EAI Endorsed Transactions
on Creative Technologies

Research Article EAI.EU

Evolutionary generation of game levels

A.M. Connor*, T.J. Greig and J. Kruse

Auckland University of Technology, Auckland, New Zealand.

Abstract

This paper outlines an approach for evolutionary procedural generation of video game content. The study deals with the automatic generation of game level designs using genetic algorithms and the development of a fitness function that describes the playability of the game level. The research explores whether genetic algorithms have the ability to produce outcomes that demonstrate characteristics that arise through human creativity, and whether these automated approaches offer any benefits in terms of time and effort involved in the design process. The approach is compared to a random method and the results show that the genetic algorithm is more consistent in finding levels; however analysis of the game levels indicates that the fitness function is not fully capturing level playability. The ability to produce playable levels decreases as the play area increases, however there is potential to produce larger maps that are both playable and arguably creative through a recombination method.

Keywords: Procedural Content Generation, Creative Computing, Novelty Generation, Video Game Design, Genetic Algorithms, Computational Creativity.

Received on 07 November 2017, accepted on 30 March 2018, published on 10 April 2018

doi: 10.4108/eai.10-4-2018.155857

*Corresponding author. Email:andrew.connor@aut.ac.nz

1. Introduction

Game content is an important factor in keeping players engaged in gaming worlds, yet games are becoming increasingly complex which has a corresponding impact on game development [1]. As well as dealing with this complexity in terms of the underlying code, there is an increasing demand for new game content. In general, game content and game asset creation are some of the most significant costs of a game development [2]. Manual content production is therefore expensive and potentially not scalable [3]. In contrast to manual content production, Procedural Content Generation is the application of computers to generate game content, specifically in terms of algorithmic generation of game content with limited or no human contribution [4]. However, PCG is considered difficult as it not only incurs considerable computational overhead, but also requires the ability to compute the technical and cultural values of the generated instances [5]. There are also open questions as to whether PCG can produce novelty in terms of content that could be comparable to the outcomes of human creativity.

The research in this paper outlines a comparative study to determine the implications of developing game content using evolutionary computation as an approach. In particular, the study compares the use of a genetic algorithm based approach to the design of game levels with more randomised content generation. This research is an initial study that attempts to compare the outcomes of the two approaches to determine whether the game levels produced by the genetic algorithm have any novel characteristics that are not found in the randomly generated levels.

The remainder of this paper is structured as follows. Section 2 outlines the background to this research and discusses related work. Section 3 describes the both genetic algorithm used to evolve the game levels in this study. Section 4 outlines the evaluation approach for different sized game levels that allows the scalability of the approaches to be inferred. Section 5 discusses the results and presents directions for future work, whilst Section 6 concludes the paper.

2. Background and Related Work

The creation of game content, such as models, levels, textures, and other items within the game world, is time-consuming and expensive [6]. In addition, it has been noted that the manual creation of game content has a range of other drawbacks that include a lack of flexibility and suggestions that manual approaches are inherently unscalable [7]. The automatic creation of game content is not new, with examples dating back to the 1980s [4], however there are continuing challenges in finding ways to reduce unwanted artefacts that can be encountered using simple random generation of content.

Procedural Content Generation (PCG) is therefore an area of study that looks at the automated generating of useful content for games. It has been argued that games are the "killer app" for the study of computational creativity but games can also be considered in much the same light from a creative computing context.

2.1. Creative Computing and Computational Creativity

The terms "Creative Computing" and "Computational Creativity" are a simple juxtaposition of two words, but a simple swap that produces a significant change in meaning. HUGILL and YANG [8] suggest that "the former is about doing computations in a creative way, while the latter is about achieving creativity through computation". YANG and ZHANG [9] continue this argument by stating that "the difference can be that Creative Computing requires computing to be creative. However, computational creativity is to generate machine creativity through simulating human creativity, which does not necessarily require computing itself to be creative".

Computational Creativity is the therefore the art, science, philosophy and engineering of computational systems that exhibit behaviours that unbiased observers would deem to be creative. It has been observed that "from a CC perspective, procedural content generation (PCG) in games has been viewed — like mathematics and engineering — as a potentially creative activity but only if done exceptionally well." [10]. These authors also suggest that "computational game creativity as the study of computational creativity within and for computer games. Games can be (1) improved as products via computational creations (for) and/or (2) used as the ultimate canvas for the study of computational creativity as a process (within)" [10]. However, whilst coherent in its own definition, this perspective does not address the need for creative approaches to developing games or the development of tools to support and understand human creativity.

Autonomous creative systems have a long history in the game industry with many examples readily cited in the literature [5] along with many commercial games having been created using such approaches, with very successful titles such as *Diablo III* and *Skyrim* being just two examples. With a growing demand for engaging but unpredictable game experiences, such autonomous creative systems are increasingly becoming a necessity for the games industry. This necessity therefore requires that the approaches to game design itself are explored creatively as well as being a process that produces creative outcomes. LIU [11] identifies some of the challenges around creative exploration of process and the need for appropriate tools to support those process in a definition of creative computing: "Creative computing aims to better understand human creativity and to formulate an algorithmic perspective on creative behaviour in human; and to design programs that can enhance human creativity without necessarily being creative themselves".

With this in mind, the automated generation of game content can therefore be viewed simultaneously using both computational creativity and creative computing as lenses to analyse and reflect on the outcomes of the work. In the context of exploring new tools to support the creative process, there has been a growing emphasis on the use of computational intelligence techniques such as evolutionary computation. Whilst such techniques are arguably not creative, they however offer the potential to enhance human creativity as well as the ability to produce a surprising result. Such surprise has been identified as an essential aspect of creativity [12] and therefore the use of such approaches may be helpful in producing an output with creative elements.

The following sections provide an overview of both Procedural Content Generation (PCG) in general, as well as specific examples of evolutionary approaches to PCG.

2.2. Procedural Content Generation

This section introduces existing work in the area of Procedural Content Generation. It is not an exhaustive review, as such surveys are already published in the literature [5]. Instead, this section introduces PCG for later considering evolutionary approaches to PCG. PCG can be used for a variety of reasons, including providing variety, reducing development time and development costs, saving space in transmission or on disk, augmenting human creativity and enabling adaptivity in games [4].

Togelius, Kastbjerg and Schedl [13] attempt to classify PCG through a process of defining what it is not. Implicit in this work is the distinction between *online* PCG and *offline* PCG, the former being where content is generated at runtime during gameplay and the latter being where content is generated during the design of the game or prior to gameplay [14]. Online PCG is both challenging and fascinating with the potential to impact factors such as the replayability of games as well as promoting the emergence of new game dynamics [15], however work outlined in this paper specifically address offline PCG.

Offline PCG facilitate the game development process and typically involves systems that assist game developers in their design process [16, 17] or through the creation of game assets [18] or levels [19-21]. Traditional approaches to PCG utilise a number of techniques or theoretical frameworks, such as L-systems [22] or other space based approaches

[23], statecharts [24] and petri nets [25], along with an emergence of declarative approaches [26] or those using techniques such as answer set programming [27].

Despite these different approaches to PCG, it has been observed that "generated content generally looks generic" [4], and there is an obvious challenge in using PCG to generate content that would exhibit the same novelty characteristics as arise from human creativity. One of the dominant trends in recent years is the growing interest in search based or evolutionary based approaches to PCG. Evolutionary algorithms have been used as novelty generators in other domains [28-30], indeed with some authors going as far as to suggest that evolutionary algorithms are better considered as continuous novelty generators as opposed to optimisers [31] and that the dominant theme of evolutionary computation is novelty generation [32].

2.3. Evolutionary PCG

Evolutionary Procedural Content Generation (EvoPCG) is a specific case of search based PCG where the search algorithm utilises an evolutionary base. In general, search based PCG [33] provides the opportunity to explore massive search spaces and find unique solutions that may not be generatable using more traditional approaches. The identification of "unexpected" solutions using search algorithms has been seen in a variety of domains, such as software engineering [34], engineering design [35, 36] and robot morphology [37] to name but a few explicitly. Many examples of such novelty generation or unexpected solutions can be found in the literature [38-41]. The application of evolutionary algorithms offer the same potential in terms of the possibility to identify surprising and novel solutions in the creation of game content.

Whilst the term search based PCG was only recently utilised [42], the use of evolutionary algorithms has been relatively extensive prior to this. As with PCG in general, surveys of existing search based approaches to PCG exist in the literature [33, 43] and this review does not attempt to replicate this work and instead focuses specifically on the use of genetic algorithms in the generation of game maps.

One of the challenges of the use of genetic algorithms in game level design is the complexity of formulating a fitness function that captures the intent of a game designer. It is perhaps this challenge that has limited the uptake of automated evolutionary approaches to procedural content generation. Whilst not specific to game level design, the challenge of capturing vague performance measures has been identified in other applications of evolutionary computation [44]. It is the formulation of the fitness function that ultimately determines the extent to which a genetic algorithm can explore beyond the expected. To circumnavigate this challenge, many applications focus on the use of interactive evolutionary computation where the designer evaluates the levels in place of a fitness function. Examples of such approaches have been applied to city layouts [45], racing car track levels [46], terrain generation [47] and game levels [48] to name but a few. Whilst these approaches are interesting in terms of computationally augmenting human creativity, they still suffer from being a manually intensive process. A more useful approach would then be the use of a fully automated approach.

Various studies have addressed the evolutionary generation of game levels, for example Ashlock, Lee and McGuinness [49] apply both cellular automata and genetic algorithms to the process of creating game level maps. Similarly, Hartsook, Zook, Das and Riedl [50] use a genetic algorithm to create 2D role playing game worlds. Whilst some studies can be found in the literature, the use of genetic algorithms in the creation of game levels is still an underexplored area. Perhaps the most relevant study to date is the work of Sorenson and Pasquier [51] who create a generic fitness function for the "fun" developed in playing a given level of a game to guide the generation process. In this particular work, the mechanics of the game creation process are modelled as constraints rather than embedded in the fitness function. For example, a game level that is not traversable is considered as violating a constraint which arguably could mean that valuable game level features are quickly lost from the genetic mating pool.

Common to all approaches identified in the literature are challenges related to the game level representation and the formulation of the fitness function. Opportunities exist to further explore the potential for applying genetic algorithms or other metaheuristic search algorithms to the game level design process to better explore how to represent levels and formulate fitness functions in order to address the challenge of creating original content [52].

3. Evolutionary Game Level Creation

This research sets out to consider whether the use of evolutionary algorithms provides any quantifiable benefit when used in the game design process to procedurally generate game levels. To that end, a Genetic Algorithm has been utilised to automatically design game levels for a simple top down shooter game. The outcome of evolving such levels is compared to randomly generated levels in the first instance. Initially, the Genetic Algorithm approach also involve the random generation of a population of levels and for consistency the same approach to generate levels is utilised as with the random generation baseline.

3.1. Game Level Representation

The game used to evaluate the approach is a top down shooter game where the player controls a character who explores the level, finds keys to open doors and confronts enemies that can be attacked using a variety of weapons. A generic level therefore consists of an m x n grid of cells and for the purpose of the level design it is assumed that the start of each level is always the bottom left cell and the exit is the top right cell. Whilst somewhat artificial, the prescription of entrance and exit cells has been used in similar studies [49].

Within the level, cells can either be walls or vacant spaces that combine to give rooms and corridors. A typical level is shown in Figure 1 where the dark spaces are the walls and the white the contiguous space of passageways and rooms.

Figure 1. Typical game level

Initial experimentation on level generation showed that generating initial levels using a cell based representation, where each cell was initialised independently as a wall or void resulted in noisy and incoherent levels. Such an implementation based on individual cells is the obvious approach, however the outcomes drove the development of the final representation into more creative directions. The adoption of an evolutionary algorithm was itself a trigger to human creativity to identify potential representations that might not otherwise have been considered. The final generative strategy used is dubbed "falling rectangles". It is analogous to simply throwing some rectangular shapes into a space and using where they fall as the level map.

To implement this in practice, a number of assumptions were made and control parameters assigned. The main assumption is that a "good" level will have a coverage of about 70% of the available space as playable space that is reachable from the start point. This goal is defined as the desired space ratio, *R*. Whilst this is given an arbitrary value, this is not a real limitation of the current work as the intention is to compare the ability of different methods to reach the goal and not to validate the goal in its own right. This approach for generating levels has been fully described in other work [53] but the essence of the approach is the determination of the number of randomly placed rectangles that will likely result in the desired space ratio, *R*. The following equation is used to calculate the number of rectangles used (n_r):

$$n_r = \left(\frac{n_c \times R}{a_p}\right) \times (1 + O) \quad (1)$$

In this equation, n_c is the total number of cells in the level determined by the width and height and *O* is the overlap factor, which is a simple adjustment that takes into account potentially overlapping tiles. Finally, a_p is the projected average area of rectangles measured in cells that is determined from the upper and lower bounds on the sizes of the generated rectangles.

Upon generation of a level, the state of each individual cell is determined to be either a wall or a void, and all future operations are conducted by manipulating cells. For the Genetic Algorithm the initial population of individuals was created by repeating this approach to generating an individual level until the initial population was complete.

3.2. Genetic Algorithm Implementation

Genetic Algorithms [54] are a search and optimisation approach that are based on the principles of natural selection and population genetics. Genetic Algorithms (GAs) have been widely used in science, engineering and other domains as an adaptive algorithm for solving practical problems and to computationally explore solution spaces. They have been successfully applied to problems as diverse as equipment selection [55], machine design [56], service composition [57] and layout design [58] to name but a few. In general, GAs are considered a robust global optimisation algorithm.

The basic operation of GAs is well documented in the existing literature [59, 60], consistently essentially of a population of individuals with in which breeding takes place to produce a new generation. During the breeding process, individuals are selected to mate based on their fitness (or quality score) with the more fit individuals likely to be selected. This results in the transfer of strong characteristics from generation to generation, which is the essence of survival of the fittest. Two individuals create two children in the next generation through the application of the crossover (or recombination) and mutation operators.

The final elements for a GA to be implemented are the creation of a fitness function that differentiates between individuals and also a specific encoding of an individual in such a way that it allows the GA to evolve better individuals over time. The specific characteristics of the GA implemented in this research are discussed in the following sections.

Chromosome Encoding

Traditionally, GAs have used a binary encoding where either real or symbolic parameters are represented by mapping to one or more bits in a binary string. In this research, the two dimensional game level map is transferred into a one dimensional binary array, where each bit

corresponds to one cell in the level map with a value of 1 indicating that the cell is currently a wall, and a 0 indicating that the cell is currently a void. A simple 4x4 grid is shown in Figure 2 along with the chromosome encoding.

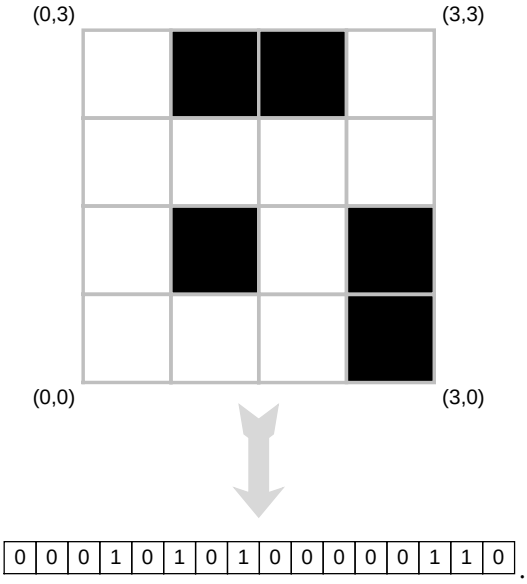


Figure 2. Chromosome encoding for a 4x4 grid

Selection

The GA implemented in this research uses a roulette wheel selection method, which is one of the most common selection approaches. In roulette wheel selection, the population of individuals is ranked according to the fitness value. A cumulative probability of selection is awarded based on the contribution of the fitness of each individual to the total fitness of the generation. Two individuals are selected for breeding based on a randomly generated number that is compared to the probability of selection.

Crossover

The crossover implementation used in this research is a simple single point crossover. A crossover point is chosen at random using a probability test for each cell against a fixed value. When a crossover point is chosen, the two parents are recombined to produce two new children as shown in Figure 3, where the crossover cell is indicated by an X.

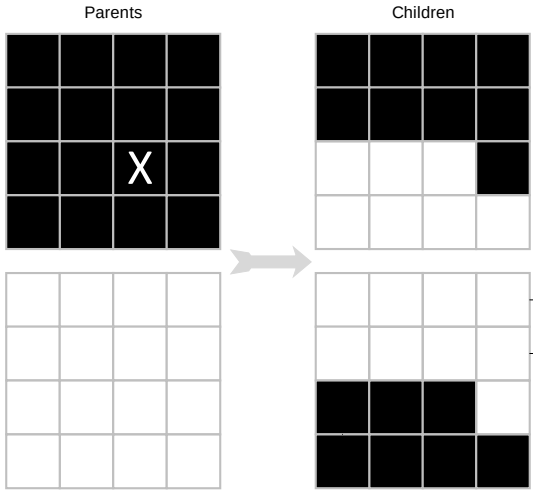


Figure 3. Single point crossover

Using these crossover scheme, existing wall and void structures are recombined without any knowledge of the spaces that may exist in the game level map.

Mutation

The normal approach to mutation in a binary string GA is to randomly flip bits with a very low probability, which in the case of the game level maps would result a cell changing from either a wall to a void, or vice versa. Initial trials indicated that this strategy alone was disruptive and impacted the ability to produce coherent and playable levels. As with the initial level generation, this outcome triggered a new thought process as to how to represent the morphing of spaces with in a given level. As a result of this, an additional strategy was implemented based on contiguous elements of the level. The resulting mutation function selects a space at random and mutates the edge of that space. This allows spaces in the level to expand or contract in a mutation operation. This may allow rooms to connect, become two separate rooms, or carve a hallway over time.

Initially, each cell in the level was tested against a very low probability of mutation to see if standard mutation occurred. If no cell mutation occurred, the contiguous mutation operator was applied. This selects a cell at random, and finds the void containing that cell. If the cell is not associated with a void, it selects another cell. There are two different operations the mutation function can perform on a space which either expand or contract the void, and for each mutation the choice of operation is chosen at random. To contract a void, each of the cells on the boundary of void is tested against a mutation probability and if mutation occurs the type of this cell flipped so that a void cell becomes a wall. This is shown in Figure 4 where the initial mutation site is shown with an X, and the set of cells available to mutate shown dotted. In this case, cell Y is undergoing mutation.

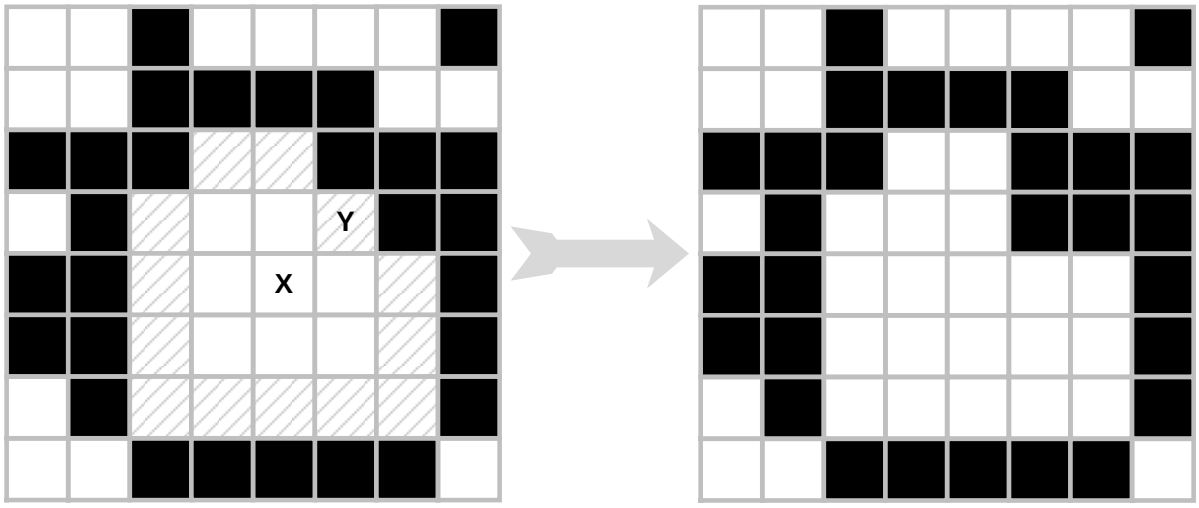


Figure 4. Mutation strategy (Contraction) with void indicated by Cell X and the mutation site indicated by Cell Y

The expansion operation works in a similar fashion, expect the cells available to flip are the external boundary of the void. In the case shown in Figure 5, these cells are shown as dotted and the expansion strategy has resulted in two different contiguous voids joining together to create a much larger navigable space in the game level.

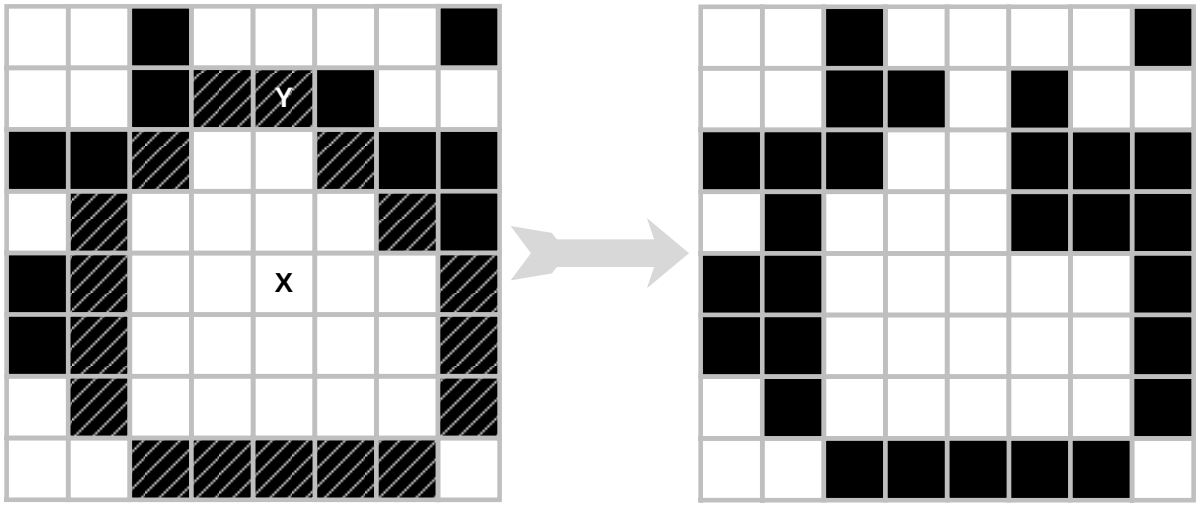


Figure 5. Mutation strategy (Expansion) with void indicated by Cell X and the mutation site indicated by Cell Y

The mutation operator is not limited to a mutation of a single cell, with each of the cells in the mutation set being tested against the mutation probability. The impact of this particular mutation strategy is that existing wall or void structures can grow or shrink relatively quickly, and are less likely to be disrupted by random cell mutations. This has the potential to result in more coherent and playable levels.

Fitness Function

The fitness function operates on the assumption that there is a target ratio between space that can be traversed and the total size of the level. Because the level map includes an explicit entry and exit cell, there is also a requirement that there must be at least one continuous path joining these two cells for the level to be playable. The space ratio is the ratio between the number of cells that are reachable in the largest contiguous space in the level and the total amount of the cells the level contains. The space ratio of a level can be compared against the desired space ratio to give an error value that indicates how far away any given level is from the target. The actual space ratio (r_a) of the level is given by:

$$r_a = \frac{n_{rc}}{n_c} \tag{2}$$

where n_c is the total number of cells in the level and n_{rc} is the number of reachable cells in the largest contiguous space. The space ratio error (R_e) is then given by:

$$R_e = |R - r_a| \quad (3)$$

Two other contributing factors are included in the fitness function calculation that are used to determine to what extent a continuous path exists between the entry and exit cells of the level. These factors are the proximity of the main space to the entry and exit cells are calculated as a ratio of how close a cell in the largest contiguous space is located to the entry (or exit) in relation to the distance between the entry and exit cells. This is illustrated in Figure 6.

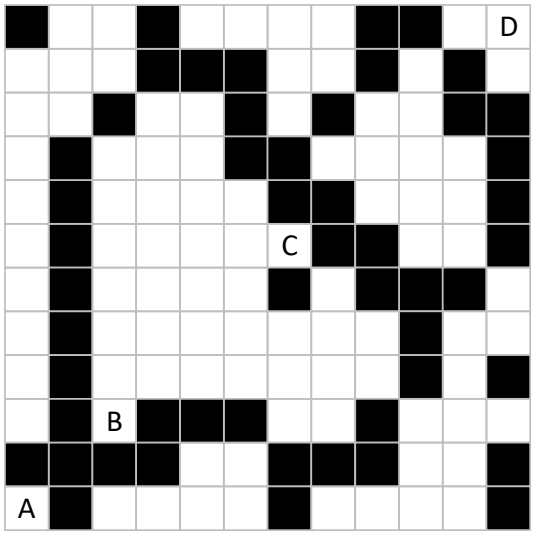


Figure 6. Proximity of largest space to entry and exit

The proximity of exit and entry are then determined by the following distance ratios.

$$P_{ex} = \frac{CD}{AD} \quad (4)$$

$$P_{en} = \frac{AB}{AD} \quad (5)$$

When the largest contiguous space includes either the exit or entrance cells then the corresponding terms drops to zero. These two terms are combined with the space ratio to create the following fitness function.

$$Fitness = (1 + P_{ex})^2 \times (1 + P_{en})^2 \times (1 + R_e)^2 \quad (7)$$

For a level where the space utilisation of the largest contiguous space is 70%, and the space contains both the entry and exits cells, this results in a theoretical minimum value of the fitness function of 1. Such clarity over a known minimum is useful in the determination of how close any given level is to a "perfect" level. The effectiveness of the formulation of the fitness function will be discussed in section 5 following the presentation of the results.

4. Game Level Evaluations

The evolutionary approach outlined in the previous section has been evaluated for different sizes of levels. For each size, the GA and random generation approach were repeated ten times to take into account the stochastic nature of the algorithm. For each approach, the settings associated with the generation of each initial level were constant with an overlap percentage of 50%.

The GA used a population size of 100, and was run for 100 generations. The GA was elitist, so that the best level in each generation was transferred to the next generation. The final solution for each run was the best ever solution found in the total number of generations. For the random generation baseline 10,000 candidate solutions were generated for each run and the fitness of each computed using the same fitness function before selecting the best solution. The results for each size of level are presented in the following sections, where each level is also evaluated using the lens of computational creativity to determine whether the levels could be considered a creative output.

4.1. Small levels (25x25)

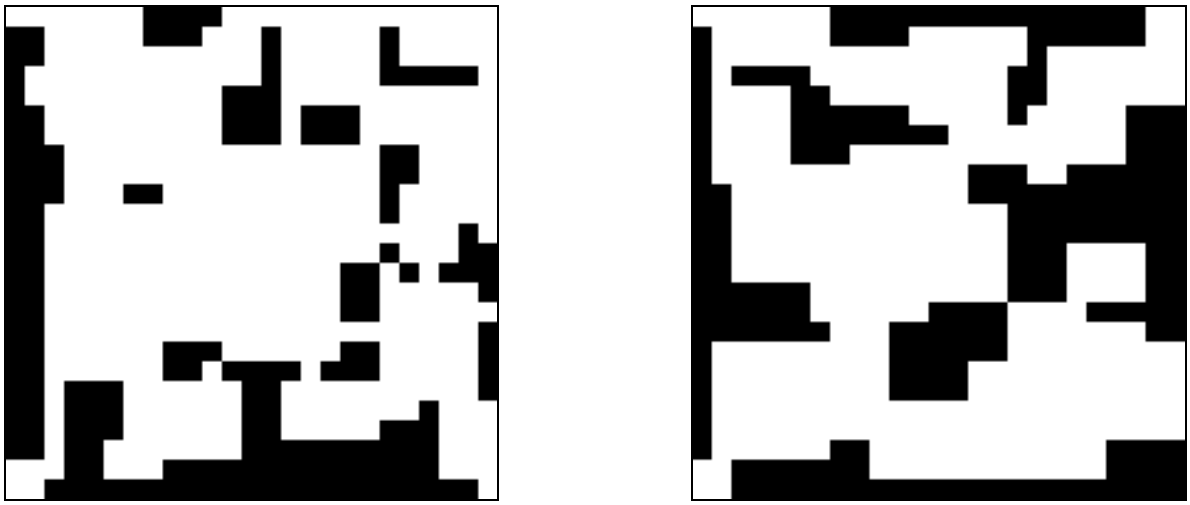
For the purpose of this research, a small level is defined as a grid with 25 cells on each dimension, resulting in a total potential of 625 cells included in the level map. Table 1 outlines the quantitative data related to the generation of solutions. Across the ten runs performed, the GA is more consistent in finding high quality solutions with every final level having the same resulting fitness. Whilst the theoretical minimum of the fitness function is 1, in practice this will not be achieved because the discrete nature of the level is such that the number of cells included in a level will never be exactly 70% of the 625 available cells. It can be assumed that the GA consistently finds levels that are as close to optimal as possible.

Table 1. Quantitative performance comparison (25x25)

	GA	Random
No. Evaluations	10,000	10,000
Mean Fitness	1.001601	1.1941024
Median Fitness	1.001601	1.197717
Best Fitness	1.001601	1.144044
Worst Fitness	1.001601	1.252609
Fitness (S.D.)	0	0.0338643

The statistics presented are representative of the ten final solutions produced from each run of the solution method. The generation of random levels exhibits both greater

variability in final outcome as well higher fitness levels, arguably showing that random level generation is not as effective in finding good levels as the GA. In this context, the term "good" is used in relation to the formulation of fitness function rather than the actual playability of the level. Actual playability can be considered by comparing the best levels generated using the two different approaches. The best solution from the 10 candidates generated using random generation was selected purely on the fitness function value, as a distinct best solution was identifiable. However, multiple solutions generated using the GA had the same fitness score and hence a best solution was selected by visually inspecting the candidates. Such a subjective evaluation is of course questionable, and the fact that it is required suggests that the fitness function is not sufficiently differentiating good characteristics in different levels. The chosen solution and the best randomly generations solution can be compared in Figure 7.



(a) GA (b) Random

Figure 7. Best candidate solutions (25x25)

Both of these levels are playable in the sense that there is a continuous path between the entry and exit cells, however the randomly generated level contains less open space and more of a sense of distinct "rooms". Given the observation in the literature that generated levels "lack meaningful macro-structure and a sense of progression and purpose" [52] it is worth exploring these differences in more detail, albeit qualitatively and subjectively.

Whilst the randomly generated levels have distinct rooms, each is a relatively large open space and as such are not likely to elicit any sense of surprise during gameplay. An attempt could be made to produce a better gameplay experience through inserting game elements such as doors and enemies, but there are few areas where immediate line of sight in the large spaces would not lead to an obvious understanding of the situation.

In contrast, whilst the procedurally generated level has one much larger space, it also contains some longer corridors that would produce a sense of progression during gameplay and many smaller spaces and hidden areas that would allow a more surprising placement of game elements Arguably, both levels could be used effectively within a top down shooter game, despite the difference in fitness score though both may result in a different player experience.

However, each could be tailored to a specific game scenario in the downstream game design processes that include the placement of game assets.

Given the small size of this level maps, it is difficult to classify whether the procedurally generated level would be considered as a creative outcome, and therefore identify whether the approach is meeting the stated goals of computational creativity. The only elements within the level that add a sense of them potentially being the outcome of a creative process are the various "pillars" in the interior of the rooms which provide potential cover from enemy fire during gameplay and would be features typically used by a human game designer.

Such features are missing from the randomly generated level, however the procedurally generated level is also missing other aspects often included by human game designers such as symmetrical design, maze-like topography and identifiable shapes and features. Whilst the scale of the solution limits the potential for this to be seen, it is unlikely that a human game designer would consider this type of level map as something creative

4.2. Medium levels (50x50)

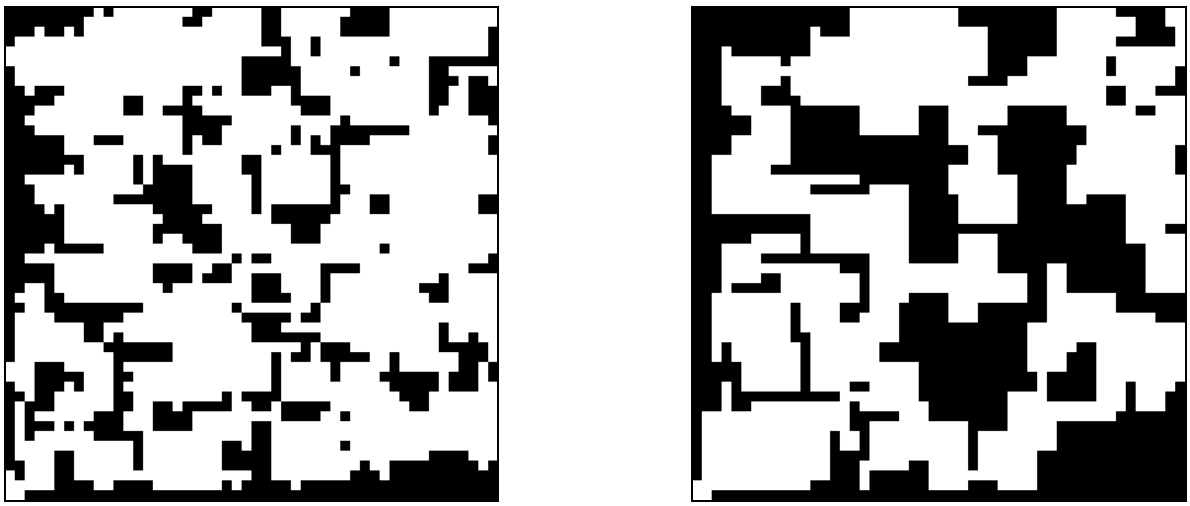
For the purpose of this research, a medium level is defined as a grid with 50 cells on each dimension, resulting in a total potential of 2500 cells included in the level map. Table 2 outlines the quantitative data related to the solution generation of the two approaches.

Table 2. Quantitative performance comparison (50x50)

	GA	Random
No. Evaluations	10,000	10,000
Mean Fitness	1.1057698	1.3673496
Median Fitness	1.105022	1.360535
Best Fitness	1.002401	1.305078
Worst Fitness	1.205604	1.43904
Fitness (S.D.)	0.0575286	0.0419825

As with the smaller level, the genetic algorithm is again more consistent with finding highly fit solutions. However, in this case the GA is not consistently finding the practically optimal solution and fitness function values are generally higher than with the smaller levels. The randomly generated levels generally exhibit a more marked difference in terms of worse fitness than was seen with the smaller 25x25 levels and are again generally less fit than those levels discovered by the GA. There is also a marginal increase in variability when compared to the random generation of small levels. Interestingly, the GA is more variable in its outcomes when the standard deviation of fitness values is used as an indication of spread. However, the worst level found by the GA still has a better fitness than the best randomly generated level.

The best solution from the 10 candidates generated using each method could be selected purely on the fitness function value as a distinct best solution was identifiable. This differs from the results for the smaller levels where manual inspection was required to determine a candidate solution from those generated by the GA. The two solutions can be compared in Figure 8.



(a) GA (b) Random

Figure 8. Best candidate solutions (50x50)

The distinction between these two levels is less marked than with the smaller 25x25 levels. At this size, both levels have the same potential to provide an engaging game level as both levels exhibit a character of distinct rooms. The marked difference in this case is that the evolved level has less “clean lines” in terms of the boundaries of the spaces in the level. Analysis of the evolution trajectory indicated that the most significant changes occurred through crossover in the early generations. However, once a good solution was found, mutation become the dominant operator. Rather than joining spaces as expected, mutation produces non-smooth walls. A simple manual cleaning of the space would result in a highly playable level.

With the increase in scale, the potential for overlap between the “falling rectangles” in the randomly generated level has resulted in a number of “pillars” as seen in the smaller procedurally generated level (25x25). However, the larger procedurally generated level also contains such features as well as maintaining the narrow corridors. Whilst the mutation operator is eroding the integrity of some spaces through the creation of many single cell alcoves, it is also the mutation operator that is the likely source of narrow corridors. Both levels lack features that might be included by a human designer and from a computational creativity perspective, neither level includes would surprise or delight a game designer inspecting the level maps nor be considered “creative”.

4.3. Large levels (50x50)

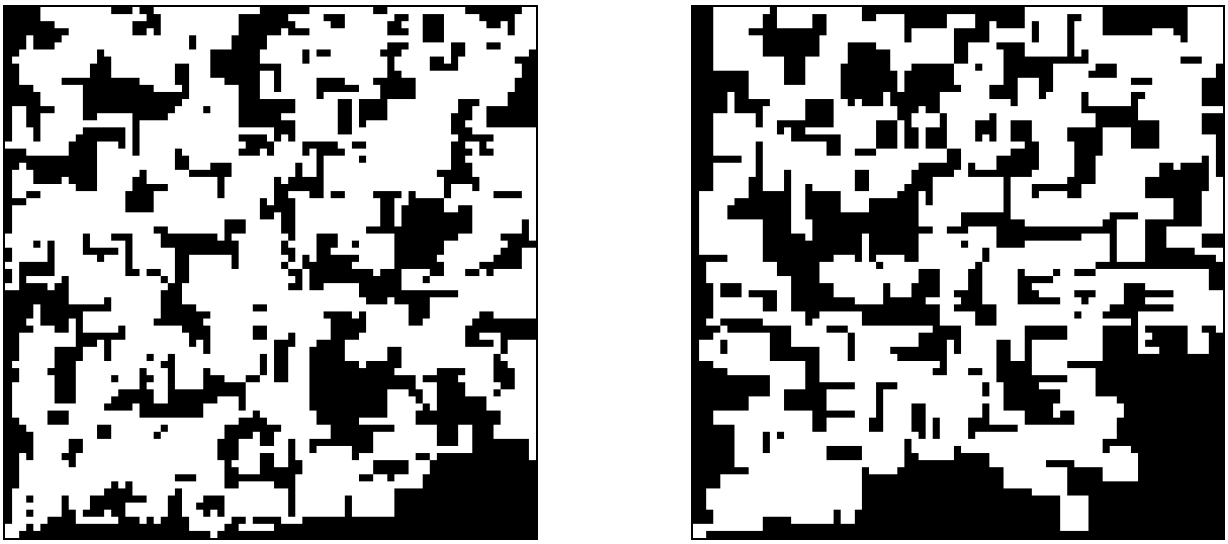
For the purpose of this research, a large level is defined as a grid with dimensions of 75 cells in each dimension, resulting in a total of 5625 cells included in the level map. Table 3 outlines the quantitative data related to the solution generation of the two approaches.

Table 3. Quantitative performance comparison (75x75)

	GA	Random
No. Evaluations	10,000	10,000
Mean Fitness	1.2234681	1.4419483
Median Fitness	1.224525	1.456114
Best Fitness	1.15319	1.334692
Worst Fitness	1.296359	1.553735
Fitness (S.D.)	0.0447513	0.063888112

The difference in performance seen when comparing the 50x50 data with the 25x25 data is repeated, but more apparent. There is more variability in the fitness scores for the randomly generated levels and again the actual fitness scores are considerably higher than both that of the randomly generated medium sized levels and the evolved larger levels, suggesting that the solutions are further away from the optimal case.

In this case, there was again a distinct best solution identifiable from the ten solutions produced by the genetic algorithm. This is shown in comparison with the best randomly generated level in Figure 9. Again, the differences between these levels is predominately related to the broken nature of the room walls that has been caused by the mutation operator. As with the medium sized levels, both solutions offer very playable levels albeit with the suggested manual cleaning of the evolved level. Saying this, the scale of these levels is such that many of the spaces would be considered fairly open during gameplay, particularly in the procedurally generated level. Both levels also contain multiple pathways between any two zones in the map which would be relatively uncommon in many games that utilise levels of this type.



(a) GA (b) Random

Figure 9. Best candidate solutions (75x75)

Whilst some features such as narrow corridors and pillars exist, arguably these larger levels are less playable simply as a result of the navigational uncertainty that would arise from these multiple pathways. Neither would be considered as particularly desirable (and hence creative) levels from the perspective of a human game designer.

5. Discussion

The previous section has presented results that facilitate the comparison of evolutionary procedural content generation with the random generation of game levels. An increase in size of the game level results in an increasingly complex generation task, however in all cases both the genetic algorithm and the random generation approaches find feasible levels. However, as the size increases there is a perceived lack of usefulness of the generated levels and arguably a reduction in the level of creativity identifiable in the levels.

The data presented in Tables 1-3 support the argument that the complexity of the generation task increases as the size of the level is increased. The increase in complexity can also be evidenced by considering how the genetic algorithm converges to a solution. Figure 10 shows the mean fitness function of the best solution in each generation across the ten runs of the genetic algorithm for each size of level.

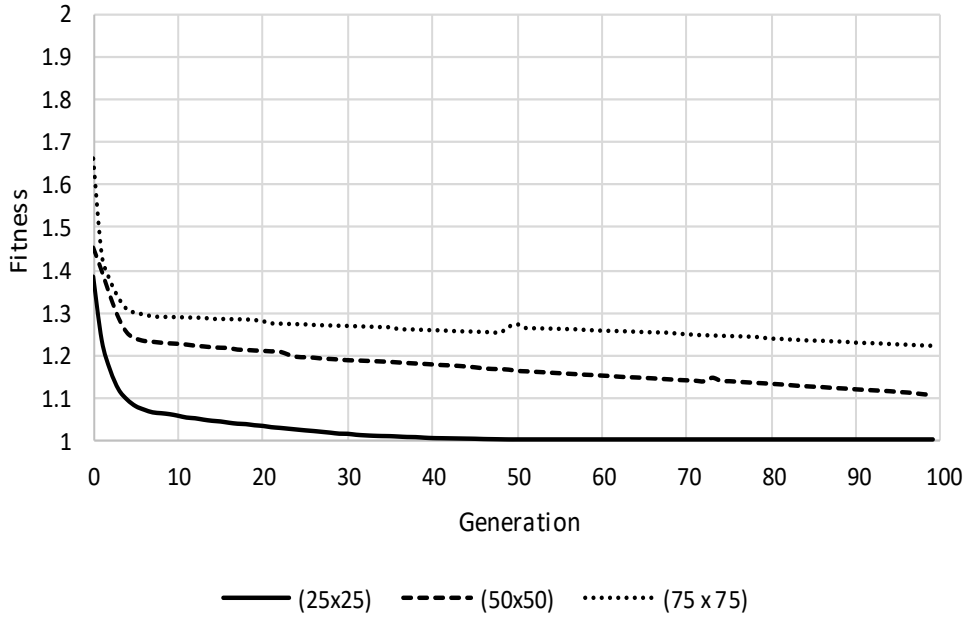


Figure 10. Convergence of the genetic algorithm

As the size of the level increases, the rate of convergence decreases and the quality of the final solution is reduced according to the fitness function. However, the current formulation of the fitness function needs to be questioned. Whilst the GA finds supposedly better solutions over time in comparison to the random generation, visual inspection of the levels in Figures 6-8 suggests that the GA potentially produces levels that are neither more or less playable than the randomly generated levels, but are arguable slightly more fragmented both in terms of the boundary walls and the overall space.

It is possible that this outcome is the result of the fitness function not really differentiating between levels with different desirability. For example, the two levels shown in Figure 11 were produced by the genetic algorithm and have the same fitness function score as the level shown in Figure 6(a). Put simply, for the smaller levels it is relatively easy for the GA to generate a solution that meets the optimality criteria defined by the fitness function, and many such solutions exist with different characteristics that are not captured or differentiated by the formulation of the fitness function.

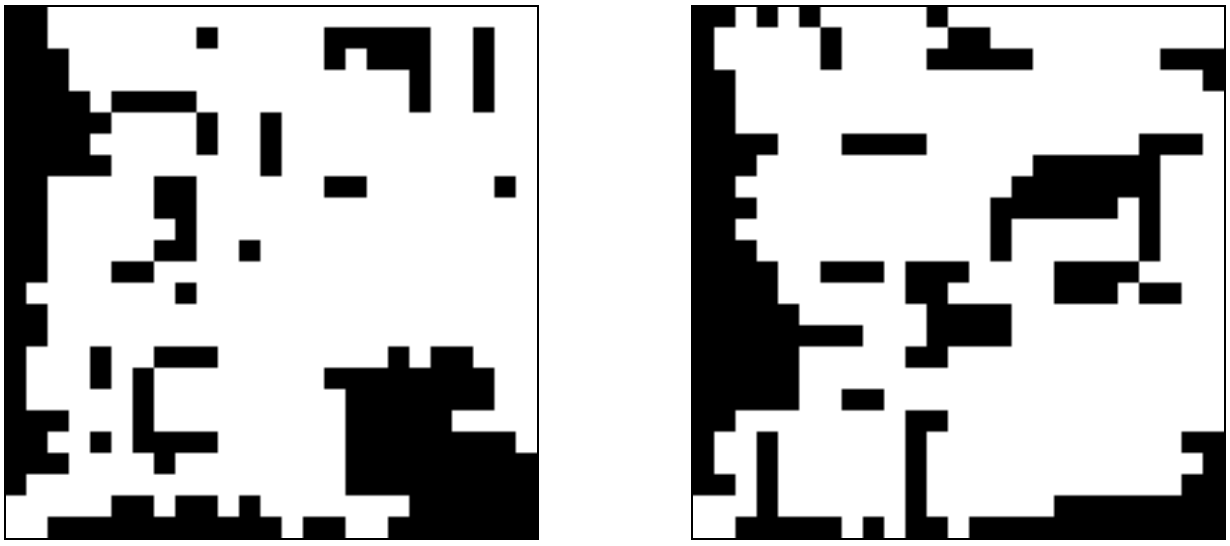
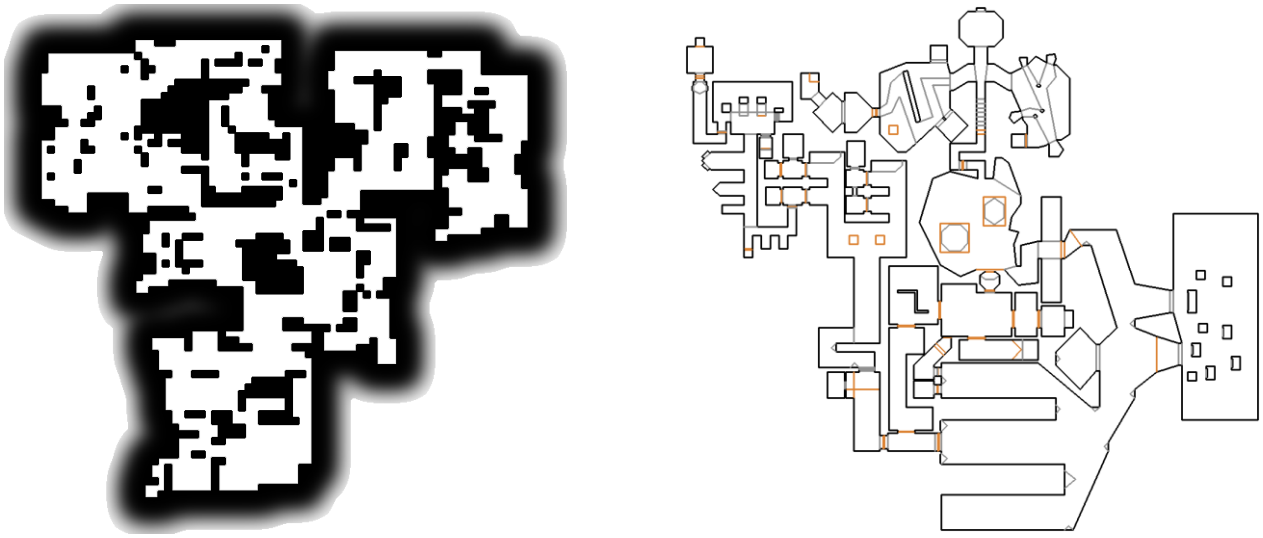


Figure 11. Game levels with the same fitness function

These two game levels have very different characteristics than the level in Figure 6(a), and consistently the randomly generated levels exhibit much more of a "room and corridor" feel. Inspection of game levels from successful games (e.g. Doom, Wolfenstein) display the characteristic of distinct rooms connected by corridors. However, whilst these levels are not potentially useful or creative as a standalone entity, there is the potential to utilise them as building blocks in a process to assemble a larger, more useful and creative outcome. The outcome of such a process is shown in Figure 12(a) in comparison with a level from Doom in Figure 12(b) sourced from the Doom Wiki [61].

To produce this game level map, five of the final solutions from the genetic algorithm were selected at random. Each level map was parsed to identify similar features or spaces, then these were aligned by using a series of translations to produce a larger combined level map. Finally, single-cell artefacts were algorithmically removed to produce a "clean" level map.



(a) (b) [61]

Figure 12. Comparison of combined level with a commercial game level map

This process has produced what is an inherently playable level that immediate intrigues the game designer to inspect and analyse the topographical structure of the level. This level can be visually compared to a level from a commercial game with immediate parallels in terms of space distribution and ratios of distances and would be a candidate level for identifying whether procedurally generated content can produce immersion and engagement [62]. As a result, this process fits with in the definition of creative computing discussed by LIU [11] whereby there are "programs that can enhance human creativity without necessarily being creative themselves". Similarly, this as an example of co-evolution, enabled by feedback, which has been considered as an essential element of creative artistic and technical development [63]. The initial intention of using genetic algorithms as a generative system led to the conclusion that a simplistic fitness function was not sufficient to produce creative outcomes, which then resulted in reflective process between developer and game designer that led quickly to a pragmatic alternative that draws upon elements of cellular automata that produces usable and stimulating outcomes. Despite this, given that generation of original content has been stated as a challenge for PCG [52], clearly more effort is required in formulating an appropriate fitness function to differentiate levels that are likely to exhibit a sense of progression and purpose. This could be achieved through characterising the spaces created in the level as either rooms or corridors and seeking a balance between the two types. This could be further extended through formulating a fitness function that embraces novelty, an approach that has seen considerable successes in other domains [64, 65]

In addition to the formulation of the fitness function, the fragmentation of the space in the evolved levels could be due to a number of factors. Arguably, all of the evolved levels show a much greater percentage of open space than the randomly generated levels. In this sense, the genetic algorithm is successful in that it is managing to move levels towards the desired space ratio. Setting the desired space ratio lower could significantly alter the results.

The fragmentation may also be the result of the genetic operators disrupting the coherence of the space over time. This can be supported in part by inspecting the best level by generation. For most of the cases, the genetic algorithm typically maintained a stronger room/corridor feel for the first 20-30 generations and at that point

fragmentation occurred though the increasing significance of the mutation operator. A less aggressive mutation strategy would be valuable, as would a fitness function that can differentiate between levels with different characteristics. It is possible that because the fitness function is not differentiating between levels as most levels would include the start and end point in the main contiguous space. Rather than being a directed, randomised search the genetic algorithm would essentially be selecting completely random individuals for mating and as a result evolution is no longer occurring.

Future work will therefore focus on the development of a more robust fitness function in the first instance. This fitness function needs to encapsulate a design goal or indeed multiple goals, which would facilitate further work that would allow the potential value of evolutionary PCG approaches to be determined. A more rigorous approach to evaluating the content generation approach will be implemented [66] and the outcomes will be compared to a selected set of alternative algorithms selected through a comprehensive review of existing methodologies for PCG.

The research direction will include the development of goal-specific genetic operators that are targeted towards the non-fragmentation of game level maps. However, future research will also consider the compositing of sub-sections of level maps into a larger whole to address the challenges identified with scaling.

The challenge of generating original or novel content procedurally for game designers remains relevant, but the results in this paper indicate that relative simple fitness functions can produce playable levels and certainly at the smaller scale these exhibit many desirable characteristics. Similarly, the recombination of elements that in themselves may not be considered a creative outcome have the potential to produce levels that have features comparable to those produced by human game designers..

6. Conclusions

This paper has presented results that compare the use of a genetic algorithm as an evolutionary PCG approach with a simple random generation of game content. This comparison needs to take place in two dimensions, namely the quality of the resulting game content and the process of game content creation.

In terms of the quality of the resulting game content, the genetic algorithm can produce playable game levels but these levels do not have the same appeal as the randomly generated game levels. The space in the evolved levels is both more open and more fragmented, which suggests that the levels may not be as engaging for a player. This fragmentation is more apparent as the size of the game level increases.

However, in terms of the process of content generation, the genetic algorithm has been shown to be more consistent in terms of finding high quality solutions. In this instance, the genetic algorithm has been led somewhat astray by the formulation of a fitness function that doesn't capture whether a candidate game level is likely to be engaging. his stresses the importance of the role of the fitness function when using evolutionary algorithms, however this paper also provides insight in to the possibility of producing creative outcomes from essentially non-creative components.

References

[1] BLOW, J. (2004) Game development: Harder than you think. *Queue* **1**(10): 28-37.

[2] KÖHLER, B., HALADJIAN, J., SIMEONOVA, B., and ISMAILOVIĆ, D. (2012) *Feedback in low vs. high fidelity visuals for game prototypes*. In Proceedings of the *Second International Workshop on Games and Software Engineering: Realizing User Engagement with Game Engineering Techniques*, Zurich, Switzerland 2-9 June (Zurich, Switzerland IEEE), 42-47

[3] IOSUP, A. (2011) POGGI: generating puzzle instances for online games on grid infrastructures. *Concurrency and Computation: Practice and Experience* **23**(2): 158-171.

[4] TOGELIUS, J., CHAMPANDARD, A.J., LANZI, P.L., MATEAS, M., PAIVA, A., PREUSS, M., and STANLEY, K.O. (2013) *Procedural content generation: Goals, challenges and actionable steps*. In Proceedings of the *Dagstuhl Follow-Ups*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik),

[5] HENDRIKX, M., MEIJER, S., VAN DER VELDEN, J., and IOSUP, A. (2013) Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications* **9**(1): Article 1.

[6] EDWARDS, R. (2006) The economics of game publishing. *IGN Entertainment Inc*.

[7] RODEN, T. and PARBERRY, I. (2004) From artistry to automation: A structured methodology for procedural content creation. In Proceedings of *Entertainment Computing–ICEC 2004*. Springer.

[8] HUGILL, A. and YANG, H. (2013) The creative turn: new challenges for computing. *International Journal of Creative Computing* **1**(1): 4-19.

[9] YANG, H. and ZHANG, L. (2016) Promoting Creative Computing: origin, scope, research and applications. *Digital Communications and Networks* **2**(2): 84-91.

[10] LIAPIS, A., YANNAKAKIS, G.N., and TOGELIUS, J. (2014) *Computational Game Creativity*. In Proceedings of the *Fifth International Conference on Computational Creativity*, Ljubljana, Solvenia, 9th – 13th June (Ljubljana, Solvenia: Association for Computational Creativity), 46-53

[11] LIU, L. (2016) When intelligence meets data: game story generation by compositional creativity. *International Journal of Creative Computing* **1**(2-4): 274-307.

[12] BODEN, M. (2003) *The Creative Mind: Myths and Mechanisms*. London: Routledge.

[13] TOGELIUS, J., KASTBJERG, E., SCHEDL, D., and YANNAKAKIS, G.N. (2011) *What is procedural content generation?: Mario on the borderline*. In Proceedings of the *2nd International Workshop on Procedural Content Generation in Games*, 1-6.

[14] KHALED, R., NELSON, M.J., and BARR, P. (2013) *Design metaphors for procedural content generation in games*. In Proceedings of the *SIGCHI Conference on Human Factors in Computing Systems*, 1509-1518.

[15] SMITH, G., GAN, E., OTHENIN-GIRARD, A., and WHITEHEAD, J. (2011) *PCG-based game design: enabling new play experiences through procedural content generation*. In Proceedings of the *2nd International Workshop on Procedural Content Generation in Games*, 1-4.

[16] SMITH, G., WHITEHEAD, J., and MATEAS, M. (2010) *Tanagra: A mixed-initiative level design tool*. In Proceedings of the *Fifth International Conference on the Foundations of Digital Games*, ACM), 209-216

[17] TREANOR, M., BLACKFORD, B., MATEAS, M., and BOGOST, I. (2012) *Game-o-matic: Generating videogames that represent ideas*. In Proceedings of the *Procedural Content Generation Workshop at the Foundations of Digital Games Conference*,

[18] WHITEHEAD, J. (2010) *Toward proccedural decorative ornamentation in games*. In Proceedings of the *2010 Workshop on Procedural Content Generation in Games*, ACM, 9

[19] MARK, B. and BERECHET, T. (2014) *Procedural 3D Cave Generation*, MA thesis. IT University of Copenhagen, 2014.: http://benjaminmark. dk/Procedural_3D_Cave_Generation. pdf

[20] SMELIK, R., TUTENEL, T., DE KRAKER, K.J., and BIDARRA, R. (2010) *Integrating procedural generation and manual editing of virtual worlds*. In Proceedings of the *2010 Workshop on Procedural Content Generation in Games*, ACM, 2

[21] KRUSE, J., SOSA, R., and CONNOR, A. (2016) *Procedural urban environments for FPS games*. In Proceedings of the *Australasian Computer Science Week Multiconference*, 1-5.

[22] DORMANS, J. (2010) *Adventures in level design: generating missions and spaces for action adventure games*. In Proceedings of the *2010 workshop on procedural content generation in games*, ACM), 1

[23] BOURKE, P. and SHIER, J. (2013) *Space Filling: A new algorithm for procedural creation of game assets*. In Proceedings of the *5th Annual International Conference on Computer Games Multimedia & Allied Technology*,

[24] DRAGERT, C., KIENZLE, J., VANGHELUWE, H., and VERBRUGGE, C. (2011) *Generating extras: Procedural AI with statecharts*.

[25] LEE, Y.S. and CHO, S.B. (2011) Context-Aware Petri Net for Dynamic Procedural Content Generation in Role-Playing Game. *IEEE Computational Intelligence Magazine* **6**(2): 16-25.

[26] SMELIK, R.M., TUTENEL, T., DE KRAKER, K.J., and BIDARRA, R. (2011) A declarative approach to procedural modeling of virtual worlds. *Computers & Graphics* **35**(2): 352-363.

[27] SMITH, A.M. and MATEAS, M. (2011) Answer Set Programming for Procedural Content Generation: A Design Space Approach. *IEEE Transactions on Computational Intelligence and AI in Games* **3**(3): 187-200.

[28] NELLIS, A. and STEPNEY, S. (2014) Computational novelty: Phenomena, mechanisms, worlds. *H. Sayama, J. Rieffel, S. Risi, R. Doursat, & H. Lipson (Eds.), Artificial life* **14**: 506-513.

[29] GOERTZEL, B., PENNACHIN, C., and GEISWEILLER, N. (2014) Probabilistic Evolutionary Procedure Learning. In *Engineering General Intelligence, Part 2*. Springer.

[30] OZOLA, S. (2013) Synthesis of Nature and Art in Latvian Cities. *GENERATIVE ART 2013*: 234.

[31] BENTLEY, P. (1999) Aspects of evolutionary design by computers. In *Advances in Soft Computing*. Springer.

[32] KORDON, A.K. (2010) Evolutionary Computation: The Profitable Gene. In *Applying Computational Intelligence*. Springer).

[33] TOGELIUS, J., YANNAKAKIS, G.N., STANLEY, K.O., and 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.

[34] HARMAN, M. (2007) *The current state and future of search based software engineering*. In Proceedings of the *2007 Future of Software Engineering*, IEEE Computer Society), 342-357

[35] CONNOR, A.M., CLARKSON, P.J., SHAPAR, S., and LEONARD, P. (2000) *Engineering design optimization using Tabu search*. In Proceedings of the *Design for Excellence: Engineering Design Conference 2000*, John Wiley & Sons, 371-378

[36] BENTLEY, P.J. and WAKEFIELD, J.P. (1998) Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. In *Soft computing in engineering design and manufacturing*. Springer.

[37] DE BEIR, A. and VANDERBORGHT, B. (2016) *Evolutionary method for robot morphology: Case study of social robot Probo*. In Proceedings of the *11th ACM/IEEE International Conference onHuman-Robot Interaction*, IEEE), 609-610.

[38] CALDAS, L.G. and NORFORD, L.K. (2002) A design optimization tool based on a genetic algorithm. *Automation in construction* **11**(2): 173-184.

[39] JOACHIMCZAK, M., SUZUKI, R., and ARITA, T. (2015) Improving evolvability of morphologies and controllers of developmental soft-bodied robots with novelty search. *Frontiers in Robotics and AI* **2**: 33.

[40] COOK, R.G. and QADRI, M.A. (2013) The adaptive analysis of visual cognition using genetic algorithms. *Journal of Experimental Psychology: Animal Behavior Processes* **39**(4): 357.

[41] NORTON, D., HEATH, D., and VENTURA, D. (2014) *Autonomously Managing Competing Objectives to Improve the Creation and Curation of Artifacts*. In Proceedings of the *ICCC*, 23-32.

[42] TOGELIUS, J., YANNAKAKIS, G.N., STANLEY, K.O., and BROWNE, C. (2010) Search-based procedural content generation. In *Applications of Evolutionary Computation*. Springer.

[43] BARRETO, N., CARDOSO, A., and ROQUE, L. (2014) *Computational Creativity in Procedural Content Generation: A State of the Art Survey*. In Proceedings of the *2014 Conference of Science and Art of Video Games*,

[44] RACHMAWATI, L. and SRINIVASAN, D. (2006) *Preference incorporation in multi-objective evolutionary algorithms: A survey*. In Proceedings of the *IEEE Congress on Evolutionary Computation*, IEEE, 962-968

[45] KRUSE, J. and CONNOR, A.M. (2015) Multi-agent evolutionary systems for the generation of complex virtual worlds. *EAI Endorsed Transactions on Creative Technologies* **2**(5): e5.

[46] CARDAMONE, L., LOIACONO, D., and 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*, 395-402.

[47] WALSH, P. and GADE, P. (2010) *Terrain generation using an interactive genetic algorithm*. In Proceedings of

the *2010 IEEE Congress on Evolutionary Computation* IEEE, 1-7.
[48] ØLSTED, P.T., MA, B., and RISI, S. (2015) *Interactive evolution of levels for a competitive multiplayer fps*. In Proceedings of the *2015 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 1527-1534
[49] ASHLOCK, D., LEE, C., and MCGUINNESS, C. (2011) Search-based procedural generation of maze-like levels. *Computational Intelligence and AI in Games, IEEE Transactions on* **3**(3): 260-273.
[50] HARTSOOK, K., ZOOK, A., DAS, S., and RIEDL, M.O. (2011) *Toward supporting stories with procedurally generated game worlds*. In Proceedings of the *2011 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE), 297-304
[51] SORENSON, N. and PASQUIER, P. (2010) Towards a Generic Framework for Automated Video Game Level Creation. In DI CHIO, C., CAGNONI, S., COTTA, C., EBNER, M., EKÁRT, A., ESPARCIA-ALCAZAR, A.I., GOH, C.-K., MERELO, J.J., NERI, F., PREUß, M., TOGELIUS, J., and YANNAKAKIS, G.N. [ed.] *Applications of Evolutionary Computation: EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*. (Berlin, Heidelberg: Springer Berlin Heidelberg), ch.
[52] TOGELIUS, J., CHAMPANDARD, A.J., LANZI, P.L., MATEAS, M., PAIVA, A., PREUSS, M., and STANLEY, K.O. (2013) Procedural content generation: Goals, challenges and actionable steps. In LUCAS, S.M., MATEAS, M., PREUSS, M., SPRONCK, P., and TOGELIUS, J. [ed.] *Artificial and Computational Intelligence in Games* (Dagstuhl: Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik), ch.
[53] GREIG, T.J. (2016) *Evaluating the perceived immersion of procedurally generated game levels*, Master of Creative Technologies, Auckland University of Technology
[54] HOLLAND, J.H. (1975) *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*: U Michigan Press.
[55] HAIDAR, A., NAOUM, S., HOWES, R., and TAH, J. (1999) Genetic algorithms application and testing for equipment selection. *Journal of Construction Engineering and Management* **125**(1): 32-38.
[56] CONNOR, A.M. (1996) *The synthesis of hybrid mechanisms using genetic algorithms*, PhD, Liverpool John Moores University
[57] CANFORA, G., DI PENTA, M., ESPOSITO, R., and VILLANI, M.L. (2005) *An approach for QoS-aware service composition based on genetic algorithms*. In Proceedings of the *7th Annual Conference on Genetic and Evolutionary Computation*, ACM, 1069-1075
[58] GEIGEL, J. and LOUI, A.C. (2000) *Automatic page layout using genetic algorithms for electronic albuming*. In Proceedings of the *Photonics West 2001-Electronic Imaging*, International Society for Optics and Photonics), 79-90
[59] SIVANANDAM, S. and DEEPA, S. (2007) *Introduction to genetic algorithms*: Springer Science & Business Media.
[60] SCHMITT, L.M. (2001) Theory of genetic algorithms. *Theoretical Computer Science* **259**(1): 1-61.
[61] DOOMWIKI. (2005) http://doom.wikia.com/wiki/File:E3M4_map.png,
[62] CONNOR, A.M., GREIG, T.J., and KRUSE, J. (2017) Evaluating the Impact of Procedurally Generated Content on Game Immersion. *The Computer Games Journal* **6**(4): 209-225.
[63] BATTEY, B. (2016) Creative computing and the generative artist. *International Journal of Creative Computing* **1**(2-4): 154–173.
[64] BERNDT, D., FISHER, J., JOHNSON, L., PINGLIKAR, J., and WATKINS, A. (2003) *Breeding software test cases with genetic algorithms*. In Proceedings of the *36th Annual Hawaii International Conference on System Sciences*, IEEE, 10 pp.
[65] MOURET, J.-B. (2011) Novelty-based multiobjectivization. In *New horizons in evolutionary robotics*. Springer.
[66] SHAKER, N., SMITH, G., and YANNAKAKIS, G.N. (2016) Evaluating content generators. In *Procedural Content Generation in Games*. Springer International Publishing.