle of a street element. The algorithm always places the flag inside a street element in order to keep the flag in the focus area of the map, and it is surrounded by landmarks such as street markings, buildings, and containers. It is common in FPS games to place the flag in an area that does not have only open areas surrounding it, in order to avoid direct line of sight. Not providing any cover will lead to very poor gameplay, as opposing team members will be able to kill each other at a long distance. Evidence of this potential flaw in map design – which has been avoided in this instance by keeping the main elements in close proximity – can be seen in older releases of *Battlefield*, *War Planes*, and similar, very open large-scale maps. This flawed design could potentially lead to both teams 'camping', rather than pursuing the intended goal of the game.

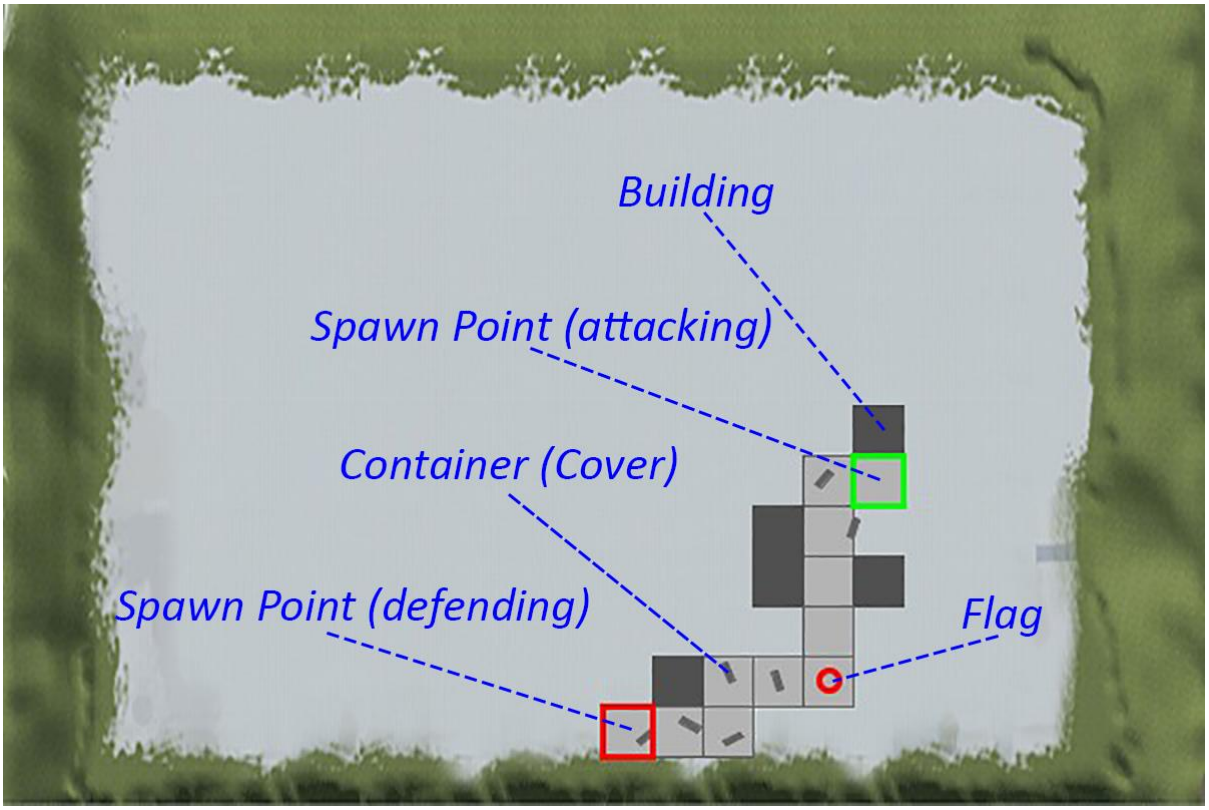

Figure 1 - Candidate map with annotations.

The dark grey squares in Figure 1 represent building placeholders. While the building assets in the final map are not exactly fully square, and do not always perfectly fill a full square, the difference between this simplified view and the resulting game map is marginal.

Shipping containers, being an element that provides additional, very effective cover due to their position in the middle of streets, and because of their odd angles with regard to the buildings and relatively linear position of the street elements – are shown as smaller grey rectangles. Figure 2 shows a three-dimensional rendering of the map example in Figure 1 to illustrate how the map translates into an actual game.

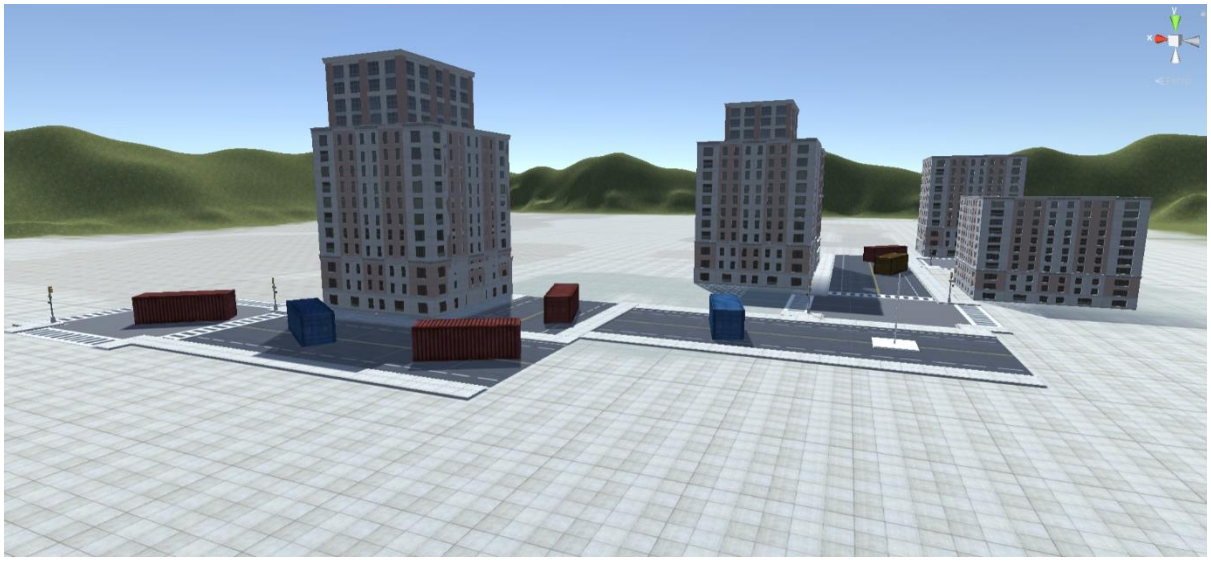

Figure 2 - Three-dimensional rendering of the map from Figure 1

While the overall cluster of elements may appear reasonably small, it is important to understand that each cell representing a building or street element is $25m^2$. Accordingly, running at fast speed from end to end will take roughly one minute, which makes this example map in Figure 1 a similar size to typical medium-sized maps in *CS:GO* or *Insurgency*. This consideration is important, as the game mode is entirely based on infantry without vehicles, with a maximum of eight players in the map, which in turn can lead to very few encounters in a significantly larger map, based on our initial gameplay tests. Few encounters were considered less engaging and less motivating by test players. Therefore, the size presented in this average example also represents a size considered 'engaging' and 'fun' by participants.

3.2 System Components

The map design in this study is loosely based on the human-based genetic algorithm framework by (Kosorukoff 2001) which allows a human user to act as an integral part of the genetic algorithm as part of a multi-agent system. The intent is to augment the abilities of map designers and allow designers to explore different options through the implementation of a number of computational agents. If the user is simply employed to control a genetic algorithm by selecting the candidates for breeding, without the computational agents being used, the resulting system will simply be an interactive genetic algorithm (Takagi 2001).

The system incorporates two agents that are responsible for map analysis, creating diversity among the potential candidate solutions in each population of the genetic algorithm, and applying design knowledge extracted from human expert statements about their process. These agents work in conjunction with a human map designer to create a playable FPS game. The overall architecture of the system is shown in Figure 3.

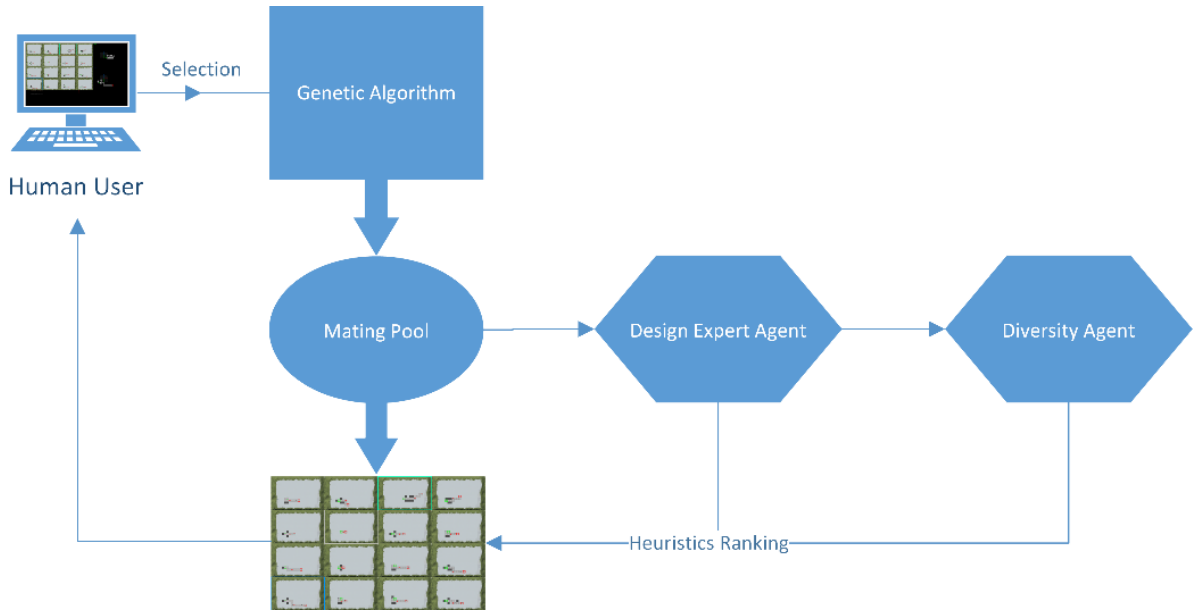

Figure 3 - Schematic overview of the multi-agent system.

The system has no mathematical fitness function that selects candidates for the next mating pool (as in the case of a canonical genetic algorithm) but uses the output of several agents to evaluate fitness of the candidate pool and individual candidates (see Figure 4). It actively selects two candidates based on metrics learned from the documented processes of expert designers.

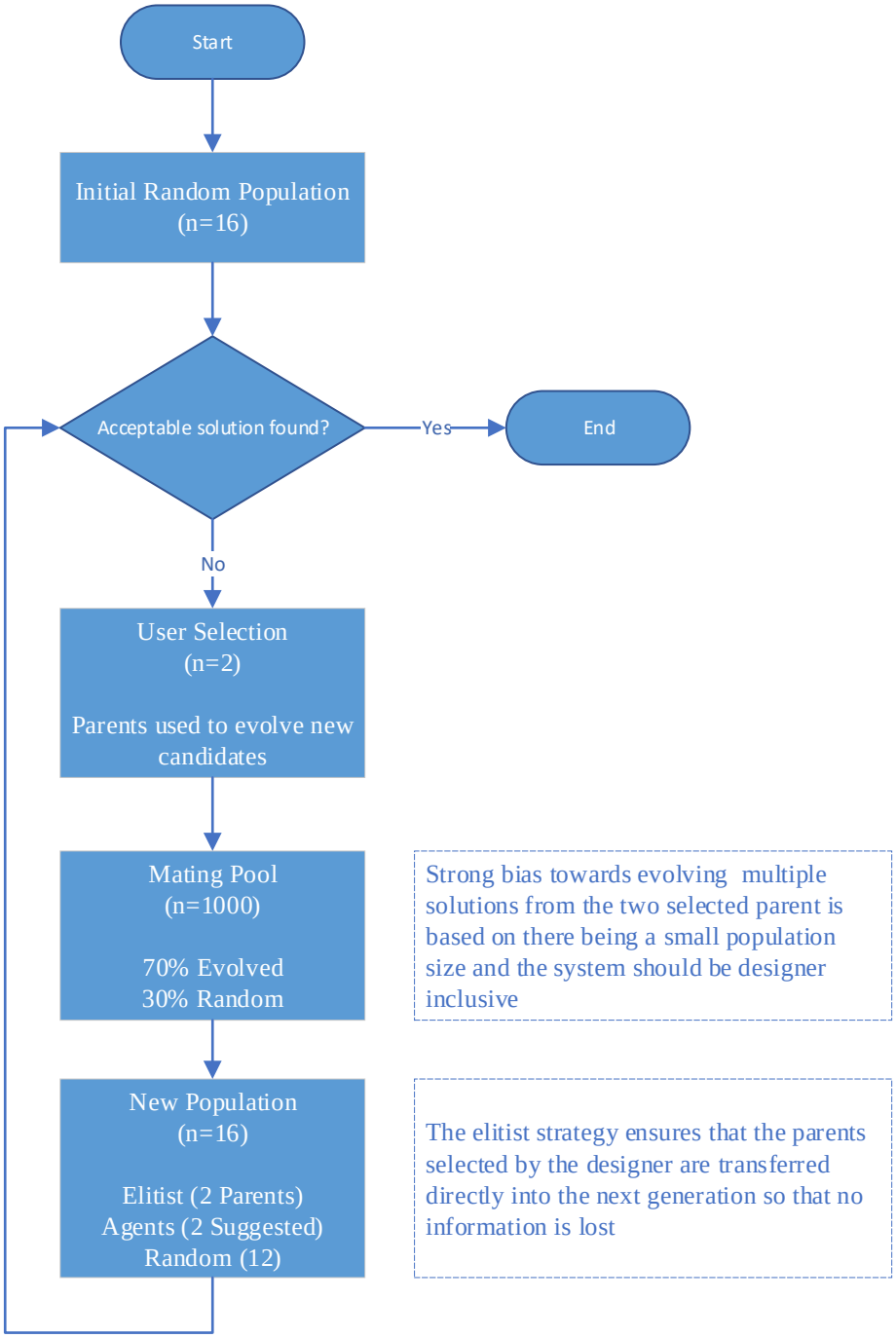

Figure 4 - Overview of interactive evolutionary process employed in this study.

It is ultimately the designer who makes the selection, regardless of whether the agents deem the user selection the fittest option. While this will potentially lead to the selection of candidates that are not the fittest by numerical measures, it provides the designer with some flexibility. Rather than being locked in by metrics, the designer is able to change course for the final result by intentionally selecting any solution, regardless of its fitness.

The genetic algorithm of this system employs an elitist strategy, which means that both parents, selected by the human agent (the map designer), are put straight back into the next population without any adaptation. The main reasoning for employing an elitist strategy is that any subsequent population should not be less fit than the previous one. Given that the designer may not have chosen the numerically fittest candidates, but perhaps any of the other candidate solutions for breeding the new population, this may not hold true. However, it ensures that the human designer views their selection from the previous population in the current selection, in case the genetic algorithm

did not recombine these two candidates into any acceptable new solutions. This measure was taken after initial tests showed that participants were sometimes confused by not seeing any solutions that were (subjectively) at least 'as good' as the previous selection they made. Presenting the user with exactly the same two solutions that were selected in the previous run, alongside 14 new candidates based on recombination, and including mutation, removed this perceived issue with the system.

In summary, any bred population of 16 candidates consisted of the two parents selected by the designer, two candidates selected from the breeding pool by computational agents based on their evaluated fitness, and 12 candidates randomly selected from the breeding pool.

3.3 Map Encoding

The Genetic Algorithm used in this study employs value encoding instead of canonical binary encoding, to ensure that properties of map elements such as street, building, flagpole, spawn point and container on street are kept intact through crossover and mutation. The chromosome for each candidate of a population is implemented as an array list with individual genes as list elements. The genes are the street elements that form the path including all relevant information as shown in Figure 5. The chromosomes are initialised with random properties for each gene and a random path length between 3 and 10 elements.

Given that the path length varies from candidate to candidate, the implementation of the value string as a list enables varying path length by adding further elements into the list. Each element represents a street section, starting with the red spawn point

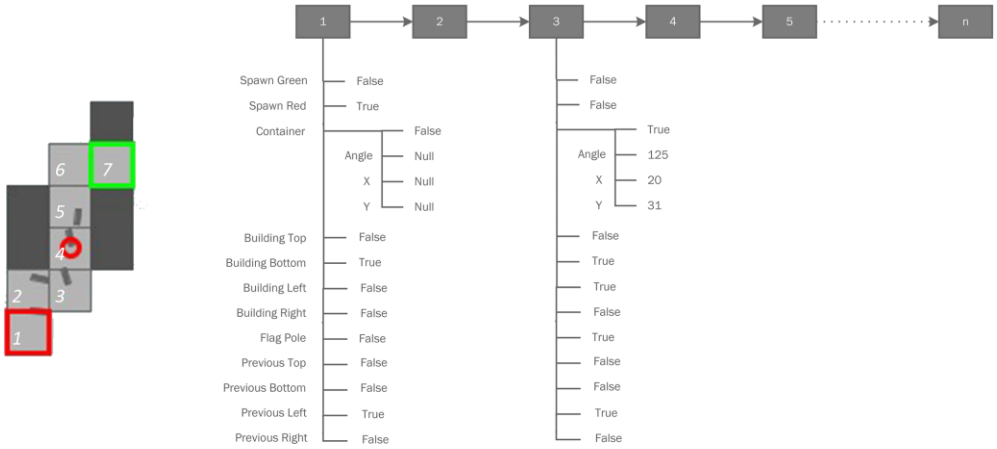

Figure 5 - Chromosome contains encoded values for each street element

The gene holds information about the direction of the neighbouring street elements, any neighbouring buildings, whether they contain a shipping container as obstacles and how the shipping container is positioned and rotated for each street element in the list. Further, information whether it is one of the two spawn points or the flagpole is stored.

3.4 Genetic Operators

The crossover used in this study is a single point crossover chosen by a probability test against each path element of the first parent, which in turn is also randomly picked from the two parents submitted for recombination by the user. Owing to the variable path length, another random point is probabilistically determined for the second parent. Figure 6 illustrates how recombination is performed.

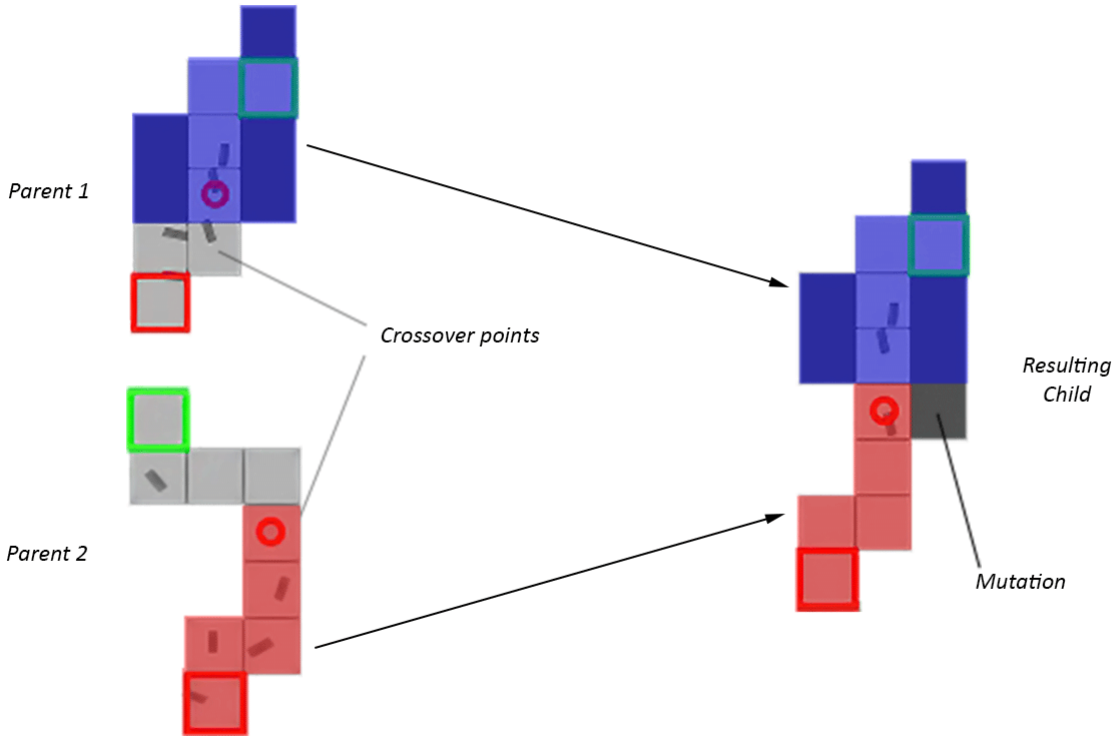

Figure 6 - Recombination including crossover and mutation

Finally, a simple mutation strategy is implemented where each of the chosen elements is potentially altered, for example adjacent buildings added or removed, path direction changed (and subsequent elements moved into their new cells) or containers removed, added or simply rotated and repositioned.

In a final step, the path is re-centred on the terrain to avoid the path creeping out of the play areas through further recombination, and the flagpole re-centred to the middle of the entire path.

3.5 Agents

The current design system uses two additional computational agents, both of which are involved in the process of selecting candidate solutions for breeding in the IGA. The two agents are identified as the design agent and a diversity agent.

3.5.1 Design Expert Agent

The design expert agent (DEA) is the core of the computational agent system. It models a (human) map designer, as suggested by Zhu et al. (2017), and applies high-level, abstract concepts to FPS maps in order to evaluate their quality. It runs independently in the system, utilises digital differential analysis (Museth 2014) to find intersections with solid obstacles from its current position, tests metrics pertaining to the relationship between both spawns and the flag pole, and assesses the entire breeding pool in preparation for candidate selection. Furthermore, it actively selects the two candidate solutions with the highest ranking from the breeding pool and feeds them into the 16 candidates of the next population.

The methods used for developing designer heuristics is similar to the work of Pinelle et al. (2008), who took a qualitative approach to find usability heuristics for video games. The heuristics that were implemented in the DEA were derived as a result of a qualitative analysis of expert game designer opinions available in the public domain and are described in Table 2.

Table 2 DEA Heuristics

Name of Heuristic	Description
Balance of cover and line-of-sight (H1)	FPS map design experts identified a balance between sufficient cover and some areas that provide direct line of sight between competing teams as one of the crucial key elements that drives gameplay. There must be enough cover so that opponents are able to escape shots taken from a long distance. Most players do not have the skill to take long shots, and as such, the opportunity to move towards the goal of the map (in this case, the flagpole capture area) is increased. Otherwise teams may fall into a lockdown with neither moving forward, known as 'camping' (Wright, Boria & Breidenbach, 2002), which leads to the next heuristic.
Remove camping hotspots (H2)	Hotspots (also called 'choke-points' by some experts) can be enjoyable areas where a balance between cover and line of sight is present. Camping hotspots however, are sections of a map where players exploit the hotspot to gain kills, rather than play to win the match (Wright et al. 2002). Removing these camping hotspots has been identified as a key element for immersive gameplay. Expert designers also pointed to better playability on several occasions as a result of these camping areas being removed.
Distance to first cover from spawn points (H3)	The third heuristic is again based on areas of the map that provide cover to players. When the match starts, both teams presumably rush towards the capture point, but given sufficient map knowledge, the teams also know where first contact with the other team is likely to occur. Therefore, in competitive matches, teams generally only rush toward those first cover points, and take a more strategic play approach to avoid getting killed too quickly. The distance to the first cover point (or in some cases, the time it takes a player to run to the first cover point, depending on the run speed implemented by the developer) is another key heuristic for good, playable maps.
Distance between spawn points (H4)	Linked to the above, the overall distance between both spawn areas defines the time players need to make first contact. However, given that there is generally more than one possible pathway that can be taken towards the capture point, for example, navigating left or right around a larger obstacle such as a building (or shipping container), the point for first clashes, as highlighted in (3) varies, depending on the path taken by the individual player. Experts identify the overall distance between spawns as another important metric.
Distance from spawn points to capture point (H5)	Finally, the distance between each spawn point and the capture point is important, assuming that they are not exactly equal in distance, because then the flag pole/ capture point would essentially be the first point of contact between teams anyway. Measuring the distance between spawn and capture point for each team across different possible pathways allows for making a statement about balance between spawns. If one team gains a significant advantage because their spawn is next to the capture point, the map may not gain significant popularity, as it may be considered playable, but not very fair. This is particularly true if the spawn of the attacking team is significantly closer to the flag than the defending team, in which case the attackers will likely not face too much resistance before reaching the capture point. Expert designers consider a near equal distance across multiple pathways as ideal, and this was implemented as another heuristic in the designer expert agent.

For the DEA to be able to select between candidate solutions, it is necessary to find a way of numerically assessing each candidate in order to rank the mating pool. In theory, the calculation used by the DEA could be used to guide the entire search in an automated process, however the efficacy of this is not evaluated in this research. Instead, the fitness calculation of the DEA is used to select a candidate for the population after recombination and mutation operators have been applied to the mating pool. However, it is important to note that this is not an implementation of the genetic operator of selection. The high fitness here simply means that it will be presented to the human designer. The human designer will ultimately decide whether this highly fit candidate will make it back into the next mating pool. In this way, the human designer stays in full control of the selection process, and the designer expert agent merely makes a recommendation.

The fitness calculation performed by this agent is based on digital differential analysis and utilises the measurements given in Table 3.

Table 3 Map Measurements

Measurement	Description
M1	Visible obstacles in the direction of the flag from the green spawn point
M2	Visible obstacles in the direction of the flag from the red spawn point
M3	Visible obstacles directly between the green and red spawn points
M4	Distance from green spawn point to first visible obstacle
M5	Distance from red spawn point to first visible obstacle
M6	Distance from flag to nearest obstacle in the direction of the red spawn point
M7	Distance from flag to nearest obstacle in the direction of the green spawn point
M8	Distance between the two spawn points
M9	Distance between red spawn point and flag
M10	Distance between green spawn point and flag

It is worth noting that the calculation of the number of obstacles is not simply a summation of the number of obstacles along a given path. Instead, the agent ray casts from the four corners of a cell to each of the four corners of the target cell and calculates the number of obstacles that would be visible to a player in the initial cell. The intention is to determine whether there is a clear line of sight between the specified cells. For example, Figure 7 shows the boundary of the calculation of M2 to determine the extent to which there is a clear line of sight from the red spawn point to the flag location for a candidate solution. This boundary is a simplification of the total number of raycasts that are undertaken to perform the calculation, as in practice there are four ray casts from each corner.

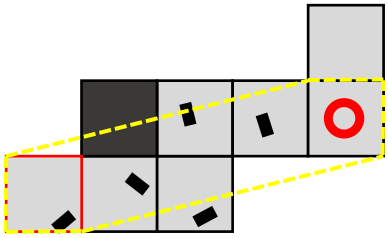

Figure 7 Obstacle Detection Example

The calculation of the number of obstacles between flag and both spawn points is based on the notion that a large number of obstacles will provide sufficient cover, whereas a low number of obstacles create open spaces. The first heuristic (H1) states that a balance of cover and open space is important for effecting engaging gameplay. The agent divides the number of obstacles between flag and spawn by the distance between flag and spawn. Therefore, if the map is larger than average, a larger number of obstacles is required to gain a high score for balance between cover and line-of-sight. If, on the other hand, the map is very small, a small number of obstacles is sufficient for providing balance that leads to a high fitness score.

$$H1 = \frac{\left(\frac{M1}{M10}\right)_{normalised} + \left(\frac{M2}{M9}\right)_{normalised}}{2}$$

Calculation of distances between certain points of interest such as the flag pole and nearby obstacles, as well as number of obstacles between spawns and the overall distance, is based on the notion that this distance represents open space, without cover to remove camping hotspots (H2), and to protect the spawn for each team. A larger distance means a lower score, and a smaller distance provides a higher score.

$$H2 = \frac{\left(\frac{1}{M6}\right)_{normalised} + \left(\frac{1}{M7}\right)_{normalised} + \left(\frac{M3}{M8}\right)_{normalised}}{3}$$

The normalised measurements of distance between spawns and first cover (M4 and M5), distance between spawns (M8), and finally, distance from spawns to flag (M9 and M10), provided calculated values for heuristics H3 to H5.

Finally, the agent calculates an average between these measurements, which results in a score between 0.0 and 1.0 as each of the heuristics calculations has already been normalised. This average represents the fitness score where a higher score indicates a higher quality candidate.

$$fitness\ score = \frac{H1 + H2 + H3 + H4 + H5}{5}$$

3.5.2 Diversity Agent

This agent has some similarities to the DEA, in that it acts on the DEA's knowledge, but applies it in a different way. It seeks to identify maps that are not represented in the population shown to the human agent, but nonetheless have a high degree of fitness, while being very different to the solutions proposed by the DEA. For example, if the DEA selects maps that are predominantly horizontal, the DA will seek out maps that are vertical, while maintaining a high level of fitness. However, fitness is the parameter it compromises on first – the most important goal for the DA is to identify maps that are different, maps that provide some degree of diversity to the population. The main idea here is to avoid the entire multi-agent system from getting stuck in a local maximum, that is, the state where the user can only select from very similar maps and therefore, cannot get out of the local maximum themselves.

To avoid creating an agent that simply selects the weakest candidates from the breeding pool, the diversity agent seeks to find a candidate where only one property is significantly different from the highest ranked solution according to the designer expert agent. For example, path length may be very different, or one team may have much less cover than in the strongest candidate. All other parameters are kept at a high ranking. The diversity agent selects the one trait that it seeks to vary in its selected candidate, based on a random decision from generation to generation. It is not the parameter that is ranked the highest by the designer expert agent that drives this decision by default. The reasoning for this is to avoid including only the most obvious counterpart of the highest ranked candidates, and instead create actual diversity.

3.6 System usage

The map design in this research is concerned with procedural generation which supports a human designer to create first-person shooter maps. The aim is to allow human designers to create a larger number of maps than would be possible by using a fully manual map editor. The motivation behind this goal is to potentially reduce the cost of game asset (levels/maps) creation, reduce the time needed to create such assets, and to increase the variety of first-person shooter maps that a human designer is able to create within a certain timeframe. Cost of game asset creation has been identified as a source that can reduce the profitability of a computer game, which in turn cuts into the profit a game company is able to make (Koster 2018). Time-to-release of a game or additional game maps are also factors that impact on a computer game company's ability to remain competitive (a shorter design cycle can lead to more frequent content releases). Reducing and simplifying game content creation can also help designers to create more iterations, which in turn may improve playability and ultimately, the success of a first-person shooter game.

The usage of the system typically starts with the random generation of a population of 16 candidate maps that form the initial generation. The creation of the next generation is enacted through a combination of the designer's choices and the input of the playability and diversity agents. The designer reviews the initial generation of solutions and selects the two candidates that best captures the design intent for intended gameplay. In this regard, the human designer is acting as a selection agent. The system therefor consists of a single human user (the designer) and multiple computational agents to control the underlying genetic algorithm of the design system. The human designer acts as the selection operator by selecting two parents within a population of 16 candidates. These candidates are presented as simplified top-down versions of a fully playable game map. An example of the user interface is shown in Figure 8, where two candidates have been selected and the corresponding 3D visualisation of the game maps visible on the right hand side of the screen.

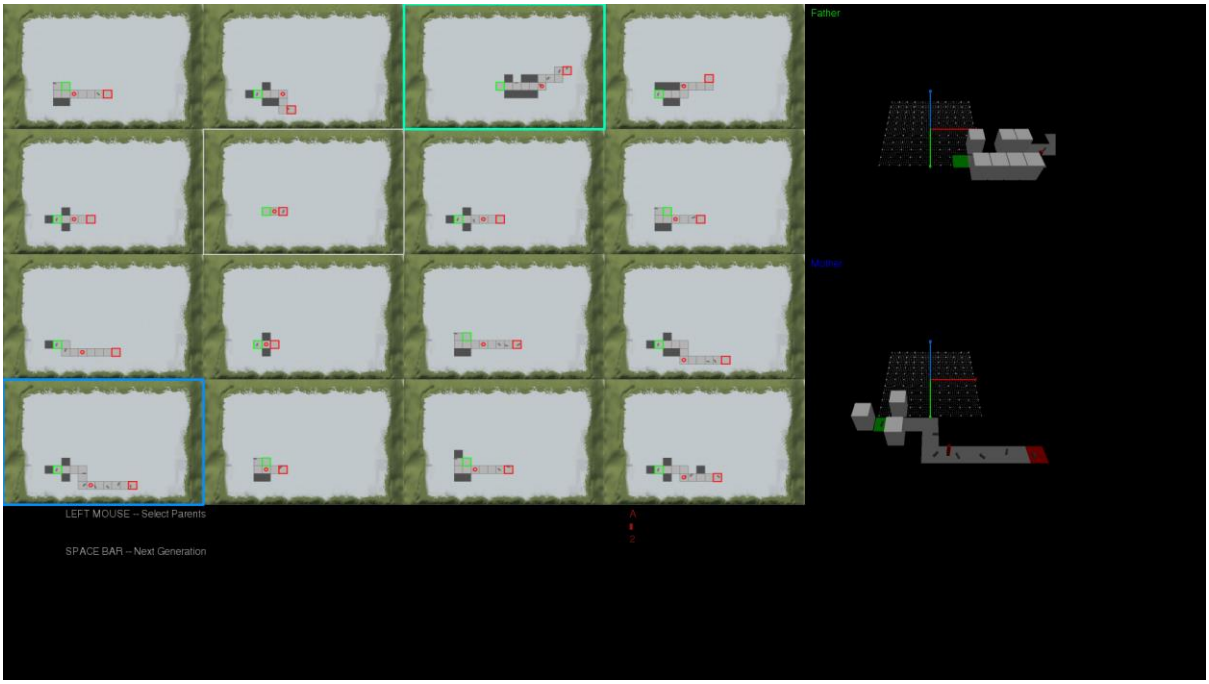

Figure 8 GUI example for the prototype tool

From this selection, a candidate pool is produced that is used to create the next population of solutions. In addition, the two selected parents are immediately copied into the next population as a form of elitism. The candidate pool is very much larger than then the population presented to the designer and is formed by breeding the two selected parents multiple times so as to produce enough children to form 70% of the candidate pool. The remaining 30% of the candidate pool is produced by breeding randomly selected candidates from the population.

The next generation presented to the user is formed in the following way. Firstly, the playability agent scans the entire candidate pool and determines the two maps with the features that are mostly likely to produce the desired gameplay according the rules embedded within the agent. These two solutions are added to the two elitist candidates already in the next population. Next, ten solutions are randomly selected from the candidate pool for inclusion. The final two solutions are selected by the diversity agent. The agent scans the candidate solutions already selected and compares them to solutions in the candidate pool in order to select two candidates that exhibit different features in order to maintain diversity in the population.

This process is repeated until the game designer is satisfied that they have a map that will promote the gameplay that they expect to achieve. The final map is then automatically imported in to the Unity3D game engine for the production of the final game.

4 Results

This section presents the results of the map generation tests with professional game designers. The least experienced designer had roughly two years of game design experience, whereas the most experienced game designer had been active in the industry for more than 21 years. This experience included working on titles in the Counter Strike series and other AAA titles. The results presented in this paper focus on the efficacy of including the additional agents in the system and therefore compare how the quality of solutions in the breeding pool change over time, based on a number of trials. In total, eleven game designers participated in the study and each undertook two test runs using the system, once with the agents activated and once purely as an Interactive Genetic Algorithm. The designers were not aware of when the agents were active.

The fitness ranking is computed by the design expert agent to allow agents to filter the breeding pool and is not used as a fitness function as such. However, the ranking data of the breeding pool over time provides insight to the efficacy of the system, which would be exhibited by an increase in fitness over time.

Figure 9 shows how the fitness rank of the population varies over time for eleven runs of the system without the agents active. As number of iterations in each run was different, varying from short runs with only 12 generations to longer runs with76 generations, the length of the runs has been normalised to 100 data points, using linear interpolation.

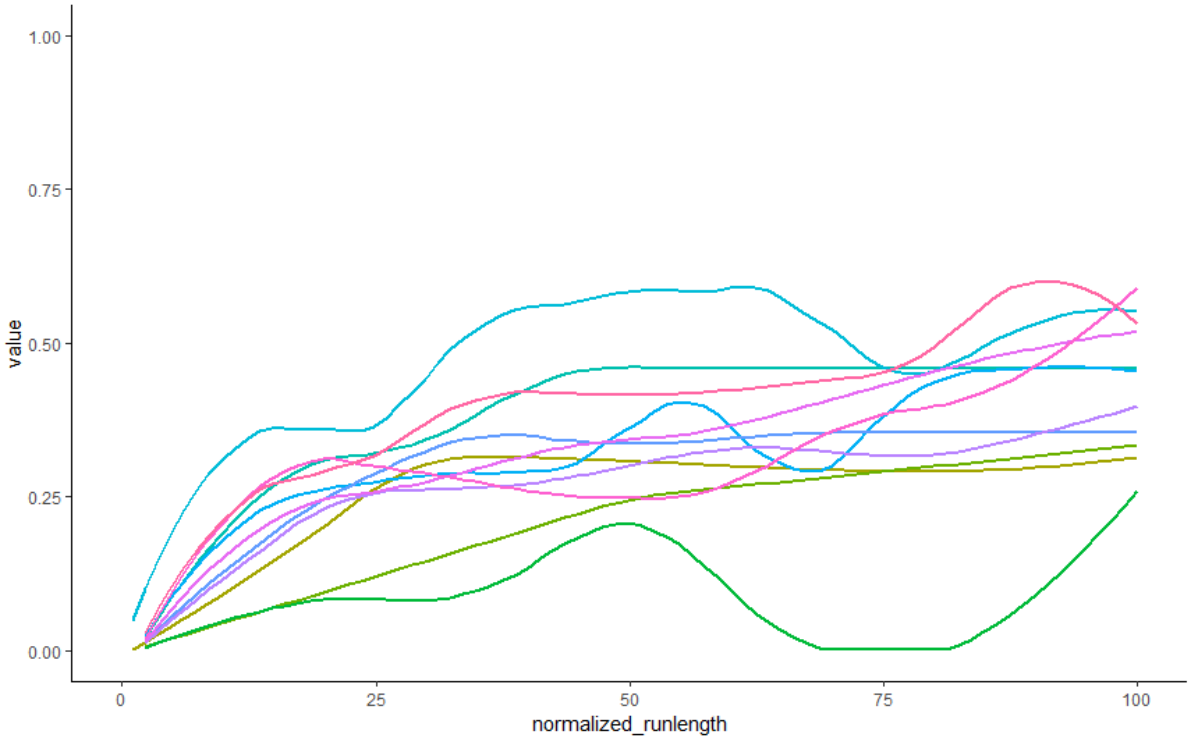

Figure 9 - Maximum fitness value, agents inactive. Each colour represents a different run for easier visual reading.

In general, the results show that the quality of the breeding pool increases over time which suggests even without the computational agents active that there is potential value in the interactive genetic algorithm. It is worth noting that this is not a graphing of cumulative increases in fitness, but the absolute fitness of each generation. As a result, several instances show that there is a decrease in fitness of the mating pool that is later corrected. Such variations may arise from either the choices of the designer in a given generation or the stochastic nature of the algorithm. Because the algorithm is elitist in nature, these variations in fitness do not necessarily imply that good solutions have been lost in the process. In comparison, Figure 10 shows the corresponding data for when the agents were active.

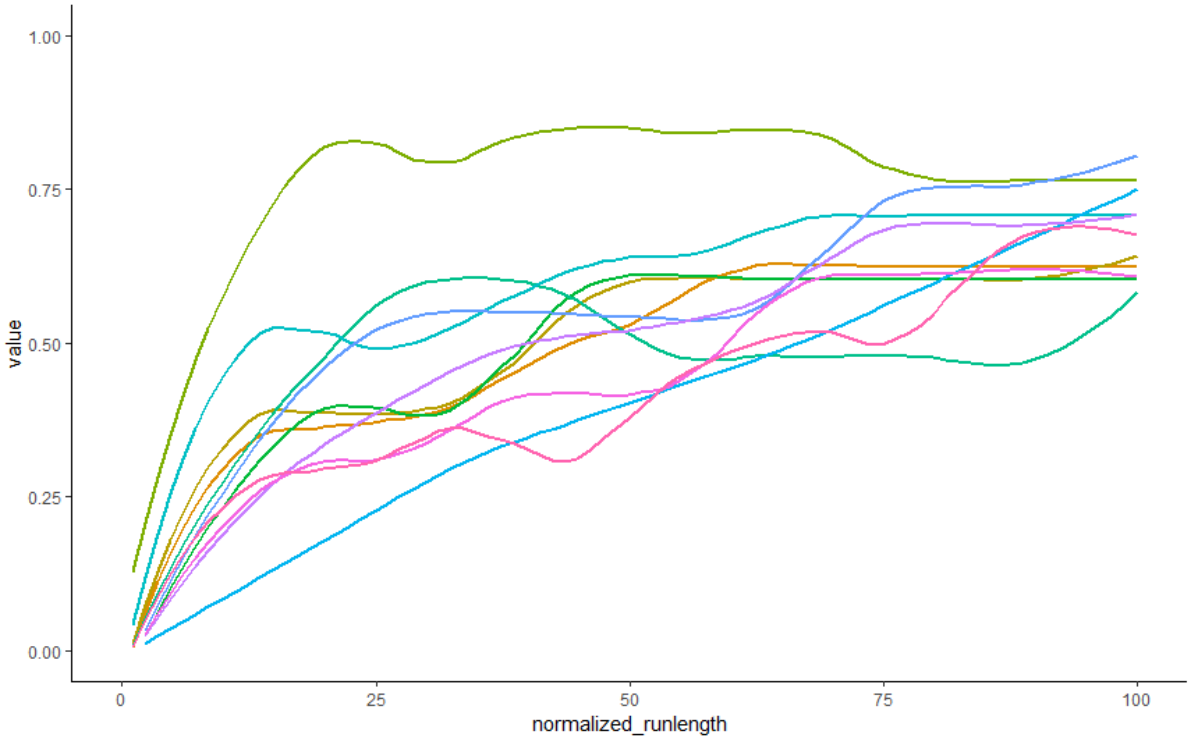

Figure 10 - Maximum fitness value, agents active. Each colour represents a different run for easier visual reading.

Despite the variance between low performing runs and high performing runs, it appears that in general, an overall higher performance was achieved when the agents were active. This can be confirmed by considering the average of all data points of all runs with agents, and the same for all runs without agents being active. This is shown in Figure 11.

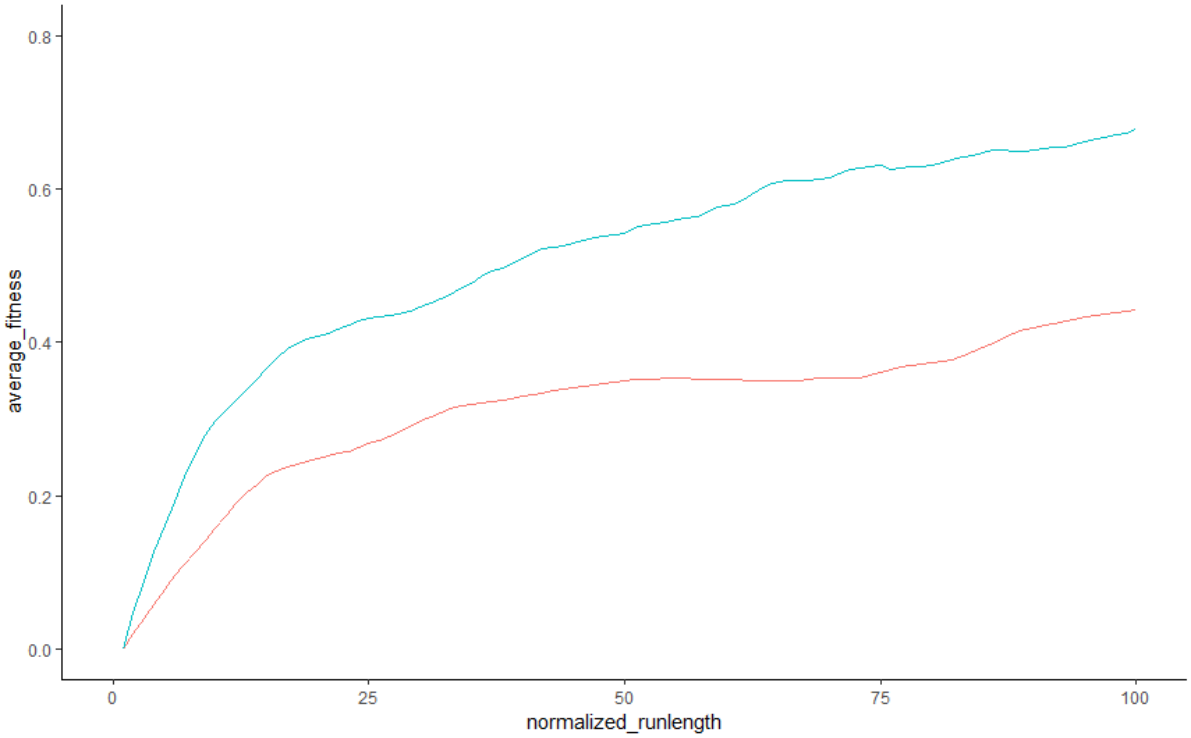

Figure 11 - Maximum pool fitness of all runs combined. The blue line signifies 'with agents' and red line 'without agents'.

The blue line represents the mean across all runs with active agents, while the red line shows the mean of all runs without agents. The agents appear to have a significant impact on how fast the fitness increases. With agents, a much higher average pool fitness is evidenced than without agents.

5 Discussion

The comparison of performance between when the agents were active and not active provides some evidence to support a claim that the agents are effective in improving the overall quality of the final solutions. It is worth noting that there was a degree of variability in both the quality of the final solution and the duration of the runs. The length of runs without agents tended to be longer (by roughly 20%) than the length of runs with active agents. Considering that a large number of runs without agents never reached the pool fitness of runs with active agents, and given that the termination was initiated not by an objective goal such as fitness of the final candidate, or a very specific design goal that designers had to reach, participants seemed to simply terminate the session after having conducted a large number of runs, and felt that they were 'not getting anywhere' in the case of the inactive agents.

As such, the results need to be considered carefully. They are not an absolute reflection of the performance of the agent system, given that the designers were able to terminate after as many runs as they felt were appropriate for reaching a playable map. This implies that some designers may have terminated despite maps not having reached very high quality. An absolute measure does not exist in this context. One suggestion for future studies aimed at establishing whether two maps with different ranks can still both be considered immersive, playable maps, is to select top performers from both active agent and inactive agent interactive runs and find players to test them. Another potential solution may be a study that sets very narrow and specific design goals. Both ideas are subject to interpretation (by either the players or the designers and researchers) but can potentially help to understand the underlying phenomenon somewhat better. Despite these differences, however, it seems that the agents contribute to a much faster increase in fitness, and also to a higher overall ranking within the breeding pool.

There are a number of other interesting points to be raised here. First, there is a logarithmic-looking increase in the fitness of breeding pools. The longer the runs, the smaller the increase of fitness of pools. This was likely due to the small population size in comparison to the breeding pool. The breeding pool comprised 70% candidates that were

bred based on the parents, and 30% bred through random DNA. For roughly 20 generations, the decisions made by the designer seemed to have a significant impact on fitness, and the increase happened rapidly. Thereafter, the increase slowed down and only marginal improvements in rank/ fitness were observed. This can be interpreted as a potential failure of the computational agents. If the user is the main contributor to the increase, given that their decision contributes 70% of the mating pool, the agents may not be as useful as results first suggest. It is important to keep in mind, however, how these user decisions were made. The user observed either their own selection, and a number of randomly selected and mutated candidates in the case of inactive agents, or the user was able to choose from their own selections, two agent-suggested candidates, and a number of randomly selected and mutated candidates (in the case of activated agents). The much quicker increase at the start of the runs with activated agents may indicate that the agents did, in fact, play a role. This can, however, only be verified by looking at what the user decided to select, and perhaps to some extent, by what caught the attention of the user more often.

Another observation is the maximum fitness rank reached by any of the runs. Neither the agent-driven nor the purely user-driven runs reached the full score of 1.0, which may infer a number of things. Perhaps the runs were not sufficiently long. If a genetic algorithm is not able to run through enough iterations to reach the stopping criteria, optimisation will terminate prematurely. If this was the case, the user simply did not make enough selections to reach the theoretical maximum fitness (playability) defined by the designer expert agent. Another possible explanation is that the ranking score is simply not reachable, and that the criteria that the agent is based on are not a complete measure of the quality. Looking at the combined plots of the mean of all runs, it can be seen that the fitness ranking did not achieve saturation, and the maximum continued increasing in an almost linear fashion following the initial rapid increase over 20 generations. It is likely that a fitness value close to the theoretical maximum of 1.0 can be reached, given enough iterations. Previous studies and observations during the data collection confirm that user fatigue arose after some time. Additionally, the subjective nature of what a ‘good, playable map’ is may also have played a role here. If users felt that there was no significant improvement to be gained after many selections, they may have terminated the run. Looking at the score that the average run reached, a fitness of 0.65 may have been considered sufficient even though further improvement may have been possible. If user fatigue contributed to premature termination, this can be mitigated by adding an additional user preference agent (Kruse and Connor 2015). Furthermore, it does not mean that the resulting maps are not playable. The theoretical maximum of 1.0 is based on expert accounts that includes professionals who created some of the most popular and highly acclaimed FPS maps ever.

5. Conclusion

This paper introduced a multi-agent approach to FPS game map design that is based on a decentralised genetic algorithm that works in conjunction with a human game designer and a number of computational agents. The human designer performs the genetic operation of selection supported by the computational agents that attempt to inform the selection process through recommending a range of high quality candidate solutions. Initial results suggest that the use of the computational agents increases the quality of the resulting map, though this is in practice often limited through user fatigue.

Ethics Statement

This research outlined in this has been approved by the ethics committee (AUTEC), reference 17/80.

References

Amato, A. (2017). Procedural Content Generation in the Game Industry. In O. Korn & N. Lee (Eds.), *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation* (pp. 15–25). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-53088-8_2

Cardamone, L., Loiacono, D., & Lanzi, P. L. (2011). 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*. Dublin, Ireland: ACM.

Cardamone, L., Yannakakis, G. N., Togelius, J., & Lanzi, P. L. (2001). Evolving interesting maps for a first person shooter. In *European Conference on the Applications of Evolutionary Computation* (Vol. 2011b, pp. 63–72). Springer.

Cifaldi, F. (2006). Analysts: FPS 'Most Attractive' Genre for Publishers. http://www.gamasutra.com/php-bin/news_index.php?story=8241

Connor, A. M., Greig, T. J., & Kruse, J. (2017). Evaluating the Impact of Procedurally Generated Content on Game Immersion. *The Computer Games Journal*, *6*(4), 209–225.

Cook, M., & Colton, S. (2014). Ludus ex machina: Building a 3D game designer that competes alongside humans. In *Proceedings of the 5th international conference on computational creativity*.

De Carli, D. M., Bevilacqua, F., Tadeu Pozzer, C., & d'Ornellas, M. C. (2011). A Survey of Procedural Content Generation Techniques Suitable to Game Development. In *2011 Brazilian Symposium on Games and Digital Entertainment* (pp. 26–35). Presented at the 2011 Brazilian Symposium on Games and Digital Entertainment. https://doi.org/10.1109/SBGAMES.2011.15

Doran, J., & Parberry, I. (2010). Controlled procedural terrain generation using software agents. *IEEE Transactions on Computational Intelligence and AI in Games*, *2*(2), 111–119.

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, *9*(1), 1–22.

Hullett, K., & Whitehead, J. (2010). Design patterns in FPS levels. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. Monterey, California: ACM.

Järvinen, A. (2009). *Games Without Frontiers: Methods for Game Studies and Design*. Tampere, Finland: VDM, Verlag Dr. Müller.

Kosorukoff, A. (2001). Human based genetic algorithm. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on, 2001* (pp. 3464–3469). IEEE.

Koster, R. (2018, January 17). The cost of games. *Gamasutra*. https://www.gamasutra.com/blogs/RaphKoster/20180117/313211/The_cost_of_games.php. Accessed 23 February 2019

Kruse, J. (2019). *Designer-driven Procedural Game Content Generation using Multi-agent Evolutionary Computation* (Thesis). Auckland University of Technology. Retrieved from https://openrepository.aut.ac.nz/handle/10292/12944

Kruse, J., & Connor, A. M. (2015). Multi-agent evolutionary systems for the generation of complex virtual worlds. *EAI Endorsed Transactions on Creative Technologies*, *2*(5), e5.

Kruse, J., Sosa, R., & Connor, A. M. (2016). Procedural urban environments for FPS games. In *Proceedings of the Australasian computer science week multiconference* (pp. 1–5).

Lanzi, P. L., Lomacono, D., & Stucchi, R. (2014). Evolving maps for match balancing in first person shooters. In *2014 IEEE Conference on Computational Intelligence and Games, 2014* (pp. 1–8). IEEE.

Liapis, A., Yannakakis, G. N., & Togelius, J. (2014). Computational game creativity. In *Proceedings of the fifth international conference on computational creativity*.

Loiacono, D., Cardamone, L., & Lanzi, P. L. (2011). Automatic track generation for high-end racing games using evolutionary computation. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 245–259.

Mark, B., Berechet, T., Mahlmann, T., & Togelius, J. (2015). Procedural Generation of 3D Caves for Games on the GPU. In *10th International Conference on the Foundations of Digital Games*. Pacific Grove, CA.

Mawhorter, P., & Mateas, M. (2010). Procedural level generation using occupancy-regulated extension. In *2010 IEEE Symposium on Computational Intelligence and Games (CIG), 2010* (pp. 351–358). IEEE.

Museth, K. (2014). Hierarchical digital differential analyzer for efficient ray-marching in openvdb. In *ACM SIGGRAPH 2014 Talks* (p. 40). ACM.

Pinelle, D., Wong, N., & Stach, T. (2008). Heuristic evaluation for games: usability principles for video game design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1453–1462). ACM. http://dl.acm.org/citation.cfm?id=1357282

Predescu, A., & Mocanu, M. (2020). A data driven survey of video games. In *2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)* (pp. 1–6). IEEE.

Provano, M., Mannetti, R., & Lomacono, D. (2015). Volcano: An interactive sword generator. In *Games Entertainment Media Conference (GEM), 2015 IEEE, 2015* (pp. 1–8). IEEE.

Short, T., & Adams, T. (2017). *Procedural generation in game design*. CRC Press.

Smith, A. M., Andersen, E., Mateas, M., & Popović, Z. (2012). A case study of expressively constrainable level design automation tools for a puzzle game. In *Proceedings of the International Conference on the Foundations of Digital Games* (pp. 156–163).

Takagi, H. (2001). Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. In *Proceedings of the IEEE* (Vol. 89, pp. 1275–1296).

Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelbäck, J., Yannakakis, G. N., & Grappiolo, C. (2013). Controllable procedural map generation via multiobjective evolution. *Genetic Programming and Evolvable Machines*, *14*(2), 245–277.

Togelius, J., Preuss, M., & Yannakakis, G. N. (2010). Towards multiobjective procedural map generation. In *Proceedings of the 2010 workshop on procedural content generation in games* (pp. 1–8).

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2010). Search-based procedural content generation. In *European Conference on the Applications of Evolutionary Computation* (pp. 141–150). Springer.

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 172–186.

Walsh, P., & Gade, P. (2010). Terrain generation using an interactive genetic algorithm. In *Evolutionary Computation (CEC), 2010 IEEE Congress on, 2010* (pp. 1–7). IEEE.

Wright, T., Boria, E., & Breidenbach, P. (2002). Creative player actions in FPS online video games: Playing Counter-Strike. *Game studies*, *2*(2), 103–123.

Yannakakis, G. N., & Togelius, J. (2011). Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, *2*(3), 147–161.

Yannakakis, G. N., & Togelius, J. (2018). Generating Content. In G. N. Yannakakis & J. Togelius (Eds.), *Artificial Intelligence and Games* (pp. 151–202). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-63519-4_4

Yoon, D., & Kim, K.-J. (2012). 3D game model and texture generation using interactive genetic algorithm. In *Proceedings of the Workshop at SIGGRAPH Asia* (p. 53–58).

Zhu, M., Zhao, F., Fang, X., & Moser, C. (2017). Developing Playability Heuristics Based on Nouns and Adjectives from Online Game Reviews. *International Journal of Human–Computer Interaction*, *33*(3), 241–253. https://doi.org/10.1080/10447318.2016.1240283