2013

Using genetic algorithms to find cellular automata rule sets capable of generating maze-like game level layouts

Andrew Pech
Edith Cowan University

Using Genetic Algorithms to Find Cellular Automata Rule Sets Capable of Generating Maze-Like Game Level Layouts

A dissertation submitted in partial fulfilment of the requirements for the degree of

Bachelor of Computer Science (Honours)

By: Andrew Pech
Student ID: 10185597

School of Computer and Security Science
Faculty of Health, Engineering and Science
Edith Cowan University, Mt Lawley, WA, Australia

Supervisor(s): Dr Martin Masek, A/Prof Peng Lam, A/Prof Philip Hingston

Date of submission: 4 Nov 2013

Table of Contents

Abstract

The video game industry has grown substantially over the last decade and the quality of video games has also been advancing rapidly. In recent years, video games have been advancing to a point that the increased time required to manually create their content is making this process too costly. This has made procedural content generation a desirable option for game developers due to its speed of generating content, and the variety of content that a single PCG method can produced.

The main purpose of this dissertation is to detail a new approach to procedurally generate video game level layouts, and to aid in further research in the area of procedural video game content generation. The new PCG approach investigated and developed in this study combined a genetic algorithm with cellular automata and a maze generation technique into a method for generating game level layouts with desired maze-like properties. The GA in this approach was utilized to evolve CA rules that, when applied to a maze configuration, would produce layouts with desired properties.

This research discovered that CA rules could be evolved to generate level layouts with desired properties, and that there were a number of parameters which could affect the layouts these rules produced. These parameters include the number of cell states used in the CA, as well as the CA's neighbourhood size and the number of times the CA rules were applied to their maze configurations. This research also discovered that the one factor that had the largest impact on the visual aspect of the generated layouts was the chosen chromosome representation.

List of Figures

List of Tables

Definition of Terms

Attribute Similarity Measure (ASM): A value that is assigned to a generated level layout that represents how similar its attributes are to its goal layout's attributes.

Cellular Automata (CA): A process that alters a cells state over time using a definite set of rules involving the states of surrounding cells.

Evolutionary Algorithm (EA): A population-based metaheuristic optimization algorithm that attempts to mimic the biological process of evolution to achieve an optimal solution to a problem.

Genetic Algorithm (GA): A common form of evolutionary algorithm that uses mutation and crossover operators to evolve data.

Game Level: A defined area of a video game, this is usually the area that is currently loaded into memory.

Game Level Layout: The structure of a game level excluding minor details such as items, players and small static objects.

Procedural Content Generation (PCG): The process of generating media content algorithmically.

Chapter 1. Introduction

1.1 Background

Level design in video games has always been a large part of game development and has mostly been accomplished by manual means. In the early years of game development, video games were generally created by a single person who designed and programmed the game as well as developed artistic resources. As video games became larger, more people were required to develop them. Lode Runner, a video game developed in 1983, was one of the first games that had a separate individual whose sole job was level design. Until recently, the development process time was quite well balanced between programmers and artists/designers. With the rapid advancement in technology, video games have become larger and more complex, requiring more work from the artists and designers. This has led to increased costs, due to the extra development time required, and thrown out the balance in time between developing resources and programming. This has increased the motivation to employ cost reduction techniques, such as the procedural generation of content.

Procedural content generation (PCG) is a term commonly used to describe the process of generating media content algorithmically rather than manually. PCG has been used in video games in various forms, either to alleviate the burden of the design process or simply to increase a game's replay value. An early example is the classic 1980 game, Rogue (Toy & Wichmann, 1980). Rogue used PCG to generate new and unlimited levels, in the form of 2D "dungeon" environments. An example is shown in Figure 1. Since then, many other games such as NetHack (NetHack DevTeam, 1987), Moria (Koeneke & Todd, 1994), and Diablo (Blizzard North, 1996) have also mimicked Rogue's use of PCG to generate an unlimited number of random dungeons, adding to the game replay value as the players are continuously presented with new environments and designs.

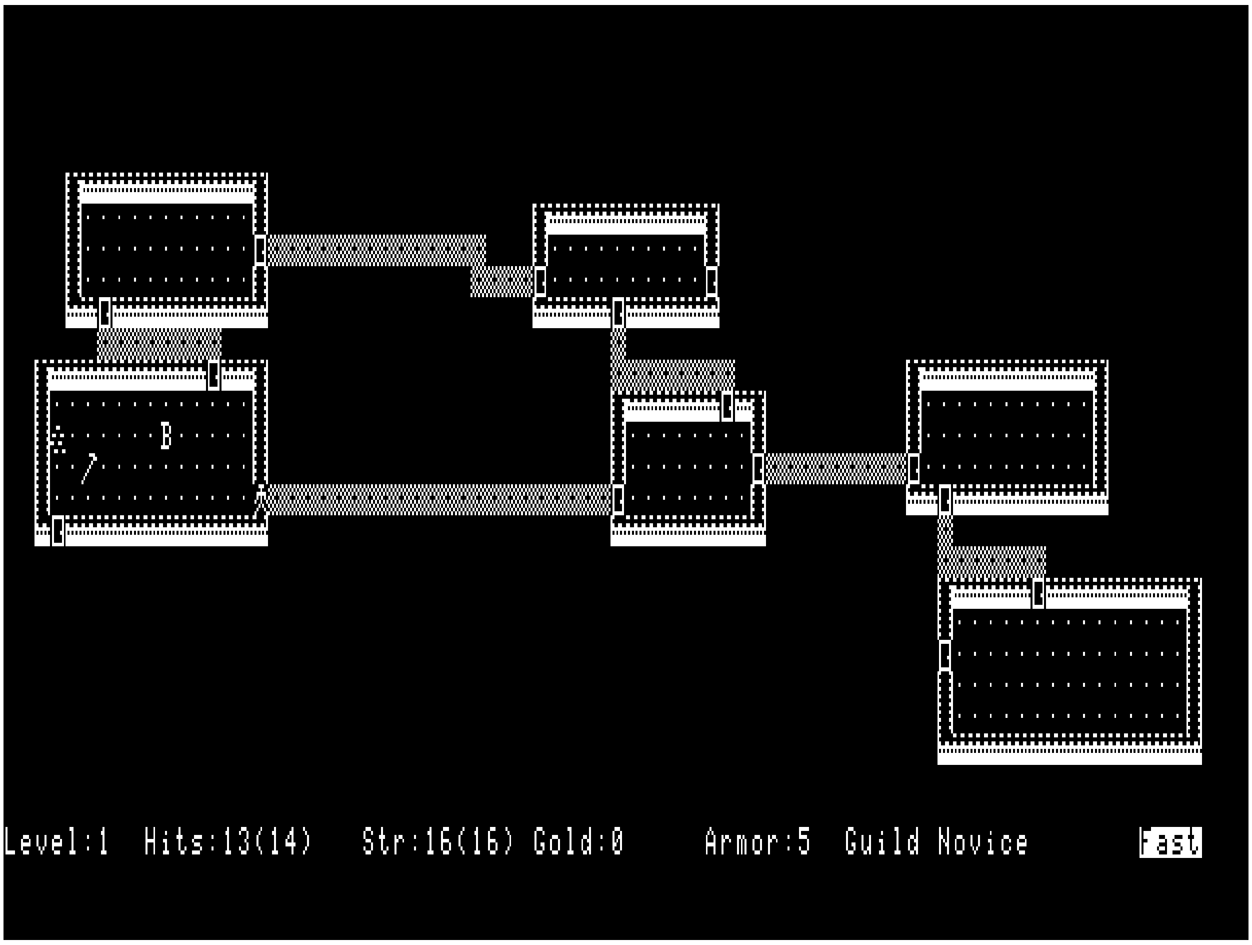

Level:1 Hits:13(14)   Str:16(16) Gold:0   Armor:5 Guild Novice   Fast

Figure 1. Screen shot of 1980's classic game "Rogue" (Retrieved from the L. Curtis Boyle Website)

Besides level-map generation, PCG has also been used for numerous other applications in game design and development. SpeedTree (2002) is an example of a software package that generates realistic looking trees, ranging over 150 species of trees and infinite variants. This makes production of vegetation to populate virtual worlds easier and faster.

According to Guðlaugsson (2006), the concept of procedural content generation originated from fractals (Mandelbrot, 1982), a concept that was discovered by Benoît Mandelbrot in 1975. A fractal is a basic shape that when repeated, creates a more complex object. This pattern can be found throughout nature and the concept can be used to procedurally reconstruct irregular phenomena. Some PCG approaches use fractals to generate artificial representations of natural phenomena, such as mountains, coastlines, lunar landscapes and music, although they will not produce exact replicas of particular phenomena and therefore titled "fractal forgeries" by Mandelbrot.

While PCG has been used in video games for decades it has only recently been paired with Cellular Automata (CA) to produce game level layouts (Johnson et al., 2010). Wolfram (1983) describes an "elementary" cellular automaton as a sequence of sites carrying values of 0 or 1 arranged on a line. More complex versions involve sites that can take on one of a number of states, and may be arranged in higher dimensions. The value at each site changes deterministically with time according to a set of definite rules involving the states of its neighbouring sites. CA were introduced by Von Neumann and Ulam (Von Neumann & Burks, 1966) as a simple mathematical model in which to study biological processes, such as self-reproduction, and have since been used for physical and computational systems (Chopard & DROZ, 1998; Mitchell et al., 1996). Some computational systems where CAs have been applied used genetic algorithms (GA) to discover rule sets that could perform the desired task. GAs are also a common search/optimization technique used in certain PCG methods including Ashlock's (2011) work, where GA was used to evolve level layouts directly with a customizable fitness function that could control the paths that the layouts contained.

The GA concept was introduced by John Holland (1992) in the mid 1960's and was designed to solve problems that humans did not fully understand. In the late 1950's and early 1960's the use of evolution in computer science heavily relied on mutation until the introduction of GAs. These algorithms attempted to find optimal solutions to problems by emulating the biological process of evolution through natural selection, mutation and reproduction.

This study developed an approach, combining GA with CA into a PCG method for generating game level layouts with desired properties. This approach uses the GA to evolve CA rule sets that are capable of generating 2D maze-like game level layouts.

1.2 Purpose

This project focused on generating rule sets for CA, using a GA, so as to produce maze-like game level layouts. The specific aims of this study are stated below.

- To explore what fitness functions are most useful in finding CA rule sets to produce maze-like game level layouts.
- Explore the effects of different CA rule set parameters for generating maze-like level layouts.
- To develop an approach to finding CA rule sets that can produce video game level layouts.

1.3 Significance

This section discusses the significant aspects of the research in this area. There are three significant aspects of this research and they are listed below.

- To expand knowledge in the growing field of PCG and its use in generating game level layouts. So far there is limited knowledge on using cellular automata with PCG to generate game level layouts and this research will add to known PCG methods in this area and assist in future research.
- The ability to automatically find cellular automata rule sets that can produce maze-like game level layouts with infinite variety, increasing video games replay value.
- Also to reduce the work load of developers, and therefore development costs.

1.4 Research Questions

The main focus of this project is based on the question:

"How can rule sets for cellular automata be evolved so as to produce maze-like game level layouts?"

To aid in this research, the following sub questions were also considered:

- What are suitable fitness functions that can evaluate a CA rule sets ability to produce maze-like game level layouts?
- How will different CA rule set parameters affect the generated maze-like game level layouts?

1.5 Contributions of this Study

This study makes contributions in the area of procedural content generation for video games, specifically in the use of cellular automata for generation of maze-like game level layouts. The detailed contributions of this research are listed in the following points.

- **Exploration of two chromosome representations for cellular automata rule tables for use in developing maze-like game level layouts.**
  Evolving CA rules is not a well explored area. The majority of work done in the area use a representation with a direct mapping from genotype to phenotype (Mitchell et al., 1996; Piwonska & Seredynski, 2010). This research experimented with the direct representation as well as an indirect representation to explore what effects using each

representation had on the level layouts that the chromosomes produced. This study found that chromosome representation had a significant effect on the visual appearance of the generated layouts, with indirect representation generally producing neater and more structured layouts.

- **Experimentation with the idea of "flavours" to explore CA with more than two cell states.**
  Generally when using CA to generate level layouts, only two cell states are used, one to represent traversable areas, and another to represent non-traversable areas. This can be seen in Johnson et al.'s (2010) work, where each cell in the CA was set to either the floor state (traversable) or the rock state (non-traversable). This study represented layouts in a similar manner, where each cell represented either a traversable area, or a non-traversable area, therefore the cells in the CA were set to one of two states, the traversable state, or the non-traversable state. But to allow for a more expressive rule set in the CA, the idea of "flavours" was used. The principle of this idea is that the actual cell state (flavour) is divided into a number of sub states for the purpose of rule set application. In this research two flavours were used, the traversable flavour, and the non-traversable flavour. This idea allowed the CA in this research to use more than two cell states while each cell via the idea of flavours is still associated with either a traversable or non-traversable area.

- **Introduces a unique method of extracting attributes from 2D level layouts.**
  As part of the evaluation process used in this research, attributes of 2D level layouts were compared to desired attribute values to determine their similarity. As this area is not well covered in the literature, there was no suitable automatic technique to extract the level attributes used in this study. To address this issue, this research developed a unique approach, incorporating image analysis techniques, which can automatically extract attributes from level layouts. These attributes include the number of traversable areas, their largest and average size, number of passageways and their average length, number of rooms and their average size, number of dead-ends, and number of culs-de-sac.

- **An approach combining maze generation with CA and GA to produce level layout.**
  To date, limited work has been done in the area of 2D maze-like level generation, including work by Ashlock et al. (2011) and Johnson et al. (2010). Ashlock's (2011) approach used GA to evolve level layouts directly. The drawback to this approach is the slow evolutionary process, making this approach less ideal for run time level generation. Johnson's (2010) approach used manually designed CA rules to generate cave-like levels, but manually designing CA rules can be a difficult process especially when complex results are desired. The approach developed in this research used a combination of GA and CA, an approach already attempted in other fields but not in the area of PCG. This approach used GA to automatically find CA rules that, when applied to maze configurations, result in 2D level layouts with desired properties. This approach addresses the issue of manually designing CA rules by using a GA to evolve CA rules which, once generated, can produce a number of similar layouts with a particular style in a short space of time. This makes the result of the evolutionary process ideal for run time level generation. This approach involves the use of a modified recursive backtracker algorithm to generate a collection of initial maze configurations, a unique method of extracting attribute values from 2D level layouts for evaluation, and a genetic algorithm to evolve CA rule tables.

1.6 Structure of the Thesis

This thesis is divided into a total of 6 chapters and this section briefly summarizes the content of each.

- **Chapter 1** introduces the subject of this study, including background information of PCG and a brief description of cellular automata and genetic algorithms, before outlining its purpose and significance, and then discussing the research questions that this study aims to answer. It also lists the new contributions to knowledge made by the study.
- **Chapter 2** presents an extensive literature review that covers all aspects related to this study. It gives a detailed description of PCG and its different types, a description of genetic algorithms and how they have been used to evolve cellular automata rules, and various PCG methods used to develop video game levels. Chapter 2 also gives a detailed description of existing techniques that were used in this study.

- **Chapter 3** explains the proposed approach in detail, describing the generation of a population of perfect mazes, the chromosome representations that were explored, how the chromosomes were evaluated, and the steps of the GA process.
- **Chapter 4** details the results obtained from the proposed approach by performing visual comparisons between generated layouts and their goal layout. This is followed by analysis of variance tests to determine which factors associated with the GA and CA had a significant impact on the results that this approach achieved.
- **Chapter 5** brings the study and this thesis to a close by providing concluding remarks on the work done, discussing possible directions for future research, and providing a summary of answers to the research questions posed at the beginning of chapter 1.

1.7 Summary

This chapter has provided a basic premise for the study by providing relevant background information on procedural content generation, and discussing what this study aims to achieve and how this may be significant in the field of PCG. The research questions that this study aims to answer have also been presented here. To set this study in the context of the existing body of knowledge, an extensive review of current PCG techniques and methods of evolving CA rules was performed. This is presented in the next chapter.

Chapter 2. Literature Review

This chapter details the existing literature on procedural content generation and its various elements along with a review of the techniques that were used in this research. The chapter is presented in two main sections, the state of play and technical review. State of play gives a broad description of PCG and where it has been used, focusing on its use in game level design. The technical review details genetic algorithms, cellular automata and the recursive backtracking maze generation method used in the game level layout generation technique developed in this study.

2.1 State of Play

PCG has found wide use in game development, including game level generation, and this section outlines existing techniques used for this process. This section is divided into three sub-sections. Section 2.1.1 on procedural content generation describes the main types of PCG and the different ways in which PCG is used to generate content. Section 2.1.2 covers previous work that deals with evolving rule sets for cellular automata using genetic algorithms (GAs). Section 2.1.3 details the different attributes that make up a maze generation algorithm and a collection of known maze generation algorithms, and section 2.1.4 describes existing PCG algorithms for generating game levels and layouts.

2.1.1 Procedural Content Generation

PCG can be used to generate many different types of content. In the video game industry it has been used to generate 3D models, textures, game levels, rule sets and even the behaviour of non-player characters (“SpeedTree”, 2003; Perlin & Hoffert, 1989; Togelius et al, 2010). Togelius et al (2011) listed five aspects that should be considered when designing a PCG algorithm for a video game so as to tailor the algorithm to specific needs. These aspects are: online vs. Offline content generation, essential vs. non-essential content, random seed vs. vector parameters, stochastic vs. deterministic generation and constructive vs. generate and test. They will now be discussed.

Online content generation defines the process of generating content during runtime of the application, whereas offline content generation is the process of generating content outside of the application and there are pros and cons to using either. Online generation has the advantage of in-game variation, where infinite variations of content can be generated constantly presenting new content to the player. For example, for a maze game, content that may be procedurally generated during runtime could be the maze layout, producing infinite different mazes for the player to solve. Disadvantages of this approach include the absence of human creativity and the computation time of these algorithms. In most cases, it is faster to read content in from a storage medium than to produce it algorithmically. Offline generation is the opposite, its prime advantage is that CPU cycles will not be spent generating content as it has already been created and needs only to be read into the game from a storage medium. Also, developers can make artistic alterations to content if it is generated offline. The primary drawback to this method is that the player will only experience the limited set of content that was produced during development. An example of offline generation during game development could be textures applied to models as developers may prefer this content to remain static. This content may also be enhanced by developers; ensuring developers keep control over the appearance of the game.

Essential content versus non-essential content questions how tightly the requirements for generated content must be met. Essential content is defined as content that must fit some criteria otherwise game progress is halted, whereas non-essential content can take on any form and still not impede the progress of the game. Non-essential content can often forego the additional processing required to ensure that it will be generated correctly. In the case of the maze example, essential content would be the path from start to finish. If such a path did not exist the player could not complete the maze, meaning the game could not progress. Wall textures on the other hand are not so essential, as their look will not impede game progression.

In the case of random seeds vs. parameter vectors, the type of input to the algorithm is questioned. The inputs are the parameters that get passed to the algorithm and ultimately affect the generated content. The type of input affects the level of control over the generated content. There are two common types of input, random seeds and parameter vectors. Random seeds provide the algorithm with a seed for a random number generator which it uses to obtain unspecified values, which greatly reduces the control over the algorithms output. Parameter vectors are lists of values, which are passed to the algorithm to influence generated content and give greater control over the algorithms output. Parameter vectors need not be simple static constants. PCG can take a player experience model (PEM) as input to customize a game based off player experience. This experience-driven PCG uses a PEM to determine what and how content is created. There are numerous ways to achieve player experience modelling, Yannakakis et al (2011) provides several methods on accomplishing this, from direct approaches, such as asking players for feedback, to more advanced methods, such as monitoring physiological changes in players during play. An overview of experience-driven PCG for platformer level design is presented in Shaker et al (2010).

The difference between stochastic generation and deterministic generation is the amount of randomness between instances of generated content that were generated using the same algorithm with identical input parameters. Deterministic generation algorithms will always produce the exact same content if given the same parameters while stochastic generation functions will display some randomness between instances of generated content. In this case random seeds are not considered parameters otherwise all PCG algorithms would be deterministic. The video game "Rogue" used a stochastic PCG algorithm to generate its dungeon layouts, so that each dungeon layout was different but followed the same guidelines specified by the algorithms input parameters.

Although there are many types of PCG, each PCG algorithm can be labelled as either constructive or generate and test. The aim of PCG is to generate content that meets required criteria. Content that meets these criteria is known as correct content. Constructive and generate and test methods of PCG differ in their approach to generate correct content. Constructive PCG algorithms always terminate after a set number of steps, but must have rules in place to ensure that the content is correct while it is being generated. A generate and test PCG algorithm does not necessarily produce correct content every time. This method of PCG will continuously generate new content in an iterative manner and evaluating it by a fitness function until it eventually generates correct content. This form of PCG is often enhanced with a search/optimisation technique which is referred to as search-based PCG.

Fitness functions, also known as evaluation functions, are methods that determine how closely the generated content meets the requirements. According to Togelius et al (2011) there are three key classes of a fitness function in PCG for video games. These three classes are direct, simulation-based and interactive. Direct fitness functions map particular attributes of generated content to a fitness value, such as the number of paths that lead to the exit of a maze. Simulation-based fitness functions rely on computer agents to play through the generated content and compute the fitness value based on the agents experience. An example would be to use a computer agent to navigate a maze and rate it on how long it took and how many dead ends the agent followed. Interactive fitness functions rate a piece of contents fitness based on player interaction such as number of times and the length of time the player interacted with the generated content.

There are two primary ways that a fitness function is used in generate and test PCG. In standard generate and test PCG methods, fitness functions result in a simple pass or fail. For example, if the generated content is a maze, the fitness function would test if there is an existing path from the start point to the end point, and if there is not, the fitness function will result in a failure. Whereas search-based PCG uses its fitness function to grade generated content using one or a vector of real numbers. This form of PCG attempts to produce content with higher grades of fitness by using stochastic or metaheuristic search techniques to upgrade content, rather than regenerate it.

Search-based approaches to PCG can use any form of stochastic or metaheuristic search techniques but most commonly incorporate genetic algorithms to evolve generated content. Genetic algorithms have also been paired with cellular automata to evolve cellular automata rule sets, which is detailed in the next section.

2.1.2 Genetic Algorithms and their use in Evolving Cellular Automata Rule Sets

A genetic algorithm is a population based search metaheuristic that generates a population of chromosomes which it evolves over time. Chromosomes are data representations of possible solutions that go through the selection, mutation and reproduction process. The process of selection in genetic algorithms involves a fitness function, which is used to rank the chromosomes so that the best can be selected to contribute to future generations. The selected chromosomes are mutated and 'mated' in an attempt to produce greater solutions. The process of 'mating' chromosomes together is an operation commonly referred to as crossover which exchanges genetic material between chromosomes.

Cellular automata are relatively simple processes that are capable of complicated behaviour. A cellular automaton consists of a grid of MxN cells, with each cell being set to one of a number of states, and a set of rules that are applied to each cell in the grid synchronously for T iterations. The CA rule set governs the state of a cell, C_{mn}, within the CA at a given time step based on the state of the cell's neighbourhood at the previous time step. Starting with an initial configuration for the CA grid, the rules are applied iteratively until the final configuration is reached. Although a simple process itself, one of the main bottlenecks when using cellular automata is discovering rules that will perform the desired task. There are many ways to do this including the use of genetic algorithms to search for optimal CA rule sets for particular tasks. Two tasks that have used this approach in the past are the density classification task (Mitchell et al., 1996) and the pattern reconstruction task (Piwonska & Seredynski, 2010).

The density classification task involved a one dimensional grid of cells where each cell represented one of two values, a 0 or a 1. The goal of this task was to use cellular automata to discover the majority value contained in the grid by using a set of rules that would convert each cell to the majority value by the final configuration. The method used to find an optimal rule set that performed this task was initialized by generating two populations, a population of chromosomes and a population of initial configurations for the grid values. The population of initial configurations were static while the population of chromosomes, which represented rule tables for the cellular automaton, were evolved by the genetic algorithm. Each initial configuration was processed with a cellular automaton using each of the populated rule sets. The fitness function graded each rule set based on the percentage of correct final configurations it produced. The most common rule set produced from this algorithm was labelled the expanding block rule set. This rule set yielded accurate results for the initial configurations but it did not scale well with larger initial configurations. Another rule set that was discovered was labelled the particle-based rule set which had a higher success rate than the expanding block method and degraded less rapidly on larger grid sizes.

The pattern reconstruction task also used a genetic algorithm to reduce the search for CA rule sets that could perform the desired task. The pattern reconstruction task used a two dimensional grid of cells, that were initialized to either state 1 or state 2, to form a pattern as the initial configuration. The initial configuration then had a percentage of its cells set to an unknown state labelled as state 0. The task of the cellular automaton was to reconstruct the original pattern by identifying and correctly converting cells in the unknown state back to their original state. The genetic algorithm used in Piwonska and Seredynski's approach used cellular automata rule tables as chromosomes. The cellular automaton was run on the initial configuration using each of the populated rule sets until final configurations were achieved. The fitness function graded the final configurations with a value equal to the number correct cells in state 1 plus the correct cells in state 2 minus the number of cells in the unknown state. Piwonska and Seredynski's tested their approach on two different pattern types and accurate results were found for both. The resulting rule sets performed the pattern reconstruction task with between 89% and 100% accuracy, depending on the percentage of cells in state 0 in the initial configuration. The rule sets discovered in this research performed more accurately on larger grid sizes.

Cellular automata have been used in various other computer science applications including the generation of maze-like level layouts in video games. Some video game level layouts closely resemble mazes and the next section covers different types of mazes and known PCG algorithms that generate them.

2.1.3 Maze Generation

There are many different types of mazes, and the "Think Labyrinth" (2010) website suggests that each maze is made up of seven different attributes, dimension, hyperdimension, topology, tessellation, routing, texture and focus. These attributes are described below.

- The **dimension** attribute refers to how many dimensions in space the maze covers. The types of this attribute include 2D mazes, 3D mazes, higher dimension mazes, which are mazes with more than three dimensions, and weave mazes, which are 2.5D mazes that have passages that weave over and under one another.
- The **hyperdimension** attribute refers to the dimension of the object that moves through the maze. Generally mazes are non-hypermazes where the object that gets moved through it is a point or small object, where-as in a standard hypermaze a line is used to traverse it rather than a simple point and is significantly more complex. A hyperhypermaze, also known as a hypermaze of the second order, is more complex than a hypermaze and can only exist in four or more dimensions, requiring a plane to traverse the maze.
- The **topology** attribute defines the geometry of the space that the maze exists in and is classified as either normal Euclidean space, or planair, which refers to mazes with abnormal topology.
- The **tessellation** attribute refers to the geometry of the individual cells that make up a maze and include orthogonal, delta, sigma, theta, upsilon, zeta, omega, crack and fractal. Orthogonal tessellation is the most common form and consists of a rectangular grid where passageways intersect at right angles; this literature review will not cover the rest of the tessellation types as they are more abstract and less common.
- The **routing** attribute defines the type of maze generated and is probably one of the most important aspects of a maze. These types include perfect mazes, braid mazes, partial braid mazes, unicursal mazes and sparse mazes. Perfect mazes are mazes that have no loops, closed circuits and no unreachable areas. Only one path exists

between any two points in a perfect maze. Braid mazes contain no dead ends and therefore consist of looping passageways. Partial braid mazes consist of both loops and dead ends. Unicursal mazes, also known as labyrinths, are mazes that have no junctions and are therefore made up of a single loop. A sparse maze is one that does not carve passageways into every section of the maze, leaving some inaccessible areas.

- The **texture** attribute defines the style of passageways in a maze. Types of this attribute include bias, run, elitism, symmetry and river factor.
  - **Bias** - where passageways are more likely to travel is a particular direction.
  - **Run** - where passageways are less likely to turn causing longer passageways.
  - **Elitism** – the more elite a maze is, the more direct the passageway from start to finish is.
  - **Symmetry** - where portions of the maze are rotated and flipped around its centre.
  - **River Factor** - where fewer branches are formed but flow to form longer dead ends.
- The **focus** attribute simply refers to the way in which a maze is created and can either be a **wall adder**, where passageways are formed implicitly by adding walls, or a **passage carver**, where passageways are formed explicitly by carving them out of a gird or other area. Template is another focus type which is generally a combination of a wall adder and passage carver.

Out of the mentioned attributes, routing is usually the most definitive and a maze is usually categorized by its routing type. Perfect mazes are the most common type of maze and many algorithms exist that create them. The other types of mazes, braid, partial braid, unicursal and sparse, can all be created from a perfect maze. By eliminating dead ends, or a percentage of dead ends, from a perfect maze, a braid, or partial braid, maze can be formed (Roth et at., 2010). Unicursal mazes can be formed from a perfect maze by bisecting each of its passageways which creates a loop. Table 1 displays eleven known algorithms for producing perfect mazes and lists three of their texture attributes as well as their focus. The algorithms will now be discussed in terms of algorithm groups that produce mazes with similar properties.

Algorithm Name	Is Biased	Elitism	River Factor	Focus
Recursive Backtracker	No	0.05	High	Passage
Prim’s	No	0.43	Low	Either
Kruskal’s	No	0.24	Low	Either
Aldous-Broder	No	0.22	Low	Either
Wilson’s	No	0.22	Low	Either
Hunt and Kill	Yes	0.10	High	Either
Growing Tree	No	-	-	Either
Eller’s	Yes	0.24	Low	Either
Recursive Division	No	0.14	High	Wall
Binary Tree	Yes	0.50	High	Either
Sidewinder	Yes	0.38	High	Either

Table 1. Three texture attributes and focus for eleven known perfect maze generation algorithms.

As shown in Table 1 there are three algorithms that produce un-biased mazes with a high river factor and low elitist factor. These are the recursive backtracker, hunt and kill and recursive division algorithms. The recursive backtracker is the most commonly referenced algorithm in the literature and forms perfect mazes using the passage carving focus. This algorithm works by carving passages from a random cell in a grid and branching out from previously carved cells once dead ends have been reached. The hunt and kill algorithm runs similar to the recursive backtracker except, when a dead end is found, the algorithm doesn’t branch from a previously carved cell but hunts for a random uncarved cell to start a new passage that eventually attaches itself to an existing passageway. The recursive division algorithm differs from the previous two as it is exclusively a wall adding algorithm. This method of maze generation creates random walls, either vertical or horizontal, and places random openings along them. This process is repeated until the maze is filled.

On the other end of the scale, there are three algorithms that produce mazes with a low river factor and high elitist factor. These algorithms are Prim’s, Kruskal’s and Eller’s. Prim’s algorithm was originally developed in 1930 by a Czech mathematician named Vojtě*ch* Jarník to find a minimum/maximum spanning tree for a graph. It has since been re-discovered twice, first by an American mathematician named Robert Prim in 1957, and again in 1959 by Dutch computer scientist Edsger Dijkstra (Mička, 2013). Prim’s algorithm generates mazes in a similar manner to the recursive backtracker except it does not need to hit a dead end before it branches its passageways. Kruskal’s algorithm, developed by Joseph Kruskal

(1956), was designed to find a graph's spanning tree of minimum length but has since been used to produce mazes. Unlike the previously mentioned passage carvering algorithms that carve their passages outward like a tree, Kruskal's algorithm carves passageways between cells randomly throughout the grid. Eller's algorithm is one of the fastest and most memory efficient methods of generating perfect mazes. This method creates mazes one row at a time by randomly joining cells in each row, forming sections. Each section then merges with at least one cell in the next row and the process is repeated on the new row.

The Aldous-Broder algorithm and Wilson's algorithm produce mazes with the exact same traits. The Aldous-Broder algorithm is one of the slowest and least intelligent algorithms for generating perfect mazes, but mazes it not what this algorithm was originally intended for. Two researchers, Aldous (1990) and Broder (1989), developed this algorithm independently while investigating uniform spanning trees. This algorithm works by selecting a random cell and moving to a random neighbour, if the neighbour has not been visited yet, then a passage is carved between it and the previous cell. This is quite inefficient because the algorithm can move around visited cells while not carving any passages and relies on random chance that unvisited cells will be visited. Wilson's algorithm was developed by Wilson (1996) as a faster method of generating random spanning trees than the Aldous-Broder algorithm. This algorithm works by selecting a random cell that is not part of the maze and moving to random neighbours until it finds a cell that is part of the maze, then a passage is carved along the path that was traversed.

The growing tree algorithm is a useful algorithm that can create prefect mazes of different textures. To achieve this, the algorithm adds all carved cells to a list until a carved cell has no unvisited neighbours which it is then removed from the list. Each time the algorithm carves into a new cell, it selects a cell from the list to carve from. The texture of the maze depends on how the cell is selected from the list. If the last cell is always chosen, then the algorithm creates mazes the same as the recursive backtracker, other configurations produce different results.

There are two algorithms that produce very biased mazes with high elitist factors, the binary tree and sidewinder algorithms. The binary tree is the simplest and fastest method of producing perfect mazes but passageways will only span a single direction. This algorithm is performed one row at a time by iterating over each cell and carving a passage into one of its neighbouring cells. The direction of the chosen neighbour can only be one of two directions including a vertical direction, up or down, and a horizontal direction, left or right. The sidewinder maze generation algorithm is performed one row at a time by iterating over each cell and choosing whether to carve horizontally or not. Once the algorithm has decided not to carve any further along its horizontal direction, it will pick a cell from the freshly carved passage at random to carve upwards into the previously processed row until the maze is complete.

Although standalone mazes may be used for game level layouts and have been in the past, an early example is “Pac-Man”, they can also be used as a basis for more complex level designs. The next section “Game Level Generation” details different methods of generating game levels and layouts including methods that use forms of maze layouts.

2.1.4 Game Level Generation

There are many different types of game levels ranging from 2D platformers, to strategy maps, to fully three dimensional open worlds. Variations of PCG approaches exist that can generate each level type. This section will briefly describe existing methods of game level generation, focusing more on methods that involve the use of maze generation and cellular automata.

One example of a search-based approach to procedurally generating game levels is presented in Togelius et al (2010) which proposes a PCG method of generating strategy game maps. The presented algorithm provides the ability to place player bases and up to two types of resources around a map that it also generates a height map for. This PCG method has a relatively small genotype made up of a collection of four different types of components. Each component is represented by a vector of real values that range between 0 and 1. The components are the mountains, stored as an X and Y coordinate and a height weight, the player bases, stored as an X and Y coordinate, the resource type 1, stored as an X and Y coordinate and the resource type 2 which is also stored as an X and Y coordinate. This algorithm used evolution to produce maps with certain characteristics based off different fitness functions. In the article, five fitness functions were used to grade the generated maps on attributes including distance between player bases, ground level of the placed bases, elevation difference between symmetrical cells, distance between resources and bases and distance between resources and other resources. Another contribution presented in the article was the ability to select up to two possibly conflicting map attributes to grade the content on. This was accomplished by using a multiobjective evolutionary algorithm (MOEA) which, rather than search for an optimal solution, collects a Pareto front of non-dominated solutions, which are multiple solutions for which there are no other solutions that are better or equal in all dimensions. The collected candidates are then evolved using standard recombination/mutation operators.

Another two search-based PCG algorithms were presented in Ashlock (2010) which generated puzzle game levels. The two algorithms are used to generate two types of puzzles, the chess puzzle and the chromatic puzzle. The chess puzzle algorithm starts by placing chess pieces at random on a grid and blocks all grid cells that the chess piece could move to in a single turn, rather than move the chess piece during play. The player is itself a chess piece and can only move through the maze-like level using legal moves for that chess piece. The chromatic puzzle algorithm starts by setting each cell in a grid to one of the seven colours of the rainbow randomly. The player can only move from one cell to another if the colours are the same or next to one another on the colour wheel. These algorithms used the same fitness function which graded content based on the minimum number of moves it would take the player to complete the level. This method of controlling difficulty was not perfect due to the fact the fitness function only returned the minimum number of moves required to complete the game, assuming that the higher the value the harder the puzzle, which is not always the case. The difficulty of these puzzles decreases as the gap between the minimum and maximum number of moves required to complete the puzzle decreases, which is a natural side effect of increasing the minimum number of required moves too much.

Ashlock et al (2011) also proposed another search-based PCG method for generating maze-like levels but takes a step away from puzzle levels to produce 2D top-down game levels. The method proposed used four different types of representation, to produce levels that varied visually, and five different fitness functions to control the attributes of the levels. The four types of representation were direct 1, which used a binary gene, direct 2, which used a chromatic map, indirect positive, which added walls to a traversable area, and indirect negative, which carved rooms out of a non-traversable area. The direct 1 representation is specified as a grid of values that are either 1 or 0, with one value specifying traversable area and the other specifying non-traversable area. Direct representation 2 populates a grid with colour values, as described earlier, where passageways are formed by cells of the same colour and colours adjacent on the colour wheel. The indirect positive representation uses an array of integer values to specify wall locations, directions and sizes and the indirect negative representation uses an array of integers specifying rooms, rather than walls. Each of the representation types uses the same variation operators during the evolution process, uniform crossover, which switches values/cells, and uniform mutation, which regenerates a value/cell. The five fitness functions presented in the article controlled the game levels attributes such as lengthening the path from start to finish, increasing or decreasing the number of loops and culs-de-sac and increasing or decreasing the number of branches along the paths.

Another method for producing maze-like levels was presented in Johnson et al (2010) which used cellular automata to produce infinite game levels that were cave-like in appearance. This algorithm randomly initializes a grid of cells to one of two states, either rock state or floor state. Each cell in the grid is then iterated over a set number of times altering the state of the cells based off its neighbouring cells. If enough neighbouring cells are rock cells then the cell's state is set to rock, otherwise it is set to floor. In this particular implementation, each grid of cells was considered a chunk and new chunks were generated as needed during gameplay to produce an infinite level. If two chunks did not connect via traversable areas automatically, a tunnel would be generated between the two chunks. The levels formed by this algorithm resemble sparse mazes as not every traversable area is reachable.

The research detailed in this document is closely related to Ashlock's and Johnson's, as it aims to produce level layouts, similar to those generated in Ashlock's approach, using cellular automata, as was used in Johnsons approach. Although there are existing PCG techniques that use GA to evolve level layouts, an approach using GA to evolve CA rule sets capable of generating game level layouts has not yet been explored.

2.2 Technical Review

This section gives comprehensive descriptions of concepts that were briefly described in the literature review and were relevant to this research. It is divided into four sub-sections, the first detailing the structure of search-based PCG followed by genetic algorithms, then cellular automata, and finally the recursive backtracker maze generation algorithm.

2.2.1 Search-Based Procedural Content Generation

Procedural content generation can produce many types of video game content which usually has to meet specific requirements. A few techniques have been developed that address the challenge of generating content to meet specified criteria. One of the most common of these techniques is search-based PCG (Togelius et al., 2011). This section describes the process of search-based PCG and the role of the fitness function in this method.

Search-based PCG methods differ slightly depending on the form of search/optimization technique employed. This section assumes the use of a genetic algorithm as the search technique, as was using in this research, and therefore this process starts by generating a population of content candidates. The next phase of the process assigns each candidate a fitness value by testing it with a fitness function to determine how well it meets the specified criteria. If the highest ranking candidate achieves an acceptable fitness value then the algorithm terminates. Otherwise an iterative process of selection and variation is performed until an optimal solution is found. Figure 2 represents this process.

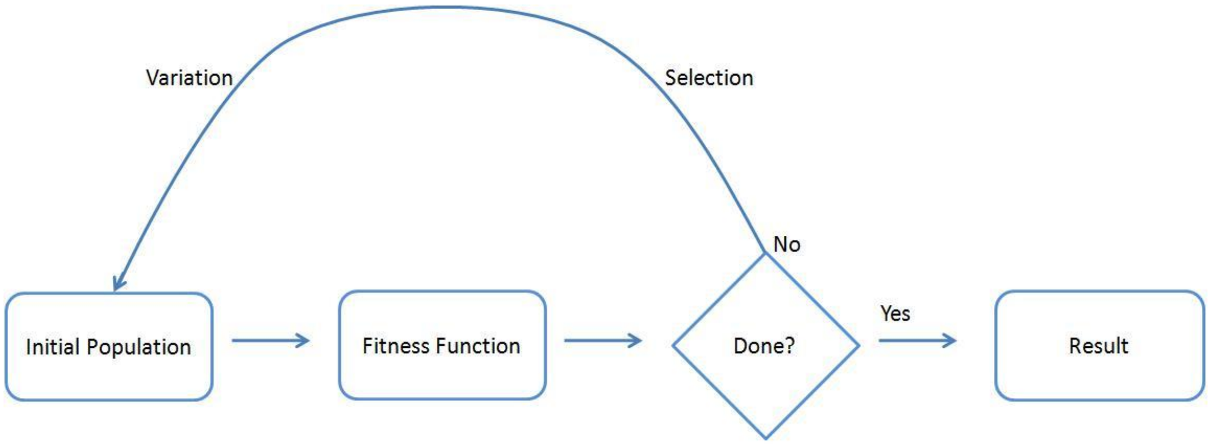


Figure 2. Diagram of the Search-Based Procedural Content Generation Process

The fitness function in search-based PCG is what specifies the requirements of the content that is to be generated. By altering the fitness function, content with different attributes can be created. An example of how a fitness function governs generated content can be seen Figure 3 where each mazes fitness is determined by the length of the path from start to finish. In this case, the search-based PCG algorithm will attempt to upgrade the maze to produce longer paths from start to finish.

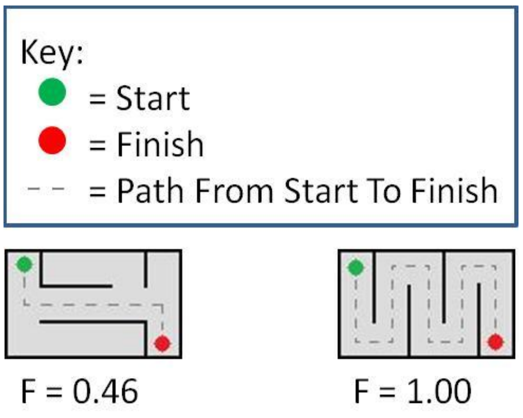


Figure 3. Illustration of a fitness function grading mazes based on the length of the path from start to finish

2.2.2 Genetic Algorithms

One search metaheuristic that is commonly used in search-based PCG is genetic algorithms. These are population-based metaheuristic optimisation algorithms that attempt to find optimal solutions to ill-defined problems. This section will describe the terms commonly used in GA, chromosome, selection, mutation, and crossover, followed by a description of the process.

Chromosomes, also known as genotypes, are data representations of possible solutions and should not be confused with the solution itself, commonly referred to as the phenotype. Each chromosome must be mapped to the solution it represents. There are two types of mapping process, direct and indirect. Genotypes used in direct mapping are linearly proportional in size to its phenotype, while genotypes used in indirect mapping are not. Figure 4 shows an example of both a direct and indirect representation of a maze. In the direct representation each cell of the maze is represented by either a 1 or a 0. This representation's size is linearly proportional to the size of the maze. The indirect representation uses integers to represent walls in the maze. Therefore the indirect representations size is not dependant on the size of the maze.

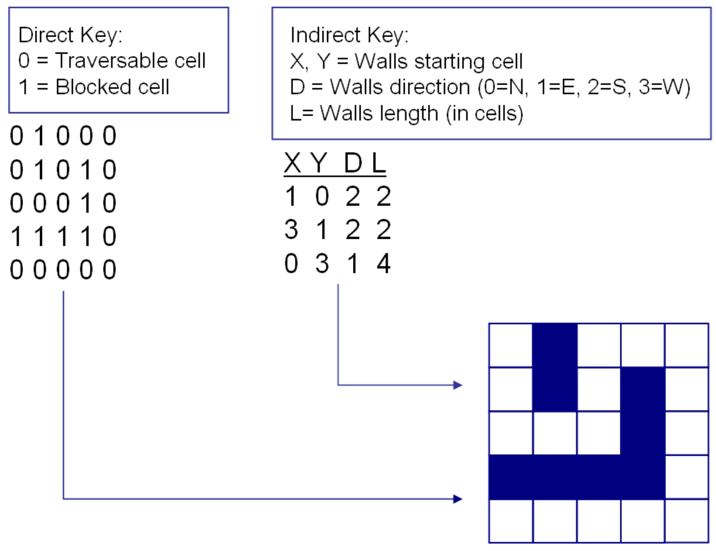


Figure 4. A direct and indirect genotype for a maze

Selection is the process of selecting the chromosomes with the highest fitness values on which to perform mutation and crossover. Genetic algorithms may employ an elitist scheme where a percentage of the fittest chromosomes are kept unchanged each generation. Elitist schemes are used to avoid a loss of good solutions after the evolution process. The loss of good solutions may be caused by setting parameters, such as the mutation rate, to unsuitable values.

Mutation is the process of altering some genes of a chromosome. This is typically performed by selecting genes of a chromosome and replacing them with new values. The mutation rate determines the probability that a gene in a chromosome will be altered and therefore should be tweaked to maximise the performance of the algorithm.

Crossover is the process of exchanging genes between two chromosomes. Typically the crossover rate determines the probability that a chromosome will undergo this process. This process involves splitting a chromosome into two or more sections and swapping the cut-out sections with those from another chromosome. The crossover rate should also be tweaked to maximise performance.

Genetic algorithms are iterative processes that initially generate a population of chromosomes and repeatedly perform selection, mutation and crossover on them until a desired solution is found. The exact steps are listed below and an illustration of the process is displayed in Figure 5.

1. Generates initial population of chromosomes.
2. Checks if the termination condition is met. The termination condition can be either that an optimal solution was found or that this step has been repeated a maximum number of times. If a condition has been met then the algorithm terminates.
3. Map genotypes to their phenotypes and assign them fitness values using the fitness function.
4. Select the group of chromosomes with the highest fitness values.
5. Perform mutation and crossover operations on the chromosomes to create a new population of chromosomes.
6. Repeat from step 2.

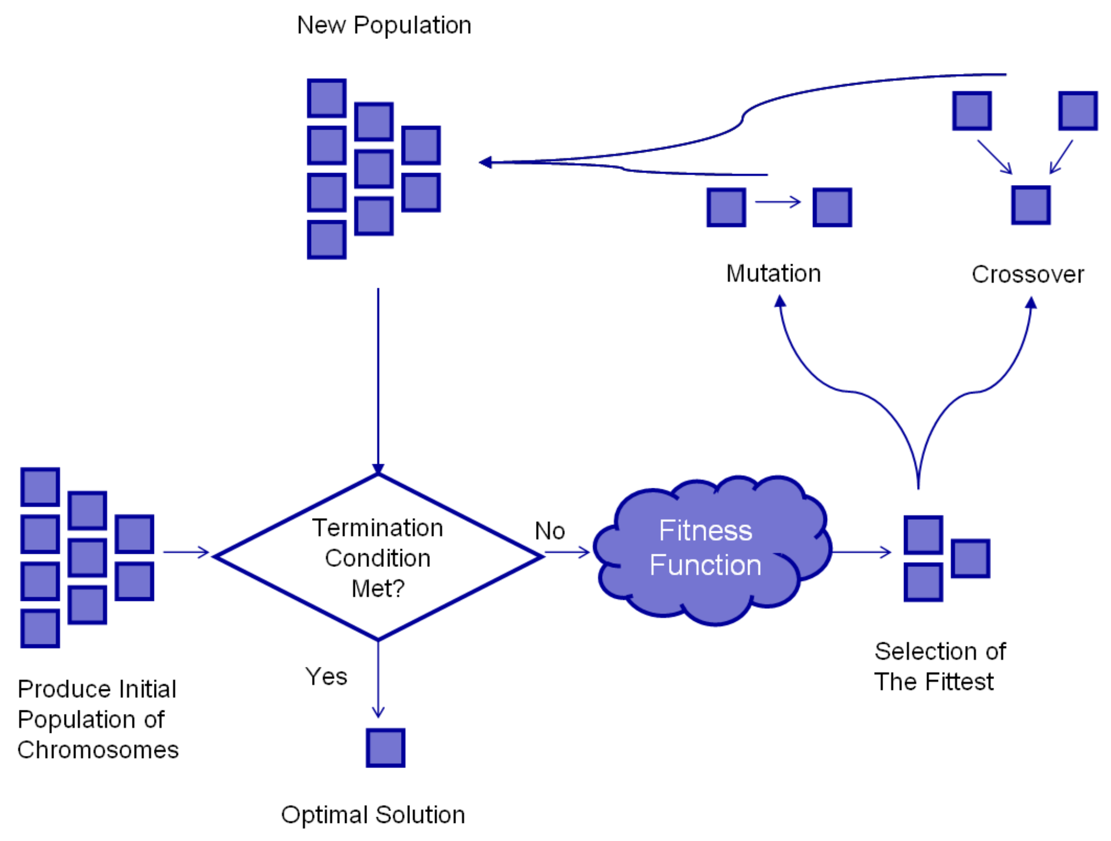


Figure 5. A diagram of the genetic algorithm process

2.2.3 Cellular Automata

Like genetic algorithms, cellular automata are also iterative processes, although these processes are relatively simple and easy to define. There are two main components of a cellular automaton, the grid of cells on which the process is performed, and the set of rules that is applied to each cell in the grid. This section defines these two components and the CA process.

The grid of cells used in CA goes through a metamorphosis of states as the state of each cell within it changes over time. There are three key attributes associated with the grid that need to be considered during its design, size, dimension and number of cell states. The grid can be of any size and dimension and the cells within it can be set to any of an infinite number of states. However, due to computational limitations it is best to find optimal settings for each of these attributes. The combination of the grids size, dimension and the state of its cells form a configuration. The initial configuration is the state of the grid before the CA process starts. This state constantly changes through this process until the process terminates. The resulting state of the grid is referred to as the final configuration which should ultimately be the desired result.

The rule set is applied to each cell in the grid synchronously during the CA process and alters the state of each cell based on its current state and the state of the cells in its neighbourhood. An example of a rule set is the majority rule, where a cells state is set to state shared by the majority of its neighbours. The neighbourhood is a key factor in this process and can greatly alter the effect of a cellular automaton. There are two commonly used neighbourhood types in cellular automata, the Moore neighbourhood and the Von Neumann neighbourhood. The Von Neumann neighbourhood is diamond shaped and with a radius of one consists of the centre cell and its four orthogonal cells, commonly referred to as 4-connected. The Moore neighbourhood is square shaped and with a radius of one consists of the centre cell and its four orthogonal and four diagonal cells, commonly referred to as 8-connected. Figure 6 displays both these neighbourhood types with two different radii.

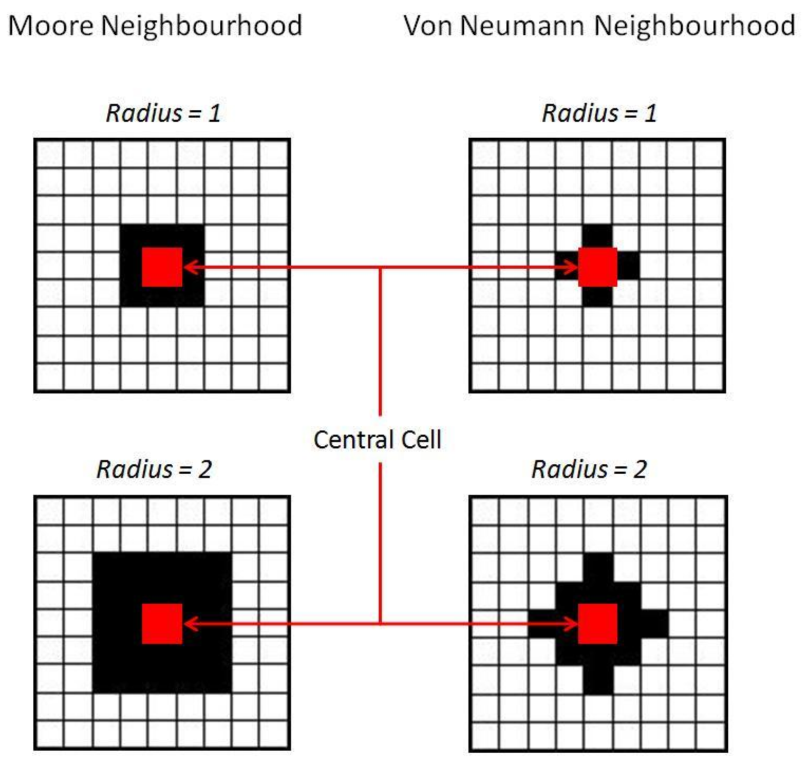


Figure 6. Illustrations of the Moore and Von Neumann neighbourhoods

Once the initial configuration has been formed and the rule set decided upon, the cellular automata process can begin. The cellular automaton applies the rule set to each cell in the initial configuration synchronously, altering the grids state. This process is applied iteratively on the grid for a set number of cycles. Once the cellular automaton terminates, the final configuration is achieved. Figure 7 shows this process.

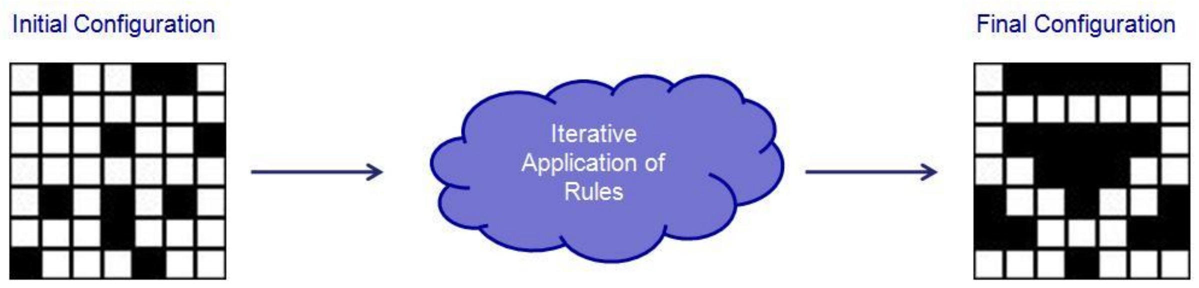


Figure 7. Diagram of the cellular automata process

2.2.4 Recursive Backtracker Maze Generation Algorithm

Like cellular automata, another algorithm that runs on a grid of cells is the recursive backtracking maze generation algorithm (Buck, 2011) which is one of the many algorithms that generate perfect mazes. This maze generation algorithm was used in this research to produce initial configurations for cellular automata. Although any maze generation technique would be useful for this purpose, the recursive backtracker is among the simplest to implement.

The recursive backtracker is an iterative, stack-based algorithm which applies a process to a stack of cells. The stack of cells starts off empty and the process is listed below.

1. Select random cell in grid and add it to the stack *S*.
2. Checks if *S* is empty. If *S* is empty then terminate the algorithm.
3. Select the last cell *C* from *S*.
4. Select one of *C's* neighbouring cells *N* that has not been carved into yet and carve a passage from *C* to *N*. The Von Neumann neighbourhood is used in this algorithm with a radius of 1.
5. Add *N* to *S*.
6. If *C* has no more unvisited neighbours, remove it from *S*.
7. Repeat from step 2.

2.3 Summary

PCG is a family of techniques that have been used in the video game industry to produce game content, including level layouts. Two common types of PCG include constructive PCG and generate & test PCG. Search-based PCG is an enhanced form of generate & test that uses search/optimisation techniques, such as genetic algorithms, to improve performance.

Search-based PCG has been paired with CA to generate simple game level layouts. CA may be capable of producing more intricate layouts but discovering CA rule sets capable of this task can be difficult. Previously, genetic algorithms have been used to find CA rule sets capable of performing particular tasks, such as density classification and pattern reconstruction.

The research that will be detailed in the following chapters developed a technique that attempts to find CA rules that produce level layouts with maze-like properties. This technique was produced using a combination of GA, CA and maze generation techniques. Although related research has been performed in this area (Ashlock et al., 2011; Johnson et al. 2010; Mitchell et al., 1996), the combination of GA, CA, and maze generation is a unique approach to level layout generation.

Chapter 3. Proposed Approach

This research developed a method of generating game level layouts with maze-like properties by using a genetic algorithm and cellular automata. The investigation and development in this project used an engineering methodology (Basili, 1993) which included four phases, study existing solutions, propose a new solution, develop and refine the proposed solution, and test and evaluate the solution.

The proposed approach, as shown in Figure 8, consisted of two phases. The first phase involved generating a collection of perfect mazes to be used in the evaluation step of the second phase, which is the GA process. These phases incorporate techniques which include a modified graph traversal algorithm for maze generation, a unique method of maze attribute extraction using a series of image processing techniques, and a genetic algorithm used to evolve cellular automata rules. The modified graph traversal algorithm generates mazes to be used as input for cellular automata and is described in section 3.1 and is followed by section 3.2 which details the GA process outlined in Figure 8.

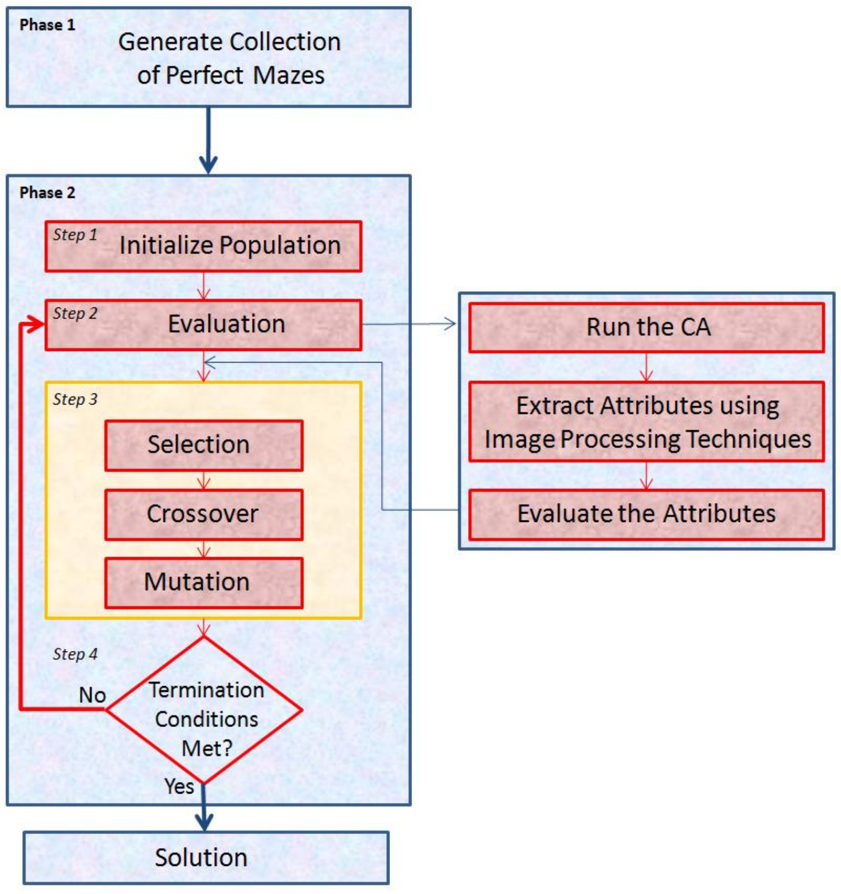


Figure 8. Visual representation of the proposed approach.

3.1 Phase 1: Generate a Collection of Perfect Mazes

Before the GA process could begin, this approach generated a collection of 100 perfect mazes which were used as initial configurations and fed as input to CA in the GAs fitness function, as described in section 3.2.2. The figure 100 was selected for the collections size as Mitchell et al.'s approach (1996) used a collection of 100 initial configurations in their evaluation technique. The collection of mazes was generated using a modified version of the recursive backtracker algorithm (Buck, 2011), a stack-based graph traversal algorithm that is commonly used to generate, and solve, perfect mazes. The implementation used in this approach differs slightly from the standard algorithm, which is described in section 2.2.4. The modified algorithm was used to produce a maze where walls were defined by blocking adjacent cells, rather than defining walls between adjacent cells. This was important as the CA configurations needed to consist of both traversable and non-traversable cells and did not support walls to be defined between cells. The pseudo-code of the modified algorithm is shown in Table 2.

```
Create a grid of cells, each cell can be set to either blocked or unblocked and can be flagged as either visited or unvisited.
For each cell in the grid.
    Set cell to blocked.
    Flag cell as unvisited.
End loop.
Create empty stack of cells.
Select a random cell from the grid and push it onto stack.
// Make sure that the selected cells X and Y coordinates are both a multiple of 2.
While stack is not empty.
    Pop the last cell from the stack.
    Set the cell as unblocked.
    Flag the cell as visited.
    Search for an unvisited neighbouring cell in a random direction, up, down, left, or right.
    // Make sure that the neighbouring cell is 2 cells away from the current cell.
    If an unvisited neighbouring cell was found.
        Push neighbouring cell onto the stack.
        Set the cell that's between the current cell and the neighbouring cell to unblocked.
    End if.
End loop.
```

Table 2. Pseudo-code for the modified recursive backtracker algorithm.

Figure 9 displays a maze generated using the standard recursive backtracker algorithm and another using the modified version. As can be seen, the left image which used the standard recursive backtracker algorithm is a maze where all the cells are traversable and are blocked from one another by walls that run between the cells. The right image is a maze generated using the modified version of the algorithm and forms walls by blocking cells, making them non-traversable.

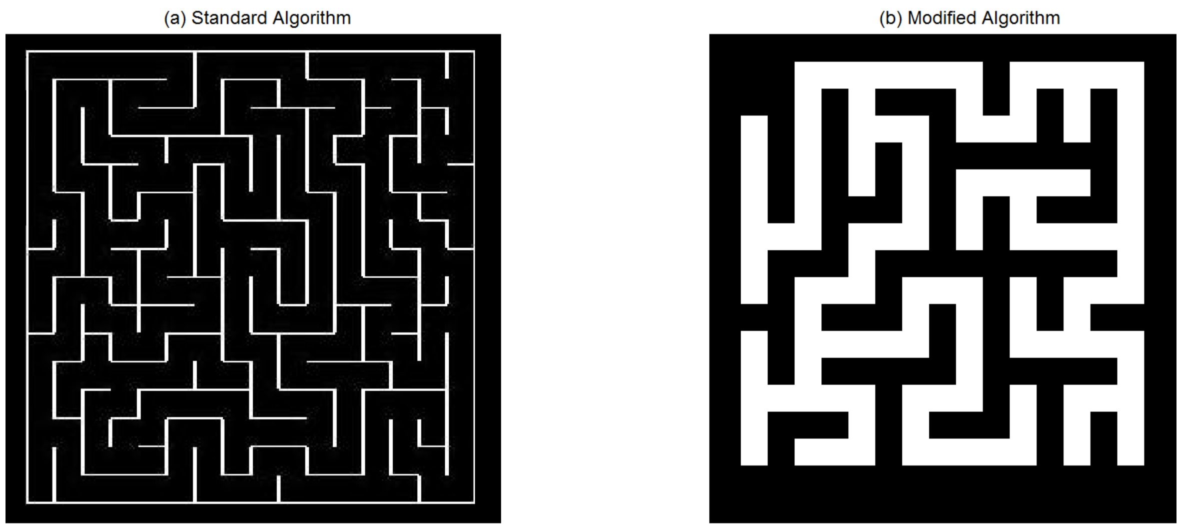


Figure 9. (a): Example of a maze generated using the standard recursive backtracker algorithm. (b): Example of a maze generated using the modified algorithm. Note that the black areas in the left image are traversable while the white areas in the right image are traversable.

3.2 Phase 2: The GA Process

Once the collection of perfect mazes had been generated, the GA process commenced, but before this process is explained, some issues that needed to be considered are outlined. These considerations include how the CA rules were represented, and how they were evaluated. These are explained in sections 3.2.1 and 3.2.2 followed by the steps of the GA process in section 3.2.3.

3.2.1 Chromosome Representation

The chromosomes, in this approach, represented CA rule tables, which are lookup tables that take a neighbourhood configuration as input and returns an associated output state. The output state is the new state of the central cell of the input neighbourhood. CA rule tables are traditionally represented by storing an output state for every possible CA neighbourhood configuration. This is suitable when using simple CA that consists of a small neighbourhood and only two cell states. However, when using more complex CA, the range of possible neighbourhood configurations quickly expands, making the traditional approach infeasible. A CA's complexity is affected by two attributes, S, the number of states each cell in the CA can be set to, and N, the size of the neighbourhood. The number of neighbourhood configurations is thus S^N. For example, a simple cellular automaton that uses a 3x3 neighbourhood grid, with each of the neighbourhoods nine cells being set to one of two states, the number of neighbourhood configurations equals 2^9, or 512. This is not a particularly large value and it is easy to accommodate a population of chromosomes that contain 512 genes, but when using a 5x5 neighbourhood grid, the number of neighbourhood configurations is equal to 2^{25}, or 33,554,432. This becomes a lot less feasible, especially when adding an additional cell state where S=3 and N=25 the number of neighbourhood configurations increases to 847,288,609,443. This is a large number of output states to accommodate, with space requirements of approximately 200 gigabytes per chromosome. This meant that to explore the effects of using more complex CA, a more space efficient method of chromosome representation would need to be utilized.

During this study, two chromosome representations were explored, both with different chromosome sizes, search space sizes. The two representations involved in this study were labelled as representation 1, or the direct representation, and representation 2, or the indirect representation. The following sub-sections will describe these representations that were explored in detail.

Representation 1 (Direct)

The first representation used the traditional method of storing an output state for every possible neighbourhood configuration in a list. Each output state was stored in lexicographic order of its corresponding neighbourhood state for lookup purposes. Each output state was stored as a two bit integer which allowed up to four possible output state values to be used. This representation was not used with neighbourhood sizes greater than nine due to prohibitive chromosome sizes, but allowed for varying ranges of output state values while still exploring the entire search space. Figure 10 demonstrates an example chromosome representing a small rule table containing 4 output states. [] demonstrates how a rule table index is calculated from a neighbourhood configuration using a simple example rule table.

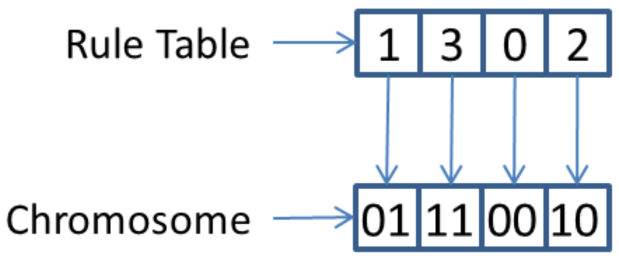


Figure 10. Illustration demonstrating encoding of rule tables using chromosome the direct representation.

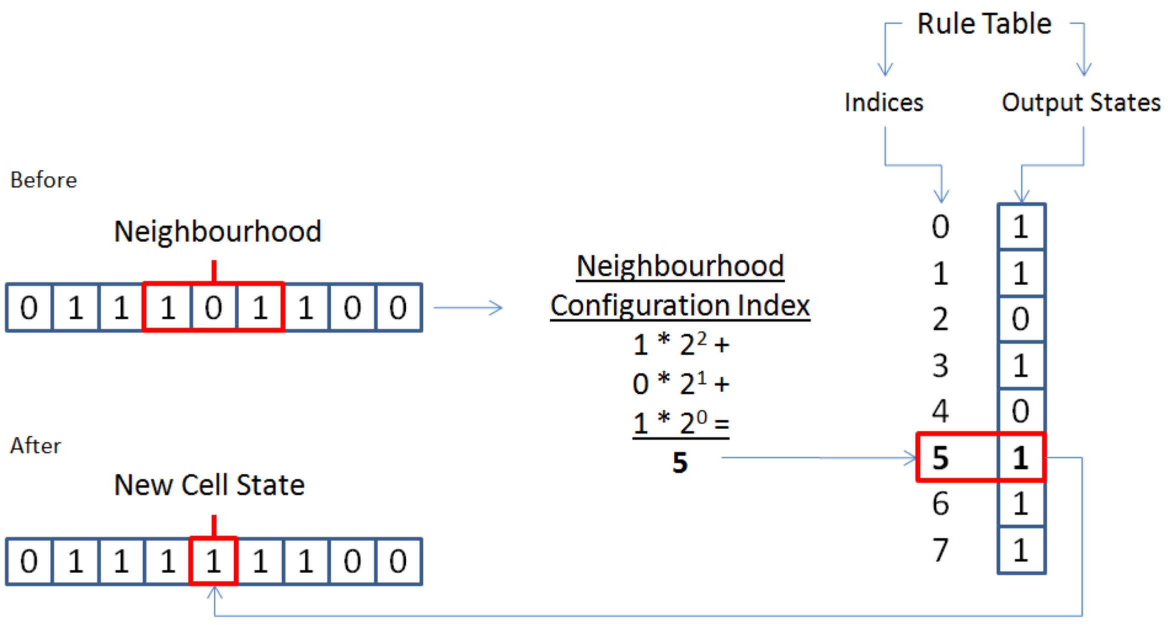


Figure 11. A diagram displaying the process of applying a direct rule table to a single cell in a simple 1D CA. The cells neighbourhood is selected and the values of its neighbours are used to create an index. The index is used to access the rule table and the cells state is changed to the output state at that index.

The flip-bit mutation operator is not a suitable mutation operator for this representation. This was because if the range of output state values being used was less than 4, flip-bit mutation could result in invalid output state values. Therefore the uniform mutation operator is used for this representation. In uniform mutation, a mutated gene was set to a random value in the range of [0, *S*-1] where *S* is the range of output state values. The crossover operator used for this representation is the single-point crossover operator.

Representation 2 (Indirect)

This representation was more space efficient than representation 1, as it used the sum of neighbourhood values to index an output state, rather than storing one output state for each unique neighbourhood configuration. This reduces the chromosome sizes significantly as the range of summed neighbourhood values is equal to (S - 1) * N + 1. Due to the greatly reduced chromosome sizes, more complex CA could be explored which allowed the use of larger neighbourhood sizes. Figure 12 demonstrates how a rule table index is calculated from a neighbourhood configuration using a simple example rule table.

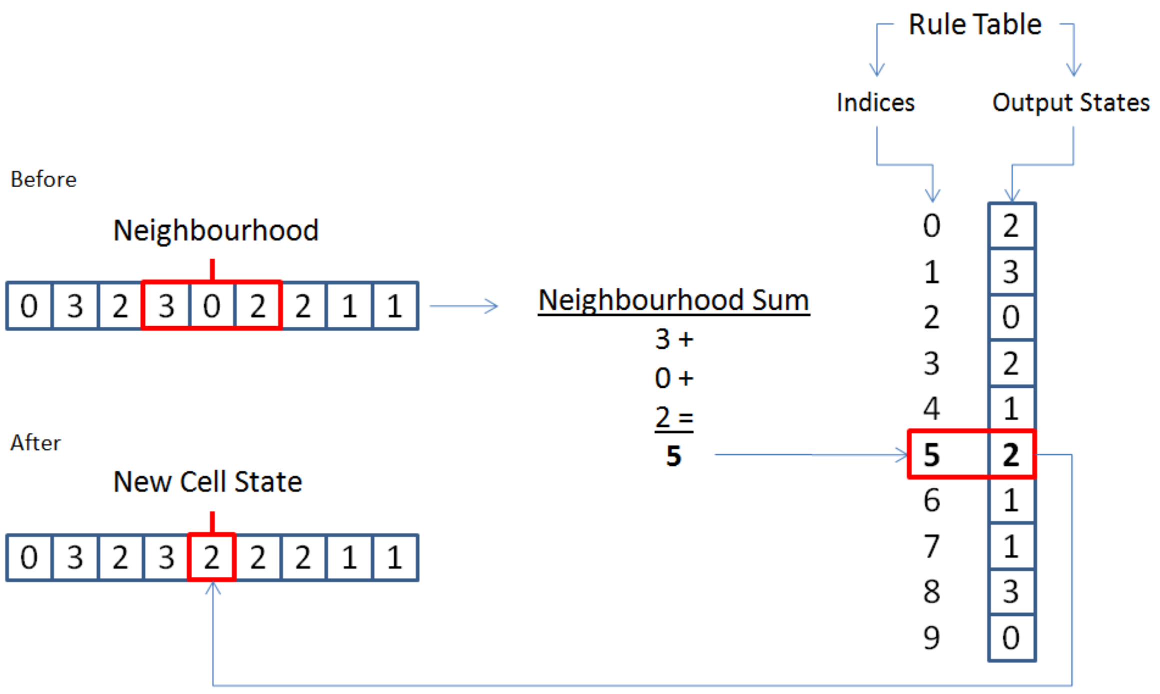


Figure 12. A diagram displaying the process of applying an indirect rule table to a single cell in a simple 1D CA. The cells neighbourhood is selected and the values of its neighbours are summed. The sum is used as an index into the rule table. The cells state is changed to the output state at that index.

The flip-bit mutation operator is also not a suitable mutation operator for this representation. This was because if the range of output state values being used was less than 4, flip-bit mutation could result in invalid output state values. Therefore the uniform mutation operator is used for this representation. In uniform mutation, as in representation 1, a mutated gene was set to a random value in the range of [0, *S*-1] where *S* is the range of output state values. The crossover operator used for this representation is the single-point crossover operator.

3.2.2 Chromosome Evaluation

Once the chromosome representation had been considered and decided, the issue of how to evaluate the chromosomes remained. Previously, CA rule tables have been evaluated by being run on a collection of initial configurations, and assigned a fitness value based on the percentage of final configurations that were in the correct state (Mitchell et al., 1996). This appeared to be a valid approach for this research, but how to evaluate the final configurations was still an issue that had to be considered.

Since the final configurations in this approach aimed to form game level layouts with desired maze-like properties, a simple weighted aggregation of attributes was used. This required the desired attributes to be extracted from the level layouts before they could be used in the evaluation process. Due to the lack of existing techniques in the literature, a unique approach was developed in this study to extract these attributes from the level layouts using a collection of image processing techniques.

The evaluation process is divided into three steps. Step 1 describes the process of applying the CA rule tables on the collection of perfect mazes that were generated in phase 1 of this approach. Step 2 included descriptions of the level layout attributes that were used and details how they were extracted. Lastly, step 3 explains how the fitness values were calculated using the extracted attributes.

Step 1: Run the CA

During the evaluation of a rule table, it was applied to the entire collection of perfect mazes that were generated in phase 1 of this approach. This process closely follows the standard CA process described in section 2.2.3, where CA rules are applied to an initial configuration, in this case, a perfect maze, for a set number of iterations. The number of iterations that the CA rules are applied in the fitness function can be varied.

The one difference between the standard CA process and the one used in this study was the use of flavours. The idea of flavours is that a single cell state can be divided into a number of sub states for the purpose of rule application. This study used two cell states, the traversable state (state 0) and the non-traversable state (state 1). The traversable state could be divided into two sub states (states 1 and 2) or three sub states (states 1, 2, and 3) which, including the non-traversable state made a total of either three or four cell states for rule application. Once a final configuration was produced, every cell in the configuration was set to their original state of either traversable or non-traversable. Therefore cells in state 0 would remain in state 0 and cells in a state of 1 or greater would be set to state 1. The reason flavours were used in this study was to explore the effects of more complex CA, as additional cell states could result in more complex behaviour from the CA and possibly produce better layouts. Figure 13 displays an example of a final configuration before and after the application of flavours.

Key:

Black Areas:	Cells in state 0
White Areas:	Cells in state 1
Red Areas:	Cells in state 2

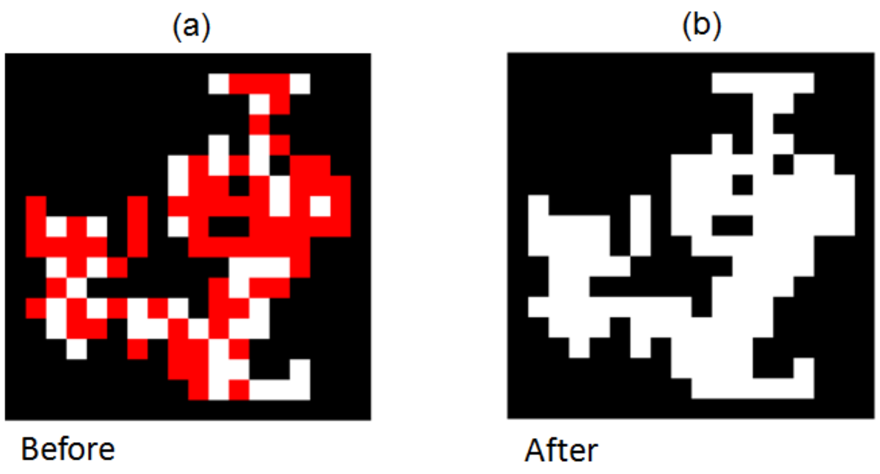


Figure 13. (a): Example of CA final configuration before application of flavours. (b): Example of CA final configuration after application of flavours.

Step 2: Extract Attributes Using Image Processing Techniques

Once the CA rules have been applied to the pre-generated mazes, attributes of these resulting layouts are extracted for evaluation. Nine attributes were extracted for evaluation from the final configurations. Each attribute was assigned a desired value, used to determine what attribute values the final configurations should have, and a weighting factor that determined how important that attribute was. Not all nine attributes have to be selected to be used in a fitness function; the attributes that were not evaluated were given a weighting factor of 0. The maze-like attributes that were explored in this research are listed and defined below followed by details on the image processing techniques that were used to extract them.

- Number of disconnected traversable areas.
- Average size of traversable areas.
- Size of largest traversable area.
- Number of passageways.

- Average length of passageways.
- Number of rooms.
- Average size of rooms.
- Number of dead-ends.
- Number of culs-de-sac.

A **traversable area** is defined as a complete collection of cells in the traversable state that are connected to one another through other cells in the traversable state. An example is shown in Figure 14. As can be seen in Figure 14, layout (a) is a game level layout that is colour coded where rooms are green, passageways are red, junctions are blue, and dead-ends are white. Layout (b) is the same level layout as (a) but with each traversable area given a unique colour, making it easy to see that layout (b) contains four traversable areas. The **size** of each traversable area is defined by the number of traversable cells contained within it.

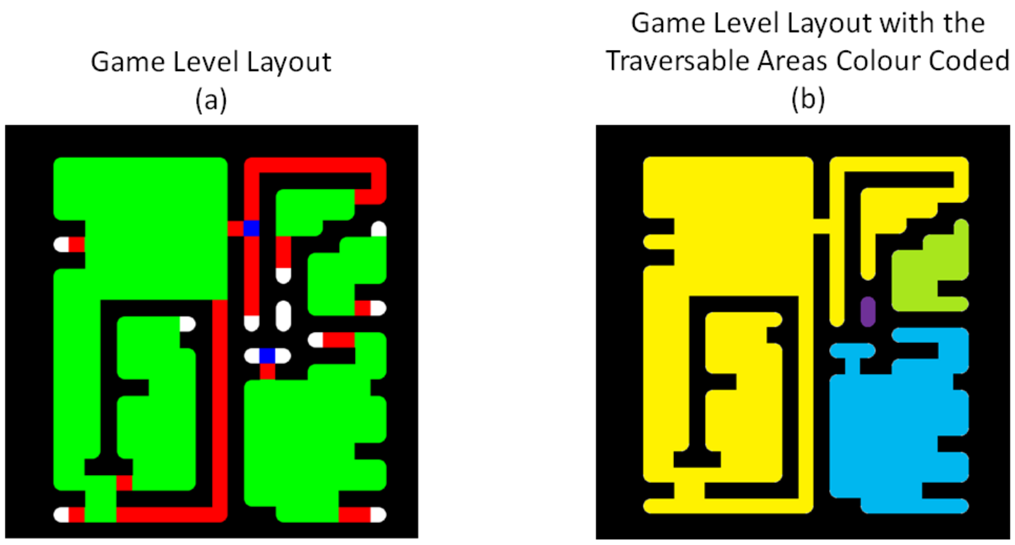


Figure 14. Illustration demonstrating the definition of a traversable area.

A **passageway** is defined as a collection of traversable cells that can only be entered from two directions (using 4-connectivity). T-junctions and X-junctions can also belong to a passageway but are handled as a special case as they can only belong to 1 passageway, not multiple. The **length** of a passageway is defined as the number of traversable cells that it consists of.

Rooms are more implicitly defined. They are defined as any traversable cell that is not a passageway, junction or dead-end. A room's **size** is defined as the number of traversable cells that it consists of.

Dead-ends are defined as traversable cells that are blocked from three or more directions (using 4-connectivity). Dead-ends are usually expected to appear at the end of passageways although they can also appear as nooks inside rooms or as a detached traversable area consisting of a single cell. Although dead-ends can appear at the end of passageways, they are not considered as part of passageways.

Culs-de-sac are defined as rooms that only have one passageway leading in or out of them. Culs-de-sac are similar to dead-ends as neither of them lead anywhere, except dead-ends are not required to be attached to a passageway and can only consist of one cell.

To extract these attributes from the level layouts, a series of image processing techniques including erosion, outlining, and a form of region growing, was used. These algorithms are detailed below followed by a description of how they were used to extract each attribute.

The Image Processing Techniques

The **erosion** method (Russ, 2006) compares a series of structuring elements to the neighbourhood of every cell within a grid. If the neighbourhood is equal to any of the structuring elements then the cell is eroded. Eroded cells are set black. The structuring elements and neighbourhoods are both 3x3 grids that are encoded into binary strings. The binary strings use nine bits with each bit representing a single cell in the 3x3 grid, with 1's being black cells and 0's being white cells. The bits in the binary strings are ordered from the top-left cell of the 3x3 grid moving down and across to the bottom-right cell, from least significant bit to most significant bit. An example can be seen in Figure 15.

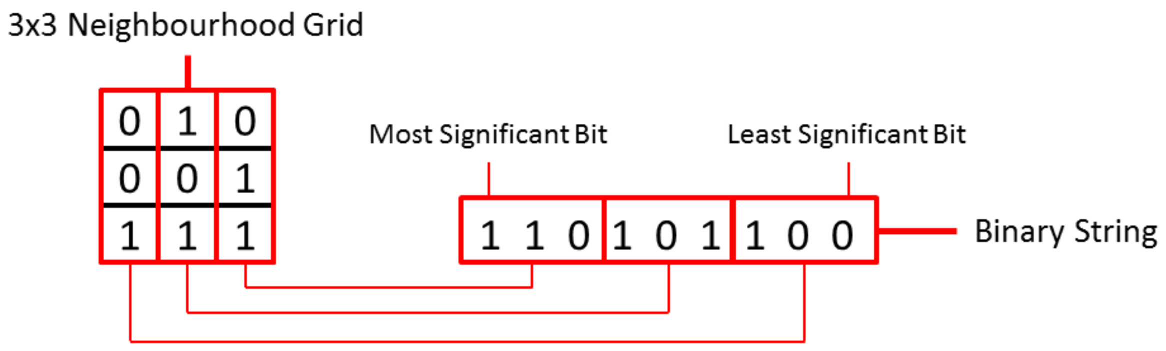
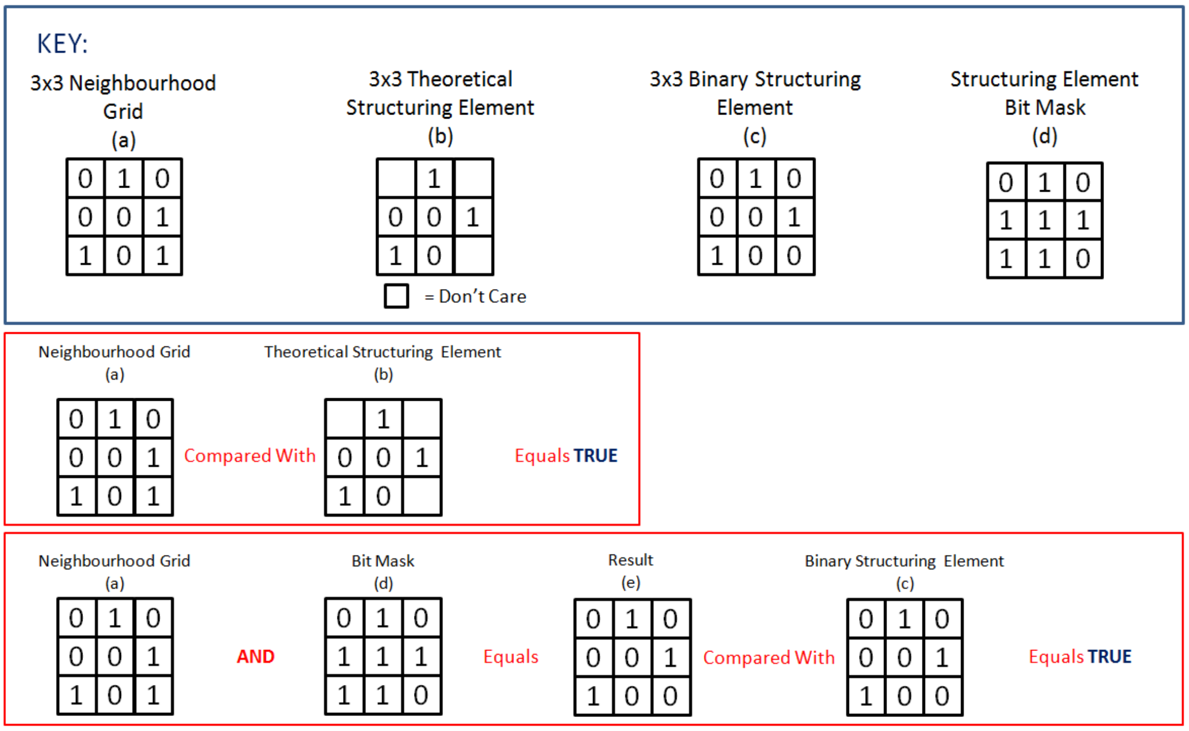


Figure 15. Illustration displaying how a 3x3 neighbourhood grid is encoded into a binary string.

Theoretically, the 3x3 structuring elements also allow cells in the element to be set to a 'don't care' state, where the corresponding cells in a 3x3 neighbourhood can be either 1 or 0 and not impact the comparison. To implement this functionality each structuring element was paired with an additional binary string which is used as a bit mask. The bit mask was used to specify which cells in a 3x3 neighbourhood the corresponding structuring element is interested in. The binary AND operator was used with the bit mask and the neighbourhood binary string to set unimportant bits in the neighbourhood to 0. Then a standard binary comparison was performed between the structuring element and the neighbourhood. An example can be seen in Figure 16.

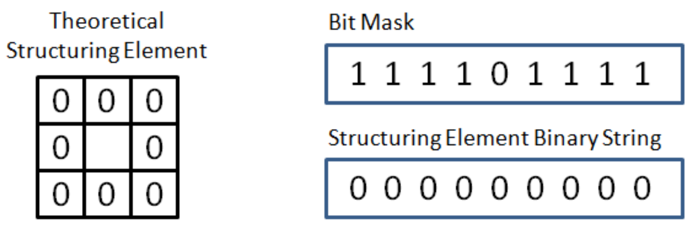
As can be seen in Figure 16, when grid (a) is compared with the theoretical structuring element (b), it results in a positive identification, meaning (a) is equal to (b). This is because each cell in (a) is set to the same value as its corresponding cell in (b), with the empty cells in (b) allowing (a) to contain either a 0 or a 1 without affecting the comparison. To implement this functionality, the binary AND operator is applied to grid (a) and the bit mask (d) which sets the unimportant cells, denoted as blank cells in the theoretical structuring element, to 0. The result of this operator is (e). The result (e) is then compared to the binary structuring element (c). This results in a positive identification as the value of each cell in (a) is equal to the value of its corresponding cell in (c).



KEY:
3x3 Neighbourhood Grid (a)
3x3 Theoretical Structuring Element (b)
3x3 Binary Structuring Element (c)
Structuring Element Bit Mask (d)
= Don't Care
Neighbourhood Grid (a)
Theoretical Structuring Element (b)
Compared With
Equals TRUE
Bit Mask (d)
Result (e)
Binary Structuring Element (c)
AND
Equals

Figure 16. An illustrated example of how bit masks are used in conjunction with structuring elements.

The **outlining** method simply set all cells to black that were not 'edge' cells. Edge cells were defined as any cell that had one or more black cells in its neighbourhood. This made erosion a perfect candidate for this task. This method was performed by using erosion with a single structuring element that eroded all cells that had no black cells in their neighbourhood. The structuring element that was used is shown in Figure 17.



Theoretical Structuring Element
Bit Mask
1 1 1 1 0 1 1 1 1
Structuring Element Binary String
0 0 0 0 0 0 0 0 0

Figure 17. The structuring element used in the outlining image processing technique.

The **region growing** technique in this approach was used to find the number of white regions in a grid, the average number of white cells contained within them, and the number of white cells contained within the largest region. This method worked by iterating over each cell in the grid. Whenever a white cell was found an expansion process would be performed. This process involved checking each of the cells neighbours and if any of its neighbours were also white it would add them to a list of cells that made up the region. When the region could not expand any further, the expansion process would end and the region growing method would continue to iterate over each cell, making sure not to expand cells already belonging to a region.

Using the Techniques

Now that the required algorithms have been detailed, the extraction process of the layout attributes will be described. Extracting the **number of traversable areas**, the size of the **largest traversable area**, and the **average size of traversable areas** was a simple process of applying the region growing algorithm to the level layout.

To extract the **room** data, the erosion method was used on the layout with a collection of structuring elements that eroded all the passageways, junctions, and dead-ends, leaving only the rooms. The different structuring elements that were used to do this are displayed in Figure 18. Although this is not the entire collection, the omitted elements consisted of varying rotations of those displayed. The region growing method was then applied to the room layout to extract the number of rooms and their average size.

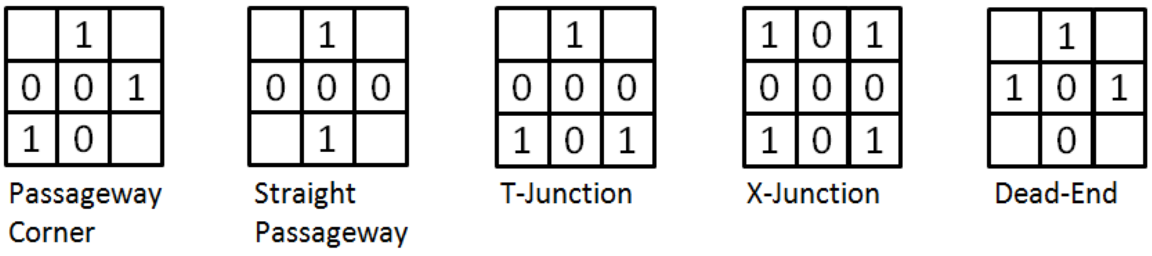


Figure 18. An illustration showing a single rotation of the structuring elements used to erode passageways, junctions and dead-ends.

Extracting the **passageway** data was a more complex procedure involving applying a series of erosion and region growing algorithms to the level layout. Below is a list of the steps that were taken to extract the passageway data.

- Apply erosion to initial layout to erode all passageways and junctions.
- Perform a binary XOR operation between the initial level layout and the eroded layout produced in step 1. This will produce a layout with only the passageways and junctions.
- Erode the layout produced in step 2 to produce a layout containing only the passageways.

- Erode the layout produced in step 2 to produce a layout containing only the junctions.
- Perform region growing on the layouts produced in both steps 3 and 4 to collect the number of junctions, *jc*, the number of passageways, *pc*, and the average size of the passageways, *ps*.
- Calculate the number of passageways, *pc'*, where *pc'* = *pc* - *jc*.
- Calculate the average size, or length, of passageways, *ps'*, where *ps'* = (*ps* + *jc*) / *pc'*.

The above steps results in the number and average length of passageways where each junction belongs to a single passageway. Figure 19 illustrates this process. Colour coded images were used so that the rooms, dead-ends, passageways, and junctions could be easily identified. These areas are colours green, white, red, and blue respectively.

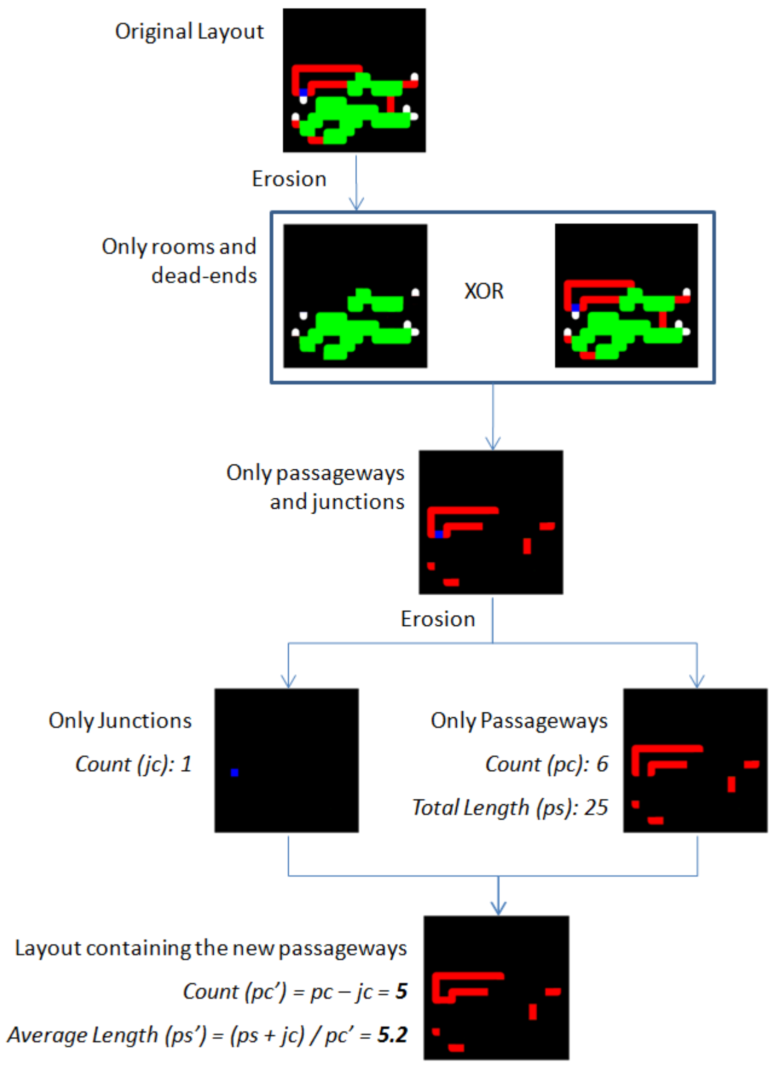

Original Layout
Erosion
Only rooms and dead-ends
XOR
Only passageways and junctions
Erosion
Only Junctions
Count (jc): 1
Only Passageways
Count (pc): 6
Total Length (ps): 25
Layout containing the new passageways
Count (pc') = pc – jc = 5
Average Length (ps') = (ps + jc) / pc' = 5.2

Figure 19. Illustrated example of how passageway attributes are extracted from a layout using image processing.

Determining the number of **dead-ends** was achieved by iterating through each cell in the layout and counting the number of white cells that were blocked from three or more directions by black cells. The number of **culs-de-sac** was calculated performing the outlining method to the room layout. The outlines were then separated into individual regions using region growing. The neighbours of every cell in each region, using 4-connectivity, were then compared with the original layout to determine if the neighbouring cell on the original layout was a passageway cell or not. The number of culs-de-sac was equal to the number of regions that had only one associated passageway cell.

Step 3: Evaluate the Attributes

Once all of the attributes had been extracted from the level layouts, they were each assigned a similarity measure, *sm*, using the formula

$$sm = (1.0 - |da - aa| / ma)^3 * aw \qquad \textit{Equation 1}$$

In *Equation 1* *da* and *aw* are the desired value of the attribute and its associated weighting factor. The value of the attribute extracted from the layout is denoted as *aa*, while *ma* is the max possible value that the attribute could be. The maximum values of the attributes are defined below. Note that *gw* and *gh* is the width and height of the level layouts measured in cells.

- Maximum traversable area count = $(gw * gh) / 2$
  This equals the maximum number of traversable areas and can be represented in the scenario of a checkerboard grid where half the cells are traversable and the other half are blocking the traversable cells from connecting to one another.
- Maximum traversable area size = $gw * gh$
  This equals the maximum size of a traversable area and can be represented in the scenario of a grid where every cell is traversable, creating one large traversable area.
- Maximum passageway count = $(gw * gh) / 2$
  This equals the maximum number of passageways and can be represented in the scenario where each second row of a grid is a passageway and each second column of a grid is a passageway. In this scenario, half the grids cells are passageways, a quarter are junctions and the other quarter are non-traversable.
- Maximum passageway length = $(gw * gh) / 2 + \max(gw, gh) / 2$
  This equals the maximum length of a passageway and can be represented in the

scenario of a grid containing a single passageway that travels from the bottom right corner of the grid to the bottom left corner. In this scenario, the passageway then moves up two cells and continues horizontally to the right side of the grid, then up two cells, and continues in this fashion until the top of the grid has been reached.

- Maximum room count = (*gw* * *gh*) / 4
  This equals the maximum number of rooms as each room must consist of a minimum of four traversable cells. This calculation is a simplification that does not take into account non-traversable cells that are needed to separate the rooms.
- Maximum room size = *gw* * *gh*
  This equals the maximum size of a room and can be represented in the scenario of a grid where every cell is traversable, creating one large room.
- Maximum dead-end count = (*gw* * *gh*) / 2
  This equals the maximum number of dead-ends and can be represented in the scenario of a checkerboard grid where half the cells are traversable and the other half are blocking the traversable cells from connecting to one another. In this scenario, each traversable cell is blocked in four directions and is therefore considered a dead-end.
- Maximum cul-de-sac count = (*gw* * *gh*) / 4
  Since a cul-de-sac has the same requirements as a room (except for the number of passageways that connect to it), it uses the same calculation as the maximum room count.

The maximum attribute values are scaling factors that ensure each attributes *sm*, before weighting is applied, is in the range of [0.0, 1.0]. Because the maximum attribute values can be quite large, the desired attribute values can be quite small in comparison. This makes the difference between the desired attribute value and the extracted attribute value, appear insignificant when it is not. This is reflected in the *sm* value, as high *sm* values can be given to extracted attributes that are not considered to be similar to the desired value. For example, using a grid size of 16, the maximum passageway count is 128. If the desired number of passageways is 7 and the extracted number of passageways is 14, the *sm* value, without weighting or the exponent, is 0.9453125. This is a high *sm* value although there is twice the desired number of passageways. Therefore an exponent was used as an additional scaling factor to reduce the *sm* value when the difference between the extracted attribute value and the desired attribute value is relatively small to the maximum attribute value.

Once a *sm* value had been assigned to each attribute, the *sm* values were summed together to produce an attribute similarity measure (ASM) for the layout. After the CA rules have been applied to each of the 100 maze configurations, and the resulting layouts have been assigned an ASM value, the ASM values are averaged to produce the fitness value assigned to the CA rule table.

3.2.3 The Steps of the GA Process

Once the issues associated with chromosome representation and fitness evaluations had been considered, the next consideration to be addressed is the GA parameter settings. Due to time restrictions, comprehensive parameter tuning was not possible and therefore De Jong's (1975) parameter settings were used. The mutation probability was the only parameter that did not follow De Jong's guidelines. De Jong suggests a mutation probability of 0.001, however this was assuming a chromosome size of 100 genes. Since the chromosome sizes in this study ranged between 10 and 262,144 genes, a mutation probability of one over the length of the chromosome was used instead (Bäck, 1993). Table 3 displays a complete list of GA parameter values that were used in this approach.

Parameter Name	Parameter Value
Population Size	50
Selection Type	Tournament (*tsize* = 5)
Crossover Operation	1-Point Crossover
Crossover Probability	0.6
Mutation Operation	Uniform Mutation
Mutation Probability	1/Length of Chromosome
Maximum Generation	5000

Table 3. Table of GA parameter values.

The rest of this section describes to four steps of the GA process. This starts with the initialization of the GA's population, followed by evaluation of the chromosomes, the repopulation process, and the terminating conditions of the GA.

Initializing the Population

The population of chromosomes used in the GA were initialized to random configurations where each gene in each chromosome was set to a value in the range of [0, S-1] where S equals the maximum number of cell states. For example, if the number of cell states being used was set to 4, then each gene in the chromosomes was set to a random value between 0 and 3. Once the chromosomes in the initial population had been initialized, each chromosome in the population would get evaluated.

Evaluating the Chromosomes

The fitness evaluation process was described in detail in section 3.2.2. Each chromosome represented a rule table which was run on each of the perfect mazes that were generated in phase 1 of this approach. This produced a number of final configurations for each rule table. Each final configuration was evaluated, using a weighted aggregation of attributes, and assigned an ASM value. The overall fitness value assigned to each CA rule table was the average of the ASM values given to all the final configurations it produced. Once the current population had been evaluated, chromosomes were selected from it to undergo mutation and crossover operators to form new chromosomes to produce the new population.

Generate New Population

The replacement scheme used in this GA started with an empty population and included an elitist selection scheme where the top five chromosomes from the current population were carried over to the empty population. After the elite chromosomes had been carried over to the new population, chromosomes were selected from the current population to have crossover and mutation operators applied to them. The new chromosomes, formed from these operators, were placed in the new population. This process was repeated until the new population had been filled. Below is a description of the selection scheme used in this approach, followed by the process of applying the crossover and mutation operators to produce the chromosomes for the new population.

Chromosomes were selected from the current population using tournament selection. The tournament selection mechanism used in this approach selected a tournament group of five random individuals from the current population and then selected the best chromosome from the group. The pseudo-code for this technique can be seen below.

```
Input:
        current_population
Output:
        selected_individual
Steps:
        Set tournament_group to an empty list of chromosomes.
        For each value in the range [1, tournament_size]
            Set random_individual to a random chromosome from current_population.
            While random_individual is in tournament_group
                Set random_individual to a random chromosome from current_population.
            Add random_individual to tournament_group.
        Set selected_individual to chromosome from tournament_group with highest fitness.
        Return selected_individual.
```

Table 4. Pseudo-code for the tournament selection operation.

Once a chromosome had been selected there was a chance that crossover would be applied to it. To determine if crossover should be applied, a random number was generated in the range of [0.0, 1.0]. If the value was less than or equal to the crossover probability then a second chromosome was selected from the current population and the crossover operator was applied to the two chromosomes. The crossover operator used was single-point crossover where the two selected chromosomes were split into two sections at a random gene and the sections were swapped between the two chromosomes to form two new chromosomes.

If crossover was applied, then the two new chromosomes were mutated using uniform mutation. If crossover was not applied, then this process was applied to the original chromosome. During this process, each gene within the chromosome would be subjected to mutation based on the mutation probability. To determine if a gene should be mutated, a random number was generated in the range of [0.0, 1.0]. If the generated value was less than or equal to the mutation probability, the gene would be mutated. The uniform mutation operator that was used would set the gene to a value in the range of [0, S-1] where S equals the maximum number of cell states used in the CA. The pseudo-code for this operator can be seen below.

```
Input:
        max_cell_state_count
        mutation_probability
        origingal_chromosome
Output:
        mutated_chromosome
Steps:
        Set mutated_chromosome to a copy of original_chromosome.
        For each current_gene in mutated_chromosome
            Set pm_chance to a random value in the range of [0.0, 1.0].
            If pm_change is less than or equal to mutation_probability then
                Set new_gene_value to a random value in the range of [0, max_cell_state_count -   1].
                While new_gene_value is equal to current_gene
                    Set new_gene_value to random value in the range of [0, max_cell_state_count - 1].
                Set current_gene to new_gene_value.
        Return mutated_chromosome.
```

Table 5. Pseudo-code for the uniform mutation operator.

After mutation, the resulting chromosome was added to the new population. The above steps were repeated until the new population had been filled. It then replaced the original population of chromosomes.

Step 4: The Termination Conditions

Once the new population was formed, the genetic algorithm checked its termination conditions. There were three terminating conditions, convergence, reaching a maximum fitness, and reaching a maximum number of generations. Convergence was achieved when the difference between the highest ranking chromosomes fitness value over the current generation and the 100 previous generations was less than a given threshold. The convergence threshold in this research was set to 0.0001 and if the fitness of the best chromosome did not improve by this amount over 100 generations then it was assumed convergence had been achieved. The maximum fitness condition would terminate the GA when the best chromosome had reached a particular fitness. In this research the threshold was set to 1.0 which was the highest fitness that a chromosome could reach, therefore no more improvements could be made. The GA would also terminate if neither of these conditions had been met by the maximum generation.

3.3 Summary

This chapter described the two phases of this approach. Phase 1 was detailed in section 3.1 and covered the generation of a collection of perfect mazes using a modified version of the recursive backtracker algorithm for use in the GA's fitness function.

Section 3.2 detailed the GA process starting with the issues that had to be considered, followed by the steps of the process. The issues included how to represent and evaluate the chromosomes. There were two representations that were explored, representation 1 which followed the traditional approach to representing CA rule tables, but could not be used with neighbourhood radii greater than 1 due to large storage requirements. And representation 2 which had substantially less storage requirements and could be used with neighbourhood radii greater than 1 but was limited in which rule tables it could represent.

The evaluation of chromosomes required running CA on the mazes generated in phase 1, extracting their attributes using a collection of image processing techniques, and using a weighted aggregation of the extracted attributes to formulate the fitness values of the chromosomes. The four steps of the GA included initializing the population of chromosomes, evaluating the chromosomes, replacing the population with modified chromosomes, and checking the GA's termination conditions. The next Chapter details the sets of experiments that were run using this approach and discusses the results and findings.

Chapter 4. Experiments, Results, and Findings

This chapter details all of the experiments that were performed during this research and discusses the results and findings. Each experiment was performed using five different fitness functions with the aim of finding CA rules that could generate different styles of level layouts. Section 4.1 details the fitness functions that were used in this research to guide the levels produced towards a particular style. Section 4.2 details the experiments that were performed to evaluate the influence of various GA and CA factors on the resulting levels, followed by section 4.3 presenting the results from the experiments.

4.1 The Fitness Functions

The fitness function chosen to evaluate levels produced by the CA rule sets governs the appearance of those levels. Based on desired level appearance, the fitness function can, for example, be chosen to favour certain sized rooms, lengths of passageways, and the number of dead-ends. During this research five different fitness functions, summarized in Table 6, were used, each to explore the possibilities of finding CA rules that could generate level layouts with different sets of attributes. These five fitness functions are labelled *f1*, *f2*, *f3*, *f4*, and *f5*. Each fitness function was defined by the desired values of the level attributes, detailed in Section 3.2.2, along with an importance rating. Each of the fitness functions (*f1* – *f5*) had non-zero weights for eight of the nine attributes (a zero weighting means that attribute is not considered). In this research the number of traversable areas was considered the most important attribute as a layout could meet every other criterion yet would not be useful as a maze-like game level if all of the passageways and rooms were not connected in a single traversable area. For this reason it was given the highest importance rating. The largest/average traversable size was considered the second most important attribute as the general size of the layout contributed greatly to its visual similarity with its goal layout. Therefore it was given the second highest importance rating, while all other attributes were given equal ratings.

	Fitness Functions									
	f1		*f2*		*f3*		*f4*		*f5*	
Grid Size:	16x16		16x16		28x28		24x24		18x18	
Attributes	Value	Weight	Value	Weight	Value	Weight	Value	Weight	Value	Weight
Accessible Area Count	1	0.27	1	0.27	1	0.27	4	0.27	2	0.27
Average Area Size	-	0.0	-	0.0	-	0.0	90	0.18	64.5	0.18
Largest Area Size	89	0.18	204	0.18	214	0.18	-	0.0	-	0.0
Passage Count	7	0.09	11	0.09	18	0.09	11	0.09	3	0.09
Average Passage Length	5.71	0.09	4.27	0.09	7.11	0.09	4.9	0.09	1	0.09
Room Count	4	0.09	7	0.09	9	0.09	5	0.09	5	0.09
Average Room Size	10.75	0.09	22	0.09	8.88	0.09	58.6	0.09	24.4	0.09
Dead-End Count	6	0.09	3	0.09	6	0.09	13	0.09	4	0.09
Cul-de-sac Count	0	0.09	2	0.09	1	0.09	1	0.09	2	0.09

Table 6. Table of attribute values and weights for each of the five fitness functions.

Three additional fitness functions were also used which factored in two, four, and six attributes, rather than eight, using the same attribute values as *f1* for the evaluated attributes. These additional fitness functions will be referred to as sub-fitness functions and are labelled *subf1*, *subf2* and *subf3*. The purpose of these sub-fitness functions was to explore the difference in the GA's performance when evaluating fewer attributes. Table 7 displays the attribute values and weights of the three sub-fitness functions.

	Fitness Functions							
	f1		*subf1*		*subf2*		*subf3*	
Grid Size:	16x16		16x16		16x16		16x16	
Attributes	Value	Weight	Value	Weight	Value	Weight	Value	Weight
Accessible Area Count	1	0.27	1	0.75	1	0.43	1	0.33
Average Area Size	-	0.0	-	0.0	-	0.0	-	0.0
Largest Area Size	89	0.18	-	0.0	89	0.29	89	0.22
Passage Count	7	0.09	-	0.0	-	0.00	7	0.11
Average Passage Length	5.71	0.09	-	0.0	-	0.00	5.71	0.11
Room Count	4	0.09	4	0.25	4	0.14	4	0.11
Average Room Size	10.75	0.09	-	0.0	10.75	0.14	10.75	0.11
Dead-End Count	6	0.09	-	0.0	-	0.00	-	0.0
Cul-de-sac Count	0	0.09	-	0.0	-	0.00	-	0.0

Table 7. Table of attributes and their values for the three sub-fitness functions. Note that *f1* is also included. This is to demonstrate that the same attribute values were used in the sub-fitness functions and for ease of comparison.

The attribute values that defined each fitness function were determined by extracting the attributes of five manually produced level layouts with the aim of finding CA rules capable of generating level layouts that would match the style of the manually produced levels. Because level layouts can change in both size and complexity, different CA grid sizes (in terms of number of grid squares) were used to accommodate these factors. The grid sizes were chosen based on the manually produced layouts of each fitness function, and are entries in Table 6 and Table 7. Each fitness function will now be described in more detail.

The first fitness function, *f1*, aimed to produce small level layouts of 89 traversable cells that were all connected to form only one traversable area. For example, looking at *f1*'s base layout in Figure 20, the traversable area is defined by the green, red, blue, and white areas that are linked together. These colours represent the layouts rooms, passageways, junctions, and dead-ends respectively. So looking at the goal layout of *f1* in Figure 20, it can be seen that it contains four small rooms. There were no culs-de-sac in *f1*'s base layout but six dead-ends. The values assigned to each of *f1*'s attributes were listed in Table 6.

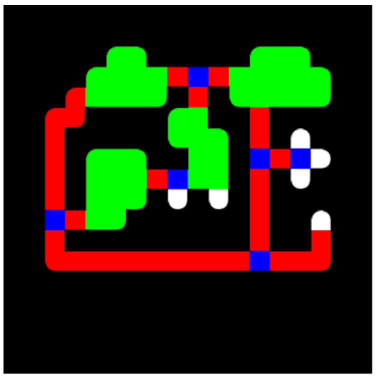


Figure 20. A colour coded version of f1's base level layout.

This base level layout was also used to design the three sub-fitness functions. When selecting which attributes to evaluate in the sub-fitness functions, the number of traversable areas was considered the most important. The reason for this is because even if a layout met every other criterion, it would not be useful as a maze-like game level if all of the passageways and rooms were disconnected. Rooms were considered the second most important attribute. This was because if all of the areas were connected and the rooms met their criteria, then the passageways should be implicitly created. Passageways were considered the next most important aspect of the layouts since they had a greater effect on the layouts than the number of dead-ends or culs-de-sac. Using these as guidelines, it was decided that *subf1* would only evaluate the accessible area count and the room count, while *subf2* additionally evaluated largest area size and average room size. The third sub-fitness function, *subf3*, evaluated the four attributes from *subf2* as well as passage count and average passage length.

The second fitness function, *f2*, aimed to produce level layouts of a single traversable area with more rooms than *f1*'s goal layout and four more passageways. The base layout for *f2* contains two culs-de-sac and only three dead-ends. A colour coded version of the base layout for *f2* can be seen in Figure 21.

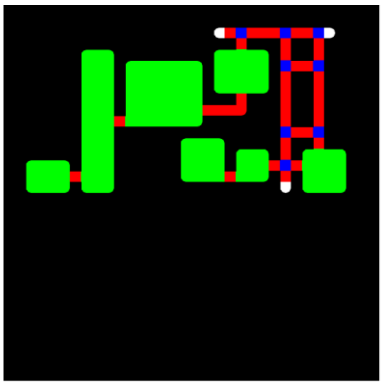


Figure 21. A colour coded version of f2's base level layout.

Figure 22 illustrates a colour-coded version of *f3*'s base layout. From this layout it can be seen that *f3* aimed to produce layouts of a single traversable area with several small rooms and many long passageways. This fitness function also aimed to produce six dead-ends and a single cul-de-sac.

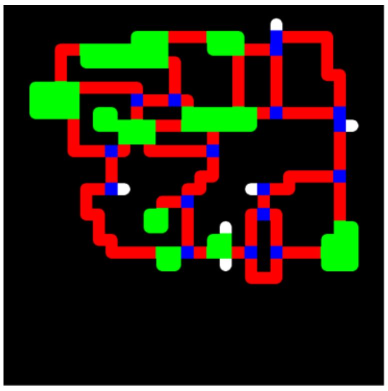


Figure 22. A colour coded version of f3's base level layout.

The base layout for the fourth fitness function, *f4*, was quite visually distinct in comparison to the others and is displayed in Figure 23. It contains four traversable areas made up of large rooms and only a few passageways. The use of additional traversable areas in this layout was to allow for special connections to be made between traversable areas as a post processing step in level design. Such special connections may include ladders in games of the platformer genre, and teleportation pads in games of the first person shooter (FPS) genre. This base layout also contained the most dead-ends but only a single cul-de-sac.

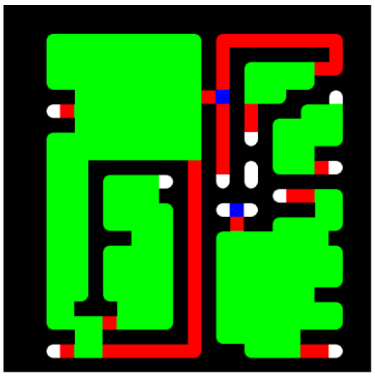


Figure 23. A colour coded version of f4's base level layout.

The fifth fitness function, *f5*, aimed to produce level layouts of two traversable areas with three passageways of length one, represented by the red sections. The base layout of *f5* (Figure 24) also had five moderate sized rooms, four dead-ends, and two culs-de-sac.

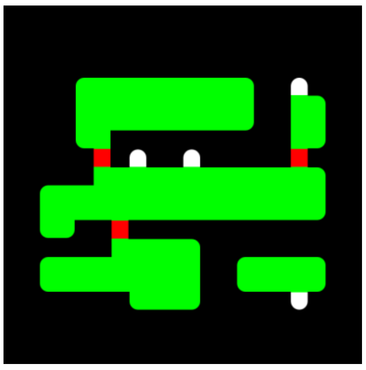


Figure 24. A colour coded version of f5's base level layout.

4.2 The Experiments

Understanding the fitness functions used and their differences, the experiments will now be described. Four sets of experiments were carried out during this research and were labelled as set 1A, 1B, 2, and 3. Each set consisted of 12 experiments and differed from one another by three factors that are listed below.

- The **chromosome representation** that was used.
- The **neighbourhood radius**. This determined the size of the Moore neighbourhood that was used with the CAs.
- The **mutation rate**. This referred to the GA's mutation probability which affected the chance of each gene in a chromosome being mutated.

Set 1A and 1B both used the direct representation with a neighbourhood radius of 1 but used different mutation rates. Set 1A used a standard mutation rate of one over the chromosome length (Bäck, 1993), while set 1B used a larger mutation rate of 0.01 for exploratory purposes. Set 2 differed from sets 1A and 1B by its chromosome representation. Set 2 used the indirect representation and shared set 1A's mutation rate of one over the chromosome length. Set 3 is identical to set 2 except that it uses a larger neighbourhood radius of 2. Table 8 lists the values of these factors that were used for each set.

As mentioned earlier, each set consisted of twelve experiments. These experiments were defined by a unique combination of two factors listed below.

- The **number of cell states**. Experiments were carried out with 2, 3, and 4 cell states.
- The **number of CA iterations**. This was the number of times a CA rule set was applied to each pre-generated maze in the GA's fitness function. The numbers of CA iterations that were used in this research were 1, 5, 10, and 25.

The experiments within each set inherit all of that set's factor settings and test all twelve combinations of different numbers of cell states with different numbers of CA iterations. Table 9 lists the cell state and CA iteration factor values used for each experiment within any given set.

	Experiment Sets			
Factors	*Set 1A*	*Set 1B*	*Set 2*	*Set 3*
Mutation Rate	1/chromosome length	0.01	1/chromosome length	1/chromosome length
Chromosome Representation	Direct	Direct	Indirect	Indirect
Neighbourhood Radius	1	1	1	2

Table 8. Table of the factor values used in each set of experiments.

	Experiments within a Set											
Factors	*#1*	*#2*	*#3*	*#4*	*#5*	*#6*	*#7*	*#8*	*#9*	*#10*	*#11*	*#12*
Cell State Count	2	2	2	2	3	3	3	3	4	4	4	4
CA Iterations	1	5	10	25	1	5	10	25	1	5	10	25

Table 9. Table of the factor values used in each experiment for any given set.

As can be seen in Table 9, each experiment has been given an index in the form of a number in the range of [1, 12]. Experiments are referred to by this index in the following manner, expa_b, where a is the set that the experiment belongs to and $_b$ is the experiment's index into that set. For example, the fifth experiment in set 1B is expressed as exp$1B_5$.

Each of the four sets of experiments was run using each of the five fitness functions described in section 4.1. In addition, sets 1B, 2, and 3 were run using each of the three sub-fitness functions. The next section discusses the results from running these experiments.

4.3 Results and Findings

This section is divided into two sub-sections that cover visual analysis of layouts that were generated using the proposed approach, and an analysis to determine the impact of different factors on the fitness values. Section 4.3.1 details the visual analysis which looks at generated layouts that were produced by using each fitness function, in comparison to their fitness functions goal layout. Section 4.3.2 details the analysis of the factors that were varied across all of the experiments. Analysis of variance (ANOVA) was used to determine if the factors had a significant effect on the fitness values that were achieved. For factors that were found to have a significant effect on fitness, the least significant difference (LSD) post-hoc test was performed to find which levels of these factors had the greatest effect on the fitness values.

4.3.1 Visual Analysis

The purpose of the visual analysis is to examine whether each fitness function succeeded in finding CA rules that were capable of generating layouts with a similar appearance to its goal layout. The comparison is to find layouts that look similar in style to the goal layout as though they could be from the same video game. Finding layouts that are identical to the goal layout is not the aim of the visual comparison as level layouts need to differ from one another. Figure 25 displays two levels from the video game "Sonic the Hedgehog" to demonstrate this point.

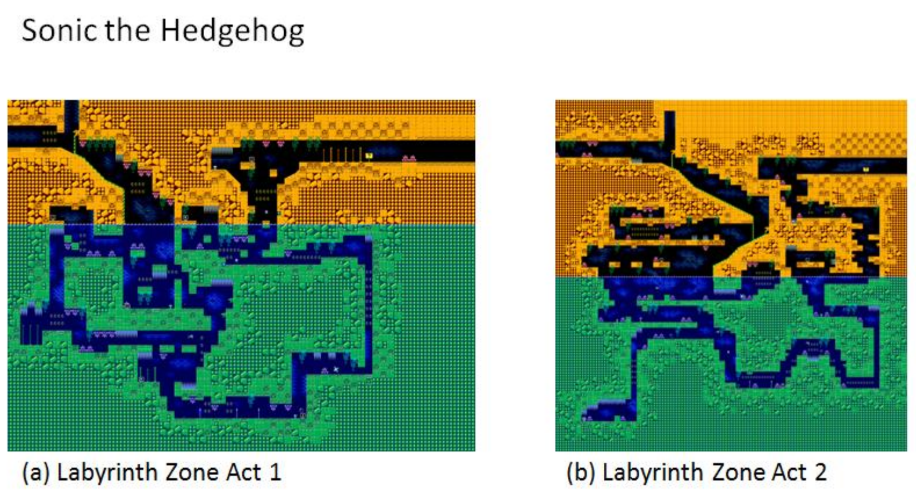


Figure 25. An example of how two level layouts from the same game are similar to one another, but not identical (Images Retrieved from the Sonic Retro Website).

To perform these comparisons, CA rule tables generated by the GA process, that were given high fitness values, were chosen and applied to ten mazes that were generated by the modified recursive back tracker algorithm which was described in section 3.1. The layouts that the CA produced were assigned an ASM, which was calculated by extracting the layout attributes and comparing them to the attributes of their goal layout. This is the same process that evaluates level layouts in the fitness function which was described in section 3.2.2. However, the ASM is not to be confused with the chromosomes' fitness. A chromosome's fitness is the average of all ASMs that are assigned to level layouts that are produced by applying the chromosome to a collection of 100 maze configurations. Layouts with a high ASM were chosen to be included in this section to demonstrate the relationship between the similarity in visual appearance and the similarity of attribute values. Determining visual similarity is a subjective process. What may appear visually similar to some may not be visually similar to others. Therefore the criteria that are used to evaluate visual similarity in this research are listed below.

- Size of the traversable areas. If two layouts are of similar size they are considered more visually similar than two layouts of different sizes.
- The proportion of negative space to positive space. In the visual comparisons, traversable areas are considered positive space while non-traversable areas are considered negative space. If the goal layout's positive space contains sections of negative space within it, then it is ideal that the generated layouts contain a similar amount of negative space within its positive space.

- Structure of the traversable areas. During the comparison of the layouts, certain structural elements are considered. These elements include the shape of the layouts rooms and whether the layouts passageways run diagonal, orthogonal, or both.

The rest of this section details the visual analysis for each fitness function. The analysis evaluates whether the generated layouts met their fitness functions criteria, and whether they were visually similar to their fitness functions goal layout.

Fitness Function 1

The primary goal of *f1*, as shown in Table 6, was to give good evaluations to CA rule tables that produce layouts of a single traversable area (weighting of 0.27), made up of approximately 89 cells (weighting of 0.18), similar to its goal layout which is displayed in Figure 26. The other attributes were given less importance, with a weighting of 0.09, and are listed in section 4.1.

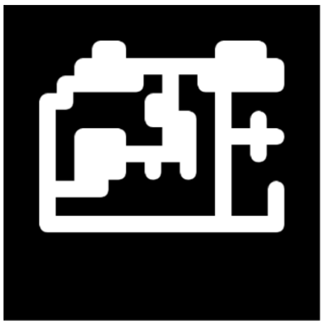


Figure 26. Goal layout of f1.

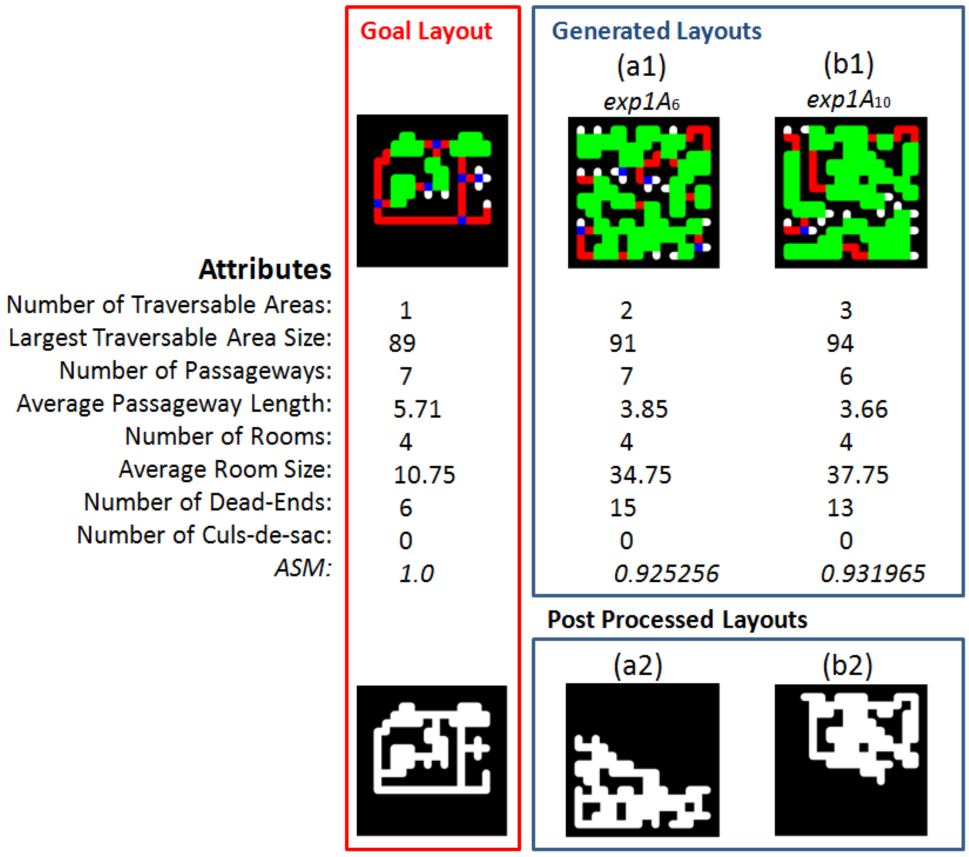
The rest of this section will discuss the results from using *f1* with each of the four experiment sets. Sets 1B, 2, and 3 will also include the results from using the three sub-fitness functions as they used the same goal layout as *f1*.

Set 1A

Figure 27 displays two level layouts that were generated by chromosomes that were given a high fitness by *f1*. These layouts were generated by chromosomes using experiments $exp1A_6$ and $exp1A_{10}$. Both of these experiments used five CA iterations, with one using three cell states, and the other using four cell states. These chromosomes were selected as they were assigned the highest fitness value by *f1*. Also displayed in Figure 27 is the goal layout of *f1* for comparison with the generated layouts. Each layout is colour coded and has its attributes listed underneath it. The colour code is listed below.

- Red areas: Passageways
- Green areas: Rooms
- Blue areas: Junctions
- White areas: Dead-Ends
- Black areas: Not Traversable

Underneath each list of attributes is a post processed version of the level layout. The post processed layouts are rendered in black and white and only displays the largest traversable area. These are displayed for a clearer visual comparison to the goal layout.



Attributes	Goal Layout	(a1) *exp1A6*	(b1) *exp1A10*
Number of Traversable Areas:	1	2	3
Largest Traversable Area Size:	89	91	94
Number of Passageways:	7	7	6
Average Passageway Length:	5.71	3.85	3.66
Number of Rooms:	4	4	4
Average Room Size:	10.75	34.75	37.75
Number of Dead-Ends:	6	15	13
Number of Culs-de-sac:	0	0	0
ASM:	*1.0*	*0.925256*	*0.931965*

Figure 27. Two generated layouts evaluated by f1 in comparison with f1's goal layout based on experiments from set 1A. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

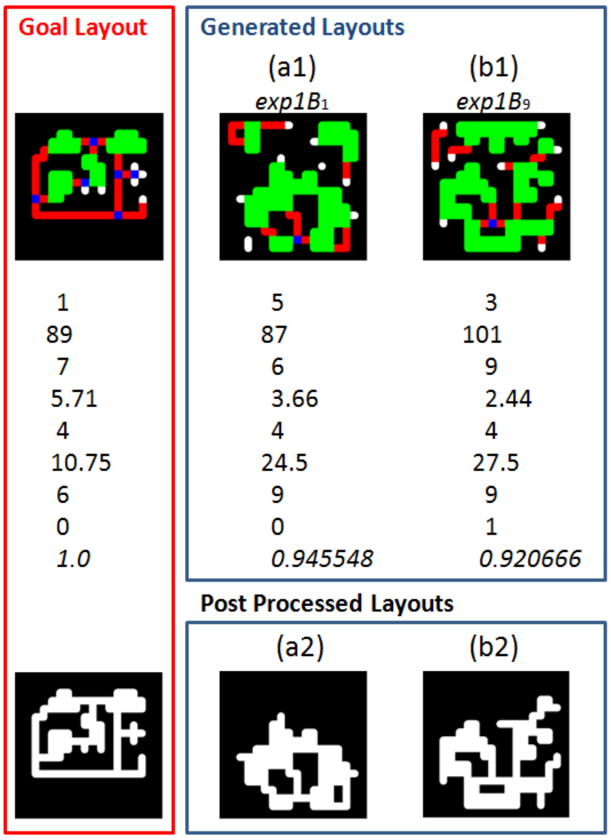
As can be seen in Figure 27, the generated layouts (a1) and (b1) have high ASM values. However, neither layout shares a particularly close visual resemblance to the goal layout. Both of the layouts contain too many traversable areas, which is a contributor to the lack of correlation between high ASM and the layouts not being very similar to the goal. This is because disconnected areas can contain some of the rooms, passageways, culs-de-sac, and dead-ends that were desired as part of the main traversable area. For example, layout (b1) produced the correct number of rooms, which contributed to the high ASM, but two of those rooms are not connected to the main traversable area. This means that the main traversable area, presented in (b2), does not contain the desired number of rooms which may have subtracted from its visual similarity with the goal layout.

The size of both layouts (a1) and (b1)'s largest traversable area is close to the desired value with (a1)'s being two cells larger (91 vs. 89) and (b1)'s being five cells larger (94 vs. 89). Comparing the number of passageways contained in the layouts and their average length, (a1) was the closest, as it contained the exact number of desired passageways but with a smaller average length of 3.85 vs. 5.71. Layout (b1) produced one less than the desired number of passageways and an even shorter average length of 3.66 vs 5.71. Both layouts (a1) and (b1) contained the correct number of rooms but with an average room size over thrice the desired value. Both layouts contained over twice the desired number of dead-ends and the correct number of culs-de-sac.

Comparing the post processed layouts, (a2) and (b2), to the criteria listed in section 4.3.1, both layouts have only a slight resemblance the goal layout. Both layouts are almost the same size as the goal layout with some of the rooms consisting of box like shapes and some containing short diagonal walls similar to the goal layout. The passageways in the generated layouts are mostly orthogonal, like the goal layouts, but neither layout contains enough negative space within the traversable area. Their negative space, particularly in (a2), is sparse and un-concentrated, with small pockets of negative space spread out over the layout.

Set 1B

Set 1B used the same parameters as set 1A except it used a higher mutation rate of 0.01. The generated layouts from set 1B are quite visually similar to the goal layout and were both generated from a chromosome that was evolved using a single CA iteration with two cell states. Figure 28 displays two colour coded layouts generated from rule tables that were produced by the GA process with the parameters of set 1B and using *f1* as the objective function.



Attributes	Goal Layout	(a1)	(b1)
Number of Traversable Areas:	1	5	3
Largest Traversable Area Size:	89	87	101
Number of Passageways:	7	6	9
Average Passageway Length:	5.71	3.66	2.44
Number of Rooms:	4	4	4
Average Room Size:	10.75	24.5	27.5
Number of Dead-Ends:	6	9	9
Number of Culs-de-sac:	0	0	1
ASM:	*1.0*	*0.945548*	*0.920666*

Figure 28. Two generated layouts evaluated by *f1* in comparison with *f1*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

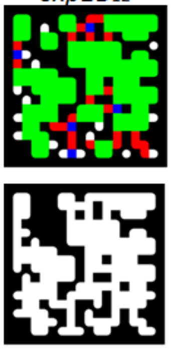
Both the layouts (a1) and (b1) contain similar attributes to those of the goal layout. Looking at Figure 28, it can be seen that neither layout contained only a single traversable area. Layout (a1)'s largest traversable area was very similar to the goal layouts, being only two cells smaller, while layout (b)'s largest traversable area was not as close, being twelve cells larger than the goal layout. When comparing the number of passageways contained in the layouts and their average length, (a1) was the closest, as it contained only one less than the desired number of passageways and with an average length of 3.66 vs 5.71. Layout (b1) produced two too many passageways and a shorter average length of 2.44 vs 5.71. Both layouts (a1) and (b2) contained the correct number of rooms but with an average room size over twice the desired value. Both layouts contained three too many dead-ends and close to the correct number of culs-de-sac, with layout (a1) containing zero and layout (b1) containing one.

Comparing the post processed layouts, (a2) and (b2), to the criteria listed in section 4.3.1, both layouts are similar to the goal layout. Both layouts are almost the same size as the goal layout with the rooms generally consisting of box like shapes while still containing some short diagonal walls similar to the goal layout. The passageways in the generated layouts are mostly orthogonal, like the goal layouts, and both layouts almost contain the same amount of negative space within the traversable area.

Figure 29 shows two layouts generated from chromosomes that were assigned low fitness values by the GA process where *f1* is the objective function. Both layouts were generated from a chromosome that was evolved using four cell states and 25 CA iterations and are displayed to give a more complete picture of how layouts with a high ASM differ from layouts with a low ASM. Both the colour coded and post processed versions of the layouts are shown with each layouts attributes.

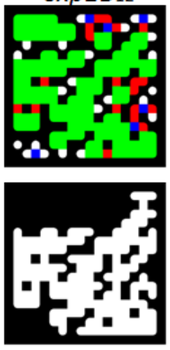
(a)

*exp1B*12



(b)

*exp1B*12

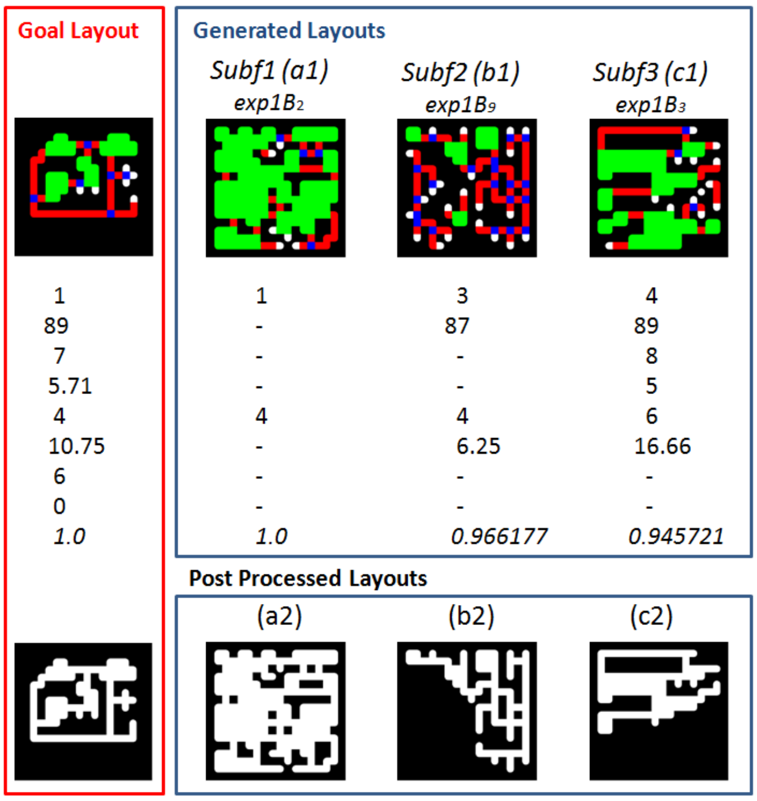


(a)	**Attributes**	(b)
4	Number of Traversable Areas	6
173	Largest Traversable Area Size	141
11	Number of Passageways	8
2.63	Average Passageway Length	3.25
8	Number of Rooms	5
17	Average Room Size	27.6
16	Number of Dead-Ends	23
1	Number of Culs-de-sac	0
0.748634	*ASM*	*0.781041*

Figure 29. Examples of generated layouts evaluated by *f1* with poor fitness values. Both the colour coded and post processed versions of the layouts are displayed, along with their attributes listed beside them.

From Figure 29 it is clear that the layouts with a low ASM do not visually resemble *f1*'s goal layout. The traversable areas of the layouts are too large and contain too many small pockets of negative space within them, although the shape of their rooms and passageways are similar to that of the goal layouts. Looking at their attributes, the layouts contained too many traversable areas with the largest one being approximately twice the desired value. Neither layout contained the correct number of passageways with an average passageway length of between 2 and 3 less than the desired value. Layout (a) contains twice as many rooms as the goal layout, while layout (b) contained five which was close to the desired value. However, both layouts average room size is too high. Both layouts contained too many dead-ends but a similar number of culs-de-sac to the goal layout.

Set 1B was also run using each of the three sub-fitness functions described in section 4.1. Figure 30 displays a single layout generated by CA rules that were evaluated using each of the sub-fitness functions. Beneath each layout is a list of their attributes and their post processed version.



Attributes	Goal Layout	Subf1 (a1)	Subf2 (b1)	Subf3 (c1)
Number of Traversable Areas:	1	1	3	4
Largest Traversable Area Size:	89	-	87	89
Number of Passageways:	7	-	-	8
Average Passageway Length:	5.71	-	-	5
Number of Rooms:	4	4	4	6
Average Room Size:	10.75	-	6.25	16.66
Number of Dead-Ends:	6	-	-	-
Number of Culs-de-sac:	0	-	-	-
ASM:	*1.0*	*1.0*	*0.966177*	*0.945721*

Figure 30. Colour coded layouts generated from CA rules that were evaluated using each of the sub-fitness functions. Each layouts attributes are listed beneath them along with their post processed versions.

From these results it is clear that evaluating two attributes is not enough to give the layouts a similar appearance to the goal layout, despite the fact that both attributes matched the goal layouts exactly. However, using four and six attributes produced more visually similar results with both layouts (b2) and (c2) being a similar size to the goal layout with a similar amount of negative space. Layout (b2)'s negative space was spread out and formed too many pockets of non-traversable areas, detracting from the visual similarity, while (c2)'s spread of negative space was more accurate. Layouts (b2) and (c2) both contain rooms of similar shapes to the goal layout, although (c2)'s rooms do not contain any diagonal walls, reducing its similarity to the goal layout. And both layouts contain orthogonal passageways similar to the goal layout.

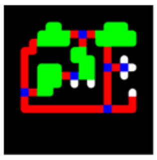
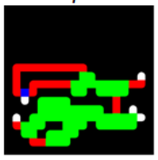
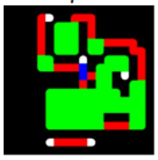
Set 2

Set 2 used the same parameters as set 1A except for the chromosome representation. Figure 31 displays two colour coded layouts generated from a rule table that was evolved using three cell states and five CA iterations, and was given a high fitness value by the objective function *f1*. Looking at the layouts in Figure 31 it is clear that the indirect representation made a significant difference in visual similarity between the generated layouts and the goal layout due to a neater and more structured appearance.

Goal Layout

Generated Layouts

(a1) *exp2*$_6$ (b1) *exp2*$_6$

Attributes	Goal Layout	(a1)	(b1)
Number of Traversable Areas:	1	1	2
Largest Traversable Area Size:	89	91	118
Number of Passageways:	7	5	6
Average Passageway Length:	5.71	5.2	5.5
Number of Rooms:	4	2	4
Average Room Size:	10.75	30	21.5
Number of Dead-Ends:	6	5	5
Number of Culs-de-sac:	0	0	1
ASM:	*1.0*	*0.951396*	*0.909426*

Post Processed Layouts

(a2) (b2)

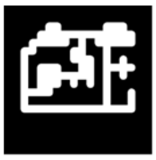
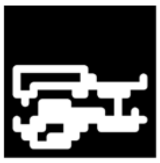
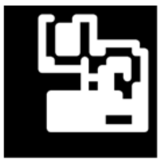
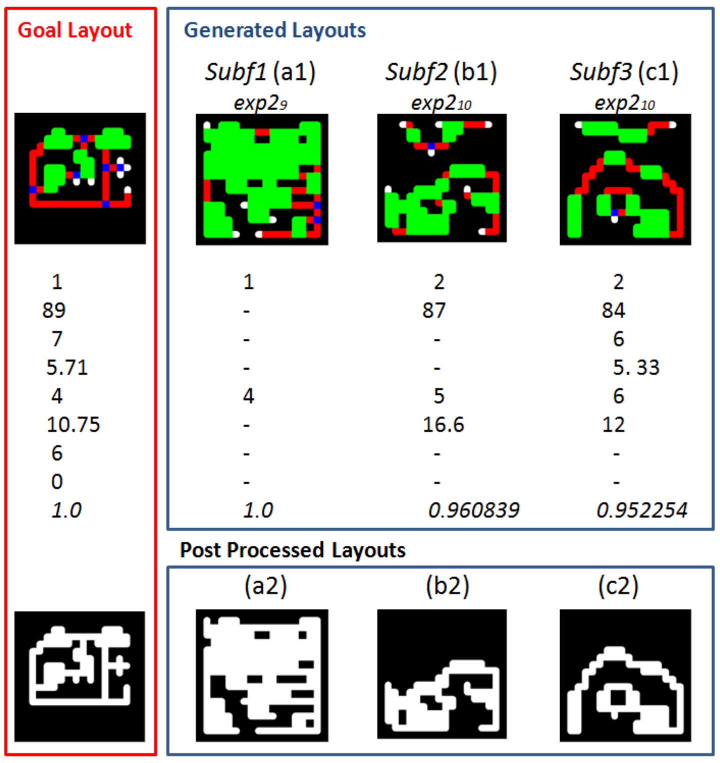




Figure 31. Two generated layouts evaluated by f1 in comparison with f1's goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

Layouts (a1) and (b1) shared similar attributes to those in sets 1A and 1B. Layout (a1) contains a single traversable area of the approximate size of the goal layout with (b1) containing two traversable areas and a larger traversable area size. Layout (a1) had two less passageways than the goal layout but with a very similar average length of 5.2. Layout (b1) contained one too few passageways and a very similar average length of 5.5. Layout (a1) contains only two rooms, as opposed to the desired four, with an average room size three times larger than the goal layout. Layout (b1) contains the desired number of rooms and has an average room size twice as large as the goal layout. Both layouts contained five dead-ends, opposed to the desired six, with layout (a1) containing no culs-de-sac while (b1) contained one.

Comparing the post processed layouts, (a2) and (b2), to the criteria listed in section 4.3.1, both layouts are similar to the goal layout. Both layouts are almost the same size as the goal layout with the rooms consisting of box like shapes with some containing some short diagonal walls similar to the goal layout. The passageways in the generated layouts are orthogonal, like the goal layouts, and both contain a similar spread of negative space.

Set 2 was also run using each of the three sub-fitness functions described in section 4.1. Figure 32 displays a single layout generated by CA rules that were evaluated using each of the sub-fitness functions. Beneath each layout is a list of their attributes and their post processed version.



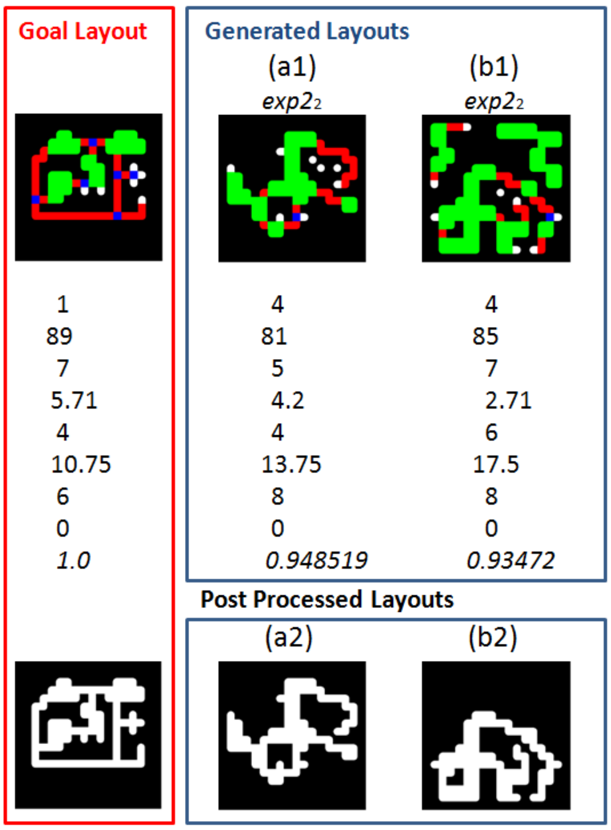
Attributes	Goal Layout	*Subf1* (a1) *exp2*$_9$	*Subf2* (b1) *exp2*$_{10}$	*Subf3* (c1) *exp2*$_{10}$
Number of Traversable Areas:	1	1	2	2
Largest Traversable Area Size:	89	-	87	84
Number of Passageways:	7	-	-	6
Average Passageway Length:	5.71	-	-	5. 33
Number of Rooms:	4	4	5	6
Average Room Size:	10.75	-	16.6	12
Number of Dead-Ends:	6	-	-	-
Number of Culs-de-sac:	0	-	-	-
ASM:	*1.0*	*1.0*	*0.960839*	*0.952254*

Figure 32. Colour coded layouts generated from CA rules that were evaluated using each of the sub-fitness functions. Each layouts attributes are listed beneath them along with their post processed versions.

From these results it is clear that evaluating two attributes is not enough to give the layouts a similar appearance to the goal layout, despite the fact that both attributes matched the goal layouts exactly. However, using four and six attributes produced more visually similar results with both layouts (b2) and (c2) being a similar size to the goal layout with a similar spread of negative space. Layouts (b2) and (c2) both contain rooms of similar shapes to the goal layout. Layout (b2) contains orthogonal passageways, however most of (c2)'s passageways are diagonal, which is not similar to the goal layout.

Set 3

Set 3 used the same parameters as set 2 except used a larger neighbourhood radius. This did not appear to have much of an effect on the on the appearance of the generated layouts in comparison to those in set 2, although it did seem to produce layouts with more diagonal passageways rather than orthogonal ones. Figure 33 displays two colour coded layouts generated from a rule table that was produced by the GA process using two cell states with five CA iterations, and using *f1* as the objective function.



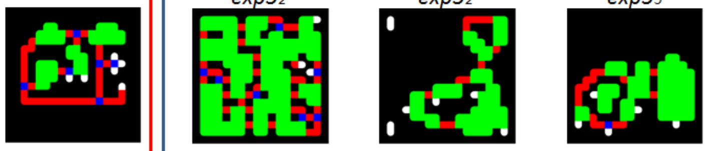
Attributes	Goal Layout	(a1)	(b1)
Number of Traversable Areas:	1	4	4
Largest Traversable Area Size:	89	81	85
Number of Passageways:	7	5	7
Average Passageway Length:	5.71	4.2	2.71
Number of Rooms:	4	4	6
Average Room Size:	10.75	13.75	17.5
Number of Dead-Ends:	6	8	8
Number of Culs-de-sac:	0	0	0
ASM:	*1.0*	*0.948519*	*0.93472*

Figure 33. Two generated layouts evaluated by *f1* in comparison with *f1*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

From the attributes of the layouts shown in Figure 33, it can be seen that all of the attributes, except for the number of traversable areas, are similar to the goal layout. Both layouts (a1) and (b1) contain three too many traversable areas but with a largest traversable area of similar size to the goal layout. Layout (a1) has a similar number of passageways to the goal layout while (b1) has the correct number of passageways, but both have a shorter average length. Layout (a1) contains the desired number of rooms and a similar average room size, while (b1) contains two too many rooms and a larger average room size. Both layouts (a1) and (b1) have a similar number of dead-ends to the goal layout, and contain the correct number of culs-de-sac.

Comparing the post processed layouts, (a2) and (b2), to the criteria listed in section 4.3.1, both layouts are similar to the goal layout. Both layouts are almost the same size as the goal layout with the rooms consisting of box like shapes. Both layouts (a2) and (b2) contain a similar spread of negative space to the goal layout, however, the passageways in (a2) and (b2) are mostly diagonal, which is unlike the goal layout.

Set 3 was also run using each of the three sub-fitness functions described in section 4.1. Figure 34 displays a single layout generated by CA rules that were evaluated using each of the sub-fitness functions. Beneath each layout is a list of their attributes and their post processed version.



Attributes	Goal Layout	*Subf1* (a1) $exp3_2$	*Subf2* (b1) $exp3_2$	*Subf3* (c1) $exp3_5$
Number of Traversable Areas:	1	1	3	2
Largest Traversable Area Size:	89	-	91	91
Number of Passageways:	7	-	-	6
Average Passageway Length:	5.71	-	-	2. 66
Number of Rooms:	4	4	5	5
Average Room Size:	10.75	-	15	16
Number of Dead-Ends:	6	-	-	-
Number of Culs-de-sac:	0	-	-	-
ASM:	*1.0*	*1.0*	*0.953603*	*0.957869*

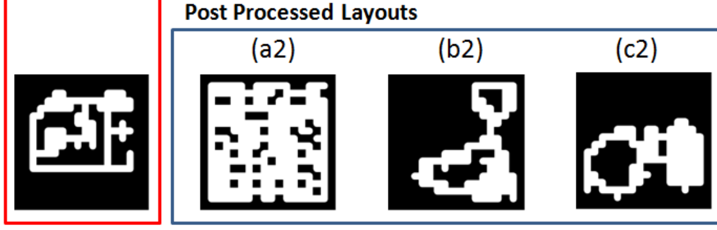


Figure 34. Colour coded layouts generated from CA rules that were evaluated using each of the sub-fitness functions. Each layouts attributes are listed beneath them along with their post processed versions.

Once again the evaluation of two attributes failed to produce visually similar layouts to the goal layout. However, using four and six attributes appears to be enough to produce layouts with a visual similarity to the goal layout. Both layouts (b2) and (c2) are a similar size to the goal layout with a similar spread of negative space. But these layouts lose similarity to the goal layout by the number of diagonal walls and passageways.

Results from Fitness Function 2

The aim of using fitness function 2 was to produce level layouts that were visually similar to its goal layout which is displayed in Figure 35. Unfortunately none of the experiments produced visually similar results.

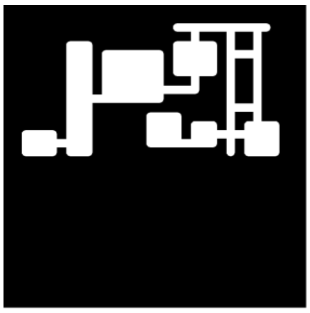
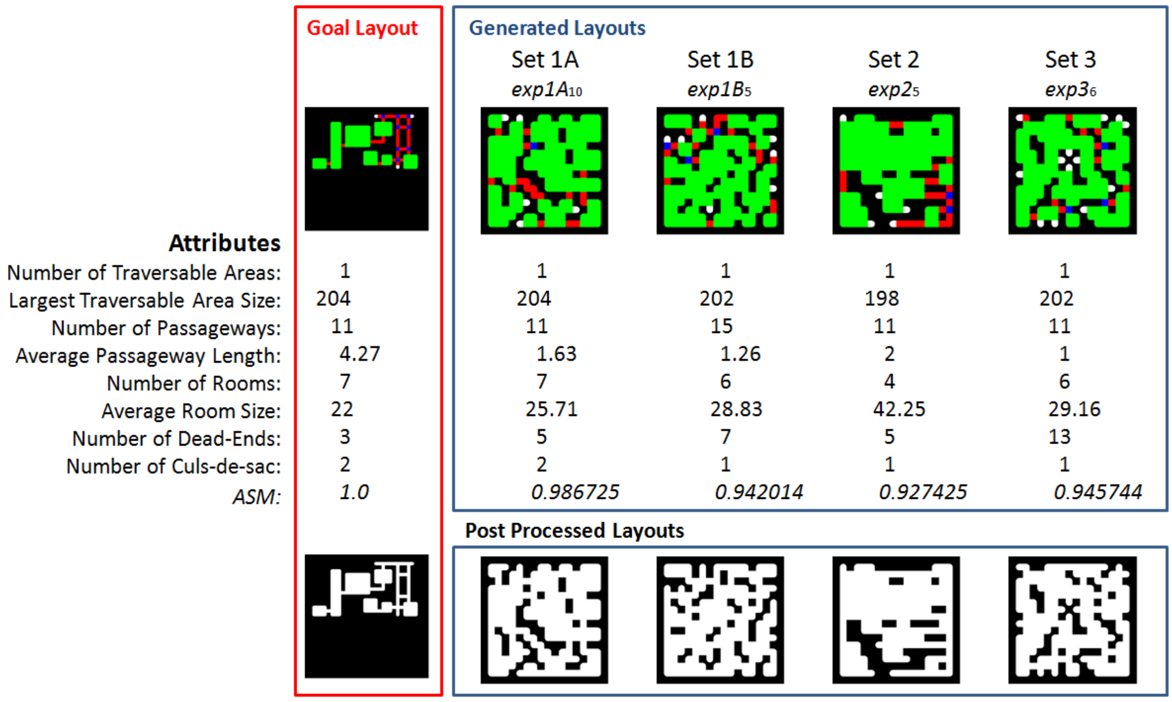


Figure 35. Goal layout of *f2*.

The layouts produced using *f2* did not have any visual similarity to the goal layout, even though they have high ASMs. Figure 36 displays one colour coded layout that was produced from each set of experiments along with their attributes and their post processed version.



Attributes	Goal Layout	Set 1A *exp1A10*	Set 1B *exp1B5*	Set 2 *exp25*	Set 3 *exp36*
Number of Traversable Areas:	1	1	1	1	1
Largest Traversable Area Size:	204	204	202	198	202
Number of Passageways:	11	11	15	11	11
Average Passageway Length:	4.27	1.63	1.26	2	1
Number of Rooms:	7	7	6	4	6
Average Room Size:	22	25.71	28.83	42.25	29.16
Number of Dead-Ends:	3	5	7	5	13
Number of Culs-de-sac:	2	2	1	1	1
ASM:	*1.0*	*0.986725*	*0.942014*	*0.927425*	*0.945744*

Figure 36. Four generated layouts, one for each set of experiments, evaluated by *f2* in comparison with *f2*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

As can be seen in Figure 36, the layouts attributes closely match those of the goal layout. The reason behind the lack of visual similarity is the grid size of the generated layouts does not match the grid size of the goal layout. This basis of decision was made on the fact that the goal layout only covered half of its grid, and rather than use a rectangular grid, a smaller square grid was chosen instead. The hypothesis behind this decision was that a layout of similar size and style would be produced, but in a square space. However, the newly selected grid size was too small to fit a layout of similar size and style, which is evident by viewing the largest traversable area attribute of the generated layouts. All of the largest traversable areas were very similar in size to the goal layouts, but the grid was not large enough to form the necessary negative space to produce layouts of a similar style to the goal layout.

Results from Fitness Function 3

The aim of fitness function 3 was to generate layouts of a single traversable area with small rooms and long passageways as shown in Figure 37. The exact values for these attributes are listed in Table 6 in section 4.1.

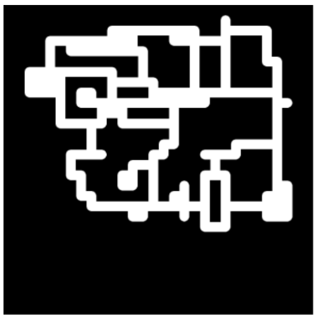
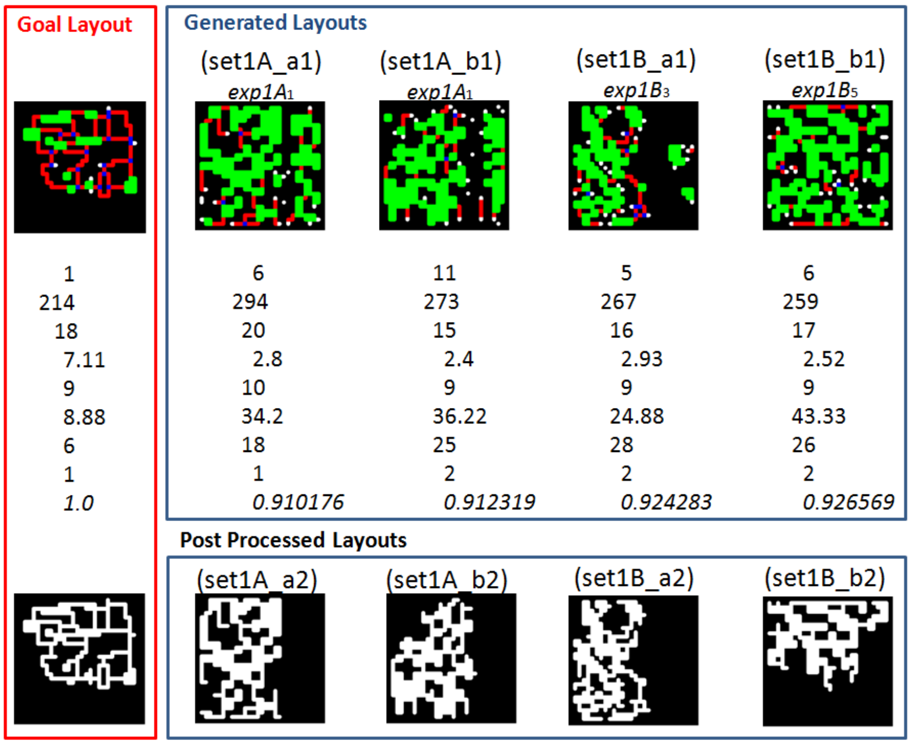


Figure 37. Goal layout of *f3*.

Using *f3*, both sets 1A and 1B produced similar results to one another but their resemblance to the goal layout was not that close. Figure 38 shows two generated layouts from both sets 1A and 1B in comparison to their goal layout. Each layouts attributes are listed beneath them along with a post processed version of the layout. As can be seen, the experiments that used a single CA iteration (set1A_a1, set1A_b1, set1B_b1) all produced similar visual results to one another, while the experiment that used ten CA iterations (set1B_a1) produced a layout with a different visual appearance.



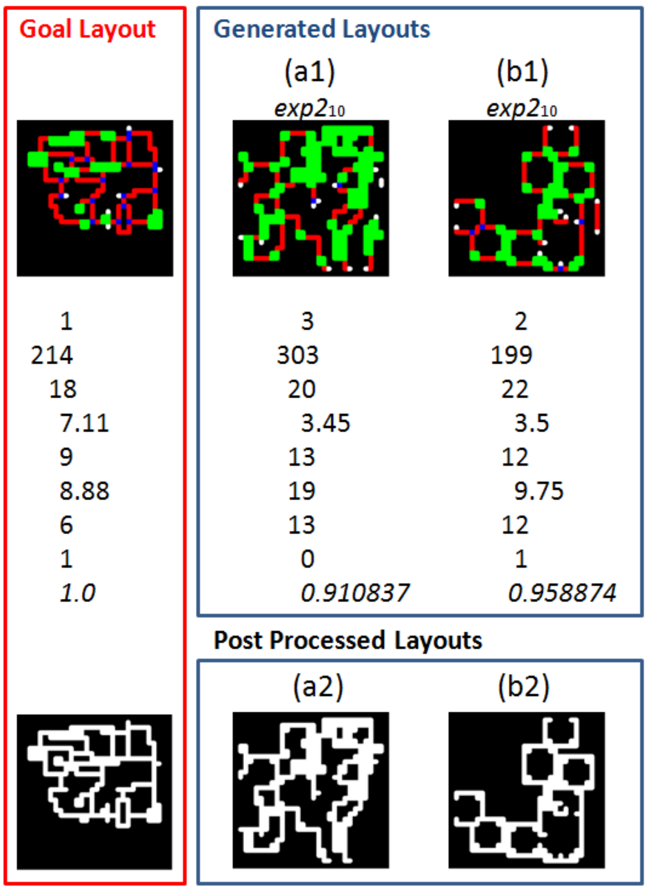
Attributes	Goal Layout	(set1A_a1) exp1A1	(set1A_b1) exp1A1	(set1B_a1) exp1B3	(set1B_b1) exp1B5
Number of Traversable Areas:	1	6	11	5	6
Largest Traversable Area Size:	214	294	273	267	259
Number of Passageways:	18	20	15	16	17
Average Passageway Length:	7.11	2.8	2.4	2.93	2.52
Number of Rooms:	9	10	9	9	9
Average Room Size:	8.88	34.2	36.22	24.88	43.33
Number of Dead-Ends:	6	18	25	28	26
Number of Culs-de-sac:	1	1	2	2	2
ASM:	*1.0*	*0.910176*	*0.912319*	*0.924283*	*0.926569*

Figure 38. Four generated layouts, two from each set 1A and 1B, evaluated by *f3* in comparison with *f3*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

The four layouts displayed in Figure 38 only contained three attributes that were similar to the goal layout. Layouts (set1A_a1), (set1A_b1), (set1B_a1), and (set1B_b1) contain between four and ten too many traversable areas, with the largest traversable area ranging between forty five and eighty cells larger than the desired size. The numbers of passageways contained within the generated layouts are close to the desired value but their average length is much smaller. Each generated layout contains the desired number of rooms, with the exception of (set1A_a1) which contains one extra room, but with a much greater than desired average room size. The generated layouts contain between three and five times the desired number of dead-ends, but close to the desired number of culs-de-sac.

Comparing the post processed layouts to the criteria listed in section 4.3.1, the generated layouts are not very similar to the goal layout. All of the layouts are of similar size to the goal layout, with layouts (set1A_a1), (set1A_b1), and (set1B_b1) having similar shaped rooms. The majority of the goal layout consists of negative space, and although layout (set1A_a2) has the most similar distribution of negative space, none of the generated layouts contain enough. The passageways in each of the layouts are too short and detract from their visual similarity to the goal layout.

Sets 2 and 3 used the indirect representation and produced very different results from one another. Figure 39 shows two layouts that were generated from set 2 in comparison to their goal layout. Both layouts were generated by a chromosome that was evolved using four cell states and five CA iterations, and each layouts attributes are listed beneath them.



Attributes	Goal Layout	(a1)	(b1)
Number of Traversable Areas:	1	3	2
Largest Traversable Area Size:	214	303	199
Number of Passageways:	18	20	22
Average Passageway Length:	7.11	3.45	3.5
Number of Rooms:	9	13	12
Average Room Size:	8.88	19	9.75
Number of Dead-Ends:	6	13	12
Number of Culs-de-sac:	1	0	1
ASM:	*1.0*	*0.910837*	*0.958874*

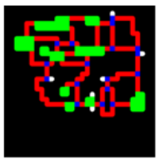
Figure 39. Two generated layouts evaluated by *f3* in comparison with *f3*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

From the attributes of the layouts shown in Figure 39, it can be seen that both generated layouts contain too many traversable areas with (a1)'s largest area being larger than desired, while (b1)'s largest area was a bit smaller. Layouts (a1) and (b1) have a similar number of passageways to the goal layout but with a shorter average length. Both layouts contain too many rooms with (a1)'s average room size being twice the desired value, although (b1)'s average room size is quite close to the goal layout. Both layouts contain twice as many dead-ends as the goal layout, with (a1) not containing any culs-de-sac and (b1) containing the desired number of one cul-de-sac.

Comparing the post processed layouts to the criteria listed in section 4.3.1, the generated layouts are quite similar to the goal layout. Both layouts (a2) and (b2) are of similar size to the goal layout, although (a2) is slightly bigger, with both layouts shaped rooms. Both layouts mostly consisted of negative space, very similar to the goal layout. The passageways in (a2) were more similar in appearance to the goal layout than the passageways in (b2). This is because (b2)'s passageways are all orthogonal and form rectangles, while (a2)'s passageways are less structured with some running diagonally similar to the goal layout.

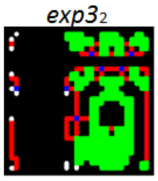
The layouts generated in set 3 were not very similar to the goal layout. Figure 40 shows two layouts that were generated from set 3 in comparison to their goal layout. Both layouts were generated by a chromosome that was evolved using two cell states with five CA iterations, and each layout's attributes are listed beneath them.

Goal Layout

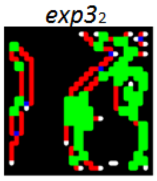


Generated Layouts

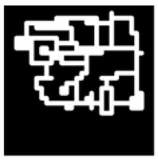
(a1) *$exp3_2$*



(b1) *$exp3_2$*

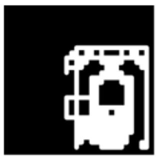


Attributes	Goal Layout	(a1)	(b1)
Number of Traversable Areas:	1	7	4
Largest Traversable Area Size:	214	211	218
Number of Passageways:	18	20	18
Average Passageway Length:	7.11	3.5	5
Number of Rooms:	9	8	7
Average Room Size:	8.88	26	25
Number of Dead-Ends:	6	12	18
Number of Culs-de-sac:	1	1	0
ASM:	*1.0*	*0.957768*	*0.960165*



Post Processed Layouts

(a2)



(b2)

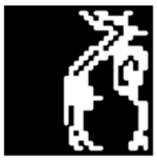


Figure 40. Two generated layouts evaluated by *f3* in comparison with *f3*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

Looking at the attribute values listed in Figure 40 it can be seen that half the attributes were similar to the desired values while the other half are not. Both layouts (a1) and (b1)'s largest traversable area is similar in size to the goal layout, and both layouts have close to the desired number of passageways, rooms, and culs-de-sac. Both layouts have too many traversable areas, shorter average passageway lengths, and larger average room sizes. Both layouts also contain two to three times the number of dead-ends.

Comparing the post processed layouts to the criteria listed in section 4.3.1, it can be seen that the generated layouts are not very similar to the goal layout. Both layouts (a2) and (b2) are of similar size to the goal layout, but their rooms are too large and complex, whereas the goal layouts rooms are small, simple, and box-like. Neither of the layouts contained a similar distribution of negative space, with some large areas and many small areas that were scattered. The passageways in (a2) were more similar in appearance to the goal layout than the passageways in (b2). This is because (a2)'s passageways are mostly orthogonal, while (a2)'s passageways are mostly diagonal.

Results from Fitness Function 4

The aim of fitness function 4 was to generate layouts containing four traversable areas with large rooms that covered the majority of the layout. The goal layout for *f4* is shown in Figure 41.

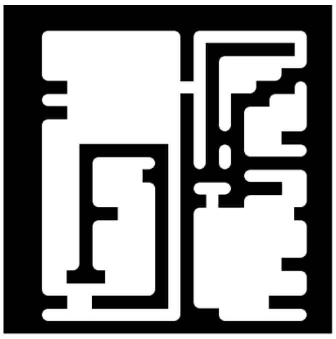
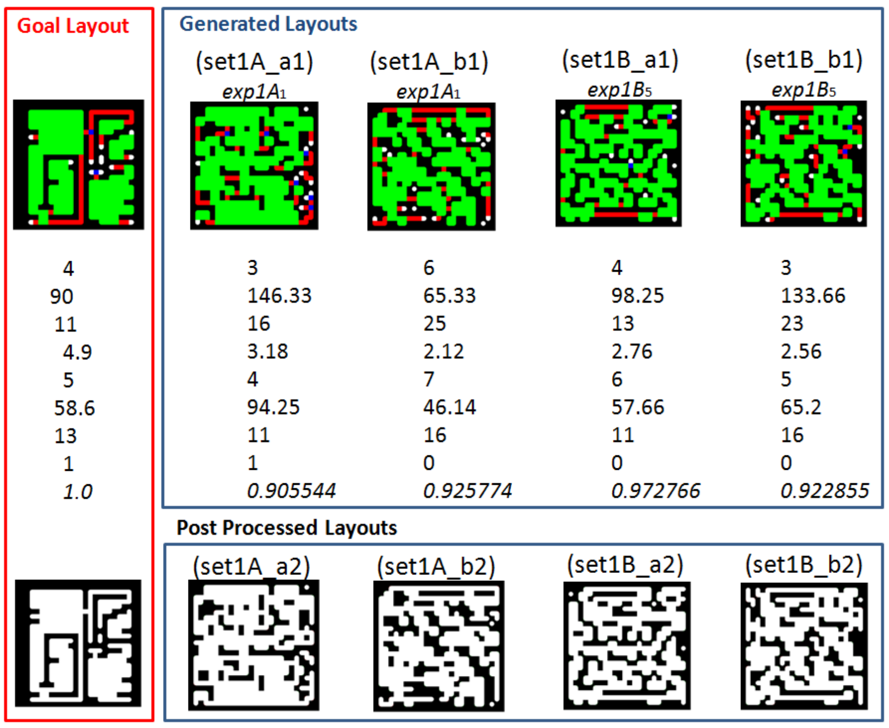


Figure 41. Goal layout for *f4*.

The layouts that were generated by chromosomes from set 1A and 1B that were given a high fitness value by *f4* were very similar to one another. Figure 42 shows two generated layouts from both sets 1A and 1B in comparison to their goal layout. Each layouts attributes are listed beneath them along with a post processed version of the layout. The post processed versions of these layouts include up to four disconnected traversable areas as this was the desired number of traversable areas used in this fitness function.



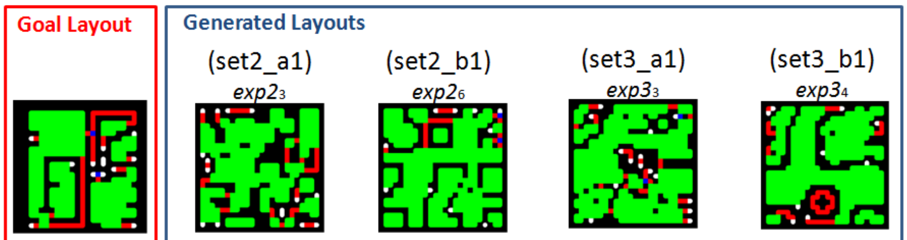
Attributes	Goal Layout	(set1A_a1) *exp1A₁*	(set1A_b1) *exp1A₁*	(set1B_a1) *exp1B₅*	(set1B_b1) *exp1B₅*
Number of Traversable Areas:	4	3	6	4	3
Average Traversable Area Size:	90	146.33	65.33	98.25	133.66
Number of Passageways:	11	16	25	13	23
Average Passageway Length:	4.9	3.18	2.12	2.76	2.56
Number of Rooms:	5	4	7	6	5
Average Room Size:	58.6	94.25	46.14	57.66	65.2
Number of Dead-Ends:	13	11	16	11	16
Number of Culs-de-sac:	1	1	0	0	0
ASM:	*1.0*	*0.905544*	*0.925774*	*0.972766*	*0.922855*

Figure 42. Four generated layouts, two from each set 1A and 1B, evaluated by *f4* in comparison with *f4*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

As can be seen in Figure 42, layout (set1B_a1) had the desired number of traversable areas while (set1A_a1) and (set1B_b1) had one less and (set1A_b1) had two more. The average size of (set1B_a1)'s traversable areas was close to the desired value while (set1B_b1) and (set1A_a1)'s average area size was too large and (set1B_b1)'s was too small. Layouts (set1A_a1) and (set1B_a1) had a similar number of passageways to the goal layout while (set1A_b1) and (set1B_b1) contain too many. All of the layouts had a shorter than desired average passageway length. Each layout contains close to the desired number of rooms and, with the exception of (set1A_a1), had a similar average room size to the goal layout. All of the layouts contained a similar number of dead-ends to the goal layout but only (set1A_a1) contained a cul-de-sac.

Comparing the post processed layouts to the criteria listed in section 4.3.1, it can be seen that the generated layouts are not very similar to the goal layout. All of the layouts are of similar size to the goal layout, but their distribution of negative space is erratic. This is unlike the goal layout as the negative space in the goal layout forms passageways of non-traversable areas, enclosing rooms within them. The layout (set1A_a1) is the most visually similar to the goal layout as it contains large open areas, but its negative space is spread out and disjointed.

Sets 2 and 3 also produced similar results to each other but, unlike sets 1A and 1B, they produced some visually similar results. Figure 43 shows two layouts that were generated from set 2 and two layouts that were generated from set 3 in comparison to their goal layout. Each layouts attributes are listed beneath them along with their post processed version.



Attributes	Goal Layout	(set2_a1)	(set2_b1)	(set3_a1)	(set3_b1)
Number of Traversable Areas:	4	7	4	5	7
Average Traversable Area Size:	90	55.71	55.62	88	62
Number of Passageways:	11	16	7	16	8
Average Passageway Length:	4.9	2.31	3	1.81	5.12
Number of Rooms:	5	9	8	7	5
Average Room Size:	58.6	37.55	51.75	56.57	76.8
Number of Dead-Ends:	13	15	10	15	9
Number of Culs-de-sac:	1	1	0	0	2
ASM:	*1.0*	*0.923705*	*0.931249*	*0.965661*	*0.944731*

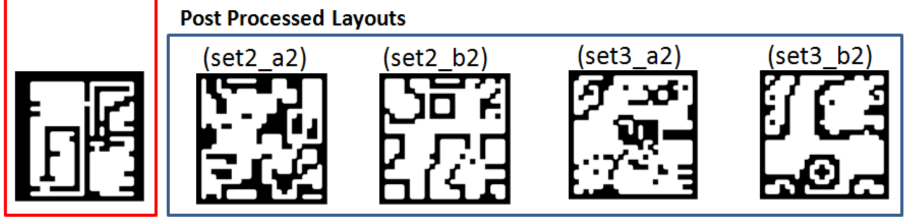


Figure 43. Four generated layouts, two from each set 2 and 3, evaluated by *f4* in comparison with *f4*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

As can be seen in Figure 43, (set2_b1) and (set3_a1) contain close to the desired number of traversable areas, while (set2_a1) and (set3_b1) contain close to twice the desired number of traversable areas. All of the layouts, except for (set3_a1), have a smaller than desired average traversable area size. All of the layouts contain either four too many or four too few passageways, with the exception of (set3_b1) which contains three too few passageways. Layout (set3_b1) was also the only layout that had a similar average passageway length, as the other three layouts average passageway length was too short. Layout (set3_b1) also contained the correct number of rooms while the other layouts contained between two and four too many. Layouts (set2_b1) and (set3_a1) had close to the desired average room size, while (set2_a1) had a smaller than desired average room size and (set3_b1) had a larger than desired average room size. Each layout contained close to the correct number of dead-ends, with (set2_a1) containing the correct number of culs-de-sac and the other layouts containing one too many or one too few culs-de-sac.

Comparing the post processed layouts to the criteria listed in section 4.3.1, it can be seen that the generated layouts are quite similar to the goal layout, specifically layout (set2_b2). All of the layouts consist mostly of positive space with negative space enclosing large areas in a similar manner to the goal layout. However, the goal layouts negative space forms thin passageways of non-traversable areas, whereas layout (set2_a2) and (set3_a2) contain several larger pockets of negative space that do not form thin passageways. Layout (set2_b2)'s traversable areas are surrounded by mostly orthogonal walls of negative space, very similar to the goal layout, while (set3_b2)'s traversable areas are not.

Results from Fitness Function 5

The aim of fitness function 5 was to generate layouts containing two traversable areas with moderately sized, rectangle rooms. The goal layout for *f5* is shown in Figure 44.

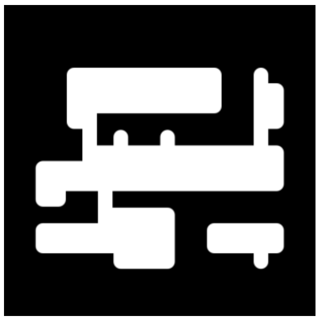
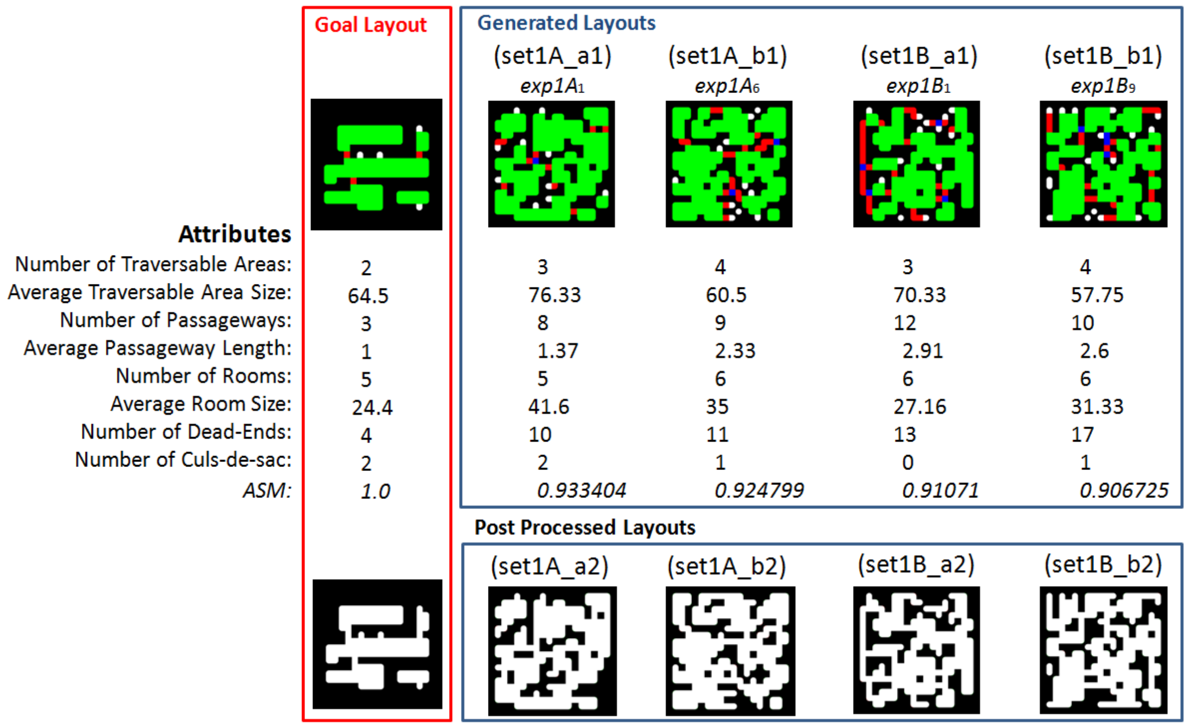


Figure 44. Goal layout of *f5*.

The layouts that were generated by chromosomes from set 1A and 1B that were given a high fitness value by *f5* were very similar to one another, although they were not similar to the goal layout. Figure 45 shows two generated layouts from both sets 1A and 1B in comparison to their goal layout. Each layouts attributes are listed beneath them along with a post processed version of the layout. The post processed versions of these layouts include up to four disconnected traversable areas as this was the desired number of traversable areas used in this fitness function.



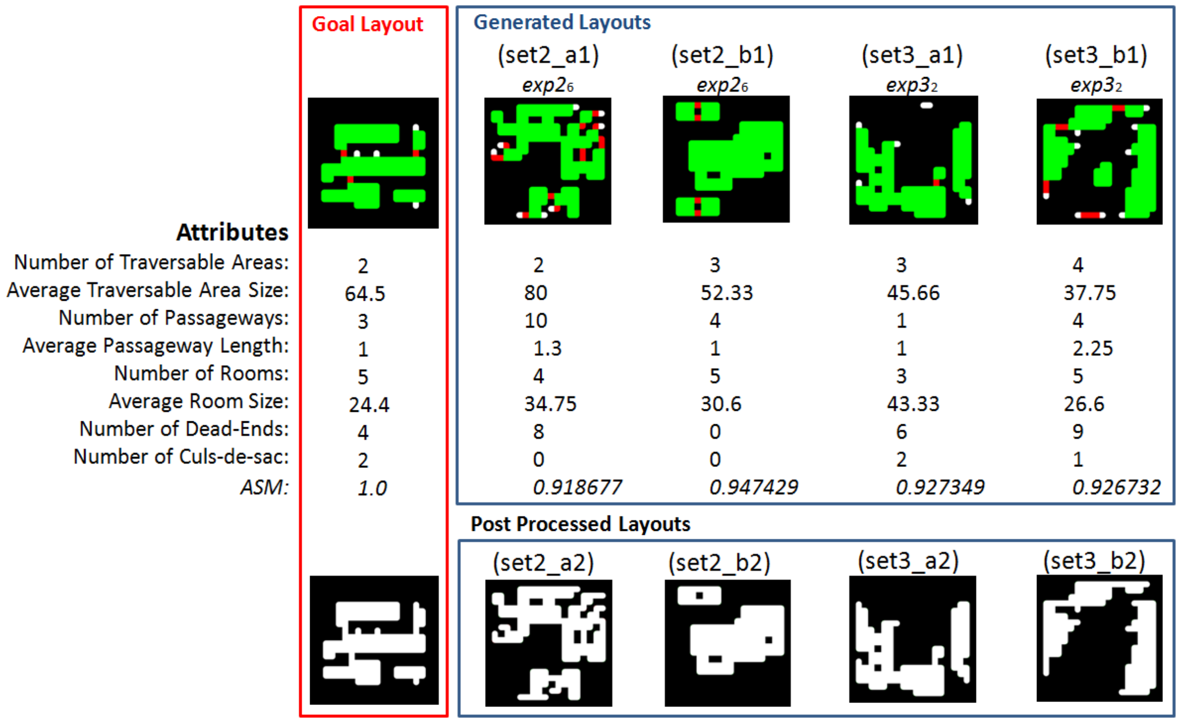
Attributes	Goal Layout	(set1A_a1) *exp1A₁*	(set1A_b1) *exp1A₆*	(set1B_a1) *exp1B₁*	(set1B_b1) *exp1B₉*
Number of Traversable Areas:	2	3	4	3	4
Average Traversable Area Size:	64.5	76.33	60.5	70.33	57.75
Number of Passageways:	3	8	9	12	10
Average Passageway Length:	1	1.37	2.33	2.91	2.6
Number of Rooms:	5	5	6	6	6
Average Room Size:	24.4	41.6	35	27.16	31.33
Number of Dead-Ends:	4	10	11	13	17
Number of Culs-de-sac:	2	2	1	0	1
ASM:	*1.0*	*0.933404*	*0.924799*	*0.91071*	*0.906725*

Figure 45. Four generated layouts, two from each set 1A and 1B, evaluated by *f5* in comparison with *f5*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

As can be seen in Figure 45, each layout contains too many traversable areas but with an average size that is close to the desired value. All of the generated layouts contain between five and nine too many passageways, with only (set1A_a1) having an average passageway length that is close to the goal layouts. The number of rooms in each layout is close to the desired value, but only (set1B_a1) has an average room size that is similar to the goal layout. All of the layouts contain too many dead-ends with (set1A_a1) being the only layout that contains the correct number of culs-de-sac.

Comparing the post processed layouts to the criteria listed in section 4.3.1, it can be seen that the generated layouts are not very similar to the goal layout. All of the layouts are larger than the goal layout and their distribution of negative space is erratic. The rooms in the goal layout are clearly rectangular while the rooms in the generated layouts are not.

Sets 2 and 3 also produced similar results to each other, and although the generated layouts are clearer and better structured, still lack a visual similarity to the goal layout. Figure 46 shows two layouts that were generated from set 2 and two layouts that were generated from set 3 in comparison to their goal layout. Each layouts attributes are listed beneath them along with their post processed version.



Attributes	Goal Layout	(set2_a1) $exp2_6$	(set2_b1) $exp2_6$	(set3_a1) $exp3_2$	(set3_b1) $exp3_2$
Number of Traversable Areas:	2	2	3	3	4
Average Traversable Area Size:	64.5	80	52.33	45.66	37.75
Number of Passageways:	3	10	4	1	4
Average Passageway Length:	1	1.3	1	1	2.25
Number of Rooms:	5	4	5	3	5
Average Room Size:	24.4	34.75	30.6	43.33	26.6
Number of Dead-Ends:	4	8	0	6	9
Number of Culs-de-sac:	2	0	0	2	1
ASM:	*1.0*	*0.918677*	*0.947429*	*0.927349*	*0.926732*

Figure 46. Four generated layouts, two from each set 2 and 3, evaluated by *f5* in comparison with *f5*'s goal layout. Each layout is colour coded to visually identify its attributes which are listed beneath each layout along with its post processed version.

As can be seen in Figure 46, (set2_a1) contains the desired number of traversable areas, while (set2_b1), (set3_a1), and (set3_b1) contain too many. Layouts (set2_b1) had a similar average traversable area size to the goal layout with the other three layouts having an average traversable area size that is too small or too large. Layouts (set1A_b1) and (set1B_b1) contain close to the desired number of passageways while (set2_a1) contains too many and (set3_a1) contains too few. All of the layouts, except for (set3_b1), had a similar average passageway length. All layouts contained close to the correct number of rooms with (set3_b1) having the most accurate average room size. None of the layout contained the correct number of dead-ends, with (set3_a1) being the only layout that contains the correct number of culs-de-sac.

Comparing the post processed layouts to the criteria listed in section 4.3.1, it can be seen that the generated layouts are not particularly similar to the goal layout, mostly due to the rooms not being rectangular in shape. Layout (set2_b2) was the most similar in visual appearance as it is mostly made up of rectangular shapes, but the lack of rooms and their large size detract from its visual similarity to the goal layout.

4.3.2 Analysis of Fitness Values through ANOVA

To investigate the impact of the interactions of the factors associated with both the GA and the CA on the five fitness functions, three-way ANOVA tests were carried out. Each ANOVA test used the chromosomes' fitness values as the dependent variable, the number of cell states and number of CA iterations as two of the independent variables, with the third independent variable being one of the following three factors: mutation rate, chromosome representation, or neighbourhood radius. Three ANOVA tests were completed for each of the five fitness functions to examine the following.

ANOVA Test 1: The impact that changes in mutation rate, number of cell states, and number of CA iterations had on the fitness values.

ANOVA Test 2: The impact that changes in the chromosome representation, number of cell states, and CA iterations had on the fitness values.

ANOVA Test 3: The impact that changes in the neighbourhood radius, number of cell states, and number of CA iterations had on the fitness values.

Two of assumptions associated with ANOVA are the assumption of normality and homogeneity of variances. To meet the assumption of normality each of the 12 experiments in each of the four sets which were associated with a specific fitness function was run 30 times. Because each sample size was equal, the assumption of homogeneity of variances is also met. Results of the ANOVA tests, in order of fitness function, starting with analysis of results from using *f1* through to *f5* are described in the remainder of this section.

Analysis of Fitness Function 1

The first ANOVA that was performed for *f1* was to examine the impact that changes in the mutation rate, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 1. To do this, data from sets 1A and 1B were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 10 shows the results from a three-way between-subjects ANOVA test with two levels of mutation rate, three levels of cell states, and four levels of CA iterations. Based on the sig, or p-value, being less than 0.05 for all effects, they were all statistically significant. The interaction effect between the three factors is, (F (6, 696) = 181.425, p < 0.05, Partial Eta Squared = 0.610).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	1.648[a]	23	.072	285.063	.000	.904[a]
Intercept	472.471	1	472.471	1879253.233	.000	1.000
MutationRate	.083	1	.083	328.211	.000	.320
CellStates	.008	2	.004	15.318	.000	.042
CAIterations	.459	3	.153	608.978	.000	.724
MutationRate * CellStates	.254	2	.127	505.324	.000	.592
MutationRate * CAIterations	.404	3	.135	535.652	.000	.698
CellStates * CAIterations	.167	6	.028	110.753	.000	.488
MutationRate * CellStates * CAIterations	.274	6	.046	181.425	.000	.610
Error	.175	696	.000			
Total	474.295	720				
Corrected Total	1.823	719				

Table 10. ANOVA Test for the fitness values achieved by the experiments in set 1A and 1B.

As the number of cell states and the number of CA iterations had a significant effect on the fitness values, two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test was performed on the mutation rate as it had only two levels, and therefore the significant interaction is between those two levels. To determine where the significant interactions lie, the least significant difference (LSD) post hoc test was performed. Table 11 shows the results from the post hoc test performed on the number of cell states, and Table 12 shows the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 11, the p-value is less than 0.05 for all interactions, which means the effect of using any number of cell states is significantly different to the effect when using any other number of cell states. The same can be seen in Table 12, where the effects of using any number of CA Iterations are significantly different to one another due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0039611583*	.00144745281	.006
	4	.0080113583*	.00144745281	.000
3	2	-.0039611583*	.00144745281	.006
	4	.0040502000*	.00144745281	.005
4	2	-.0080113583*	.00144745281	.000
	3	-.0040502000*	.00144745281	.005

Table 11. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f1* with results from sets 1A and 1B.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	.0353138389*	.00167137454	.000
	10	.0523657556*	.00167137454	.000
	25	.0679579944*	.00167137454	.000
5	1	-.0353138389*	.00167137454	.000
	10	.0170519167*	.00167137454	.000
	25	.0326441556*	.00167137454	.000
10	1	-.0523657556*	.00167137454	.000
	5	-.0170519167*	.00167137454	.000
	25	.0155922389*	.00167137454	.000
25	1	-.0679579944*	.00167137454	.000
	5	-.0326441556*	.00167137454	.000
	10	-.0155922389*	.00167137454	.000

Table 12. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f1* with results from sets 1A and 1B.

The second ANOVA that was performed for *f1* was to examine the impact that changes in the chromosome representation, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 2. To do this, data from sets 1A and 2 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 13 shows the results from a three-way between-subjects ANOVA test with two levels of chromosome representation, three levels of cell states, and four levels of CA iterations. All effects were statistically significant. The interaction effect between the three independent variables is, (F (6, 696) = 203.190, $p < 0.05$, Partial Eta Squared = 0.637).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	1.165[a]	23	.051	306.475	.000	.910[a]
Intercept	508.306	1	508.306	3075915.740	.000	1.000
CellStates	.038	2	.019	114.566	.000	.248
CAIterations	.189	3	.063	381.738	.000	.622
Representation	.272	1	.272	1648.743	.000	.703
CellStates * CAIterations	.187	6	.031	189.062	.000	.620
CellStates * Representation	.061	2	.031	184.856	.000	.347
CAIterations * Representation	.215	3	.072	434.202	.000	.652
CellStates * CAIterations * Representation	.201	6	.034	203.190	.000	.637
Error	.115	696	.000			
Total	509.586	720				
Corrected Total	1.280	719				

Table 13. ANOVA Test for the fitness values achieved by the experiments in set 1A and 2.

Once again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test was performed on the chromosome representation as it had only two levels. Table 14 shows the results from the post hoc test performed on the number of cell states, and Table 15 shows the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 14, the p-value is less than 0.05 for all interactions, which means the effect of using any number of cell states is significantly different to the effect when using any other number of cell states. The same can be seen in Table 15, where the effects of using any number of CA iterations are significantly different to one another due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	-.0174848708*	.00117350435	.000
	4	-.0114564292*	.00117350435	.000
3	2	.0174848708*	.00117350435	.000
	4	.0060284417*	.00117350435	.000
4	2	.0114564292*	.00117350435	.000
	3	-.0060284417*	.00117350435	.000

Table 14. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f1* with results from sets 1A and 2.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0366170167*	.00135504610	.000
	10	-.0142856444*	.00135504610	.000
	25	.0052495889*	.00135504610	.000
5	1	.0366170167*	.00135504610	.000
	10	.0223313722*	.00135504610	.000
	25	.0418666056*	.00135504610	.000
10	1	.0142856444*	.00135504610	.000
	5	-.0223313722*	.00135504610	.000
	25	.0195352333*	.00135504610	.000
25	1	-.0052495889*	.00135504610	.000
	5	-.0418666056*	.00135504610	.000
	10	-.0195352333*	.00135504610	.000

Table 15. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f1* with results from sets 1A and 2.

The third ANOVA that was performed for *f1* was to examine the impact that changes in the neighbourhood radius, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 3. To do this, data from sets 2 and 3 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 16 shows the results from a three-way between-subjects ANOVA test with two levels of neighbourhood radius, three levels of cell states, and four levels of CA iterations. Most of the effects were statistically significant with a p-value of less than 0.05. The interaction pair of cell states and CA iterations did not have a significant effect on the fitness values, as it has a p-value of 0.932 which is greater than 0.05. The interaction effect between the three independent variables is, (F (6, 696) = 203.190, $p < 0.05$, Partial Eta Squared = 0.018).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.601[a]	23	.026	143.642	.000	.826[a]
Intercept	515.593	1	515.593	2835229.377	.000	1.000
CellStates	.032	2	.016	87.428	.000	.201
CAIterations	.405	3	.135	742.628	.000	.762
NeighbourhoodRadius	.130	1	.130	716.443	.000	.507
CellStates * CAIterations	.000	6	5.647E-005	.311	.932	.003
CellStates * NeighbourhoodRadius	.010	2	.005	27.240	.000	.073
CAIterations * NeighbourhoodRadius	.021	3	.007	38.491	.000	.142
CellStates * CAIterations * NeighbourhoodRadius	.002	6	.000	2.128	.048	.018
Error	.127	696	.000			
Total	516.320	720				
Corrected Total	.727	719				

Table 16. ANOVA Test for the fitness values achieved by the experiments in set 2 and 3.

Again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test was performed on the neighbourhood radius as it had only two levels. Table 17 shows the results from the post hoc test performed on the number of cell states, and Table 18 shows the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 17, the p-value is less than 0.05 for all interactions, which means the effect of using any number of cell states is significantly different to the effect when using any other number of cell states. Table 18 shows that there is no significant effect on the fitness value when changing between 10 and 25 CA iterations, as its p-value of 0.459 is greater than 0.05. However, there is a significant effect between all other CA iterations due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0090590625*	.00123102990	.000
	4	.0162422375*	.00123102990	.000
3	2	-.0090590625*	.00123102990	.000
	4	.0071831750*	.00123102990	.000
4	2	-.0162422375*	.00123102990	.000
	3	-.0071831750*	.00123102990	.000

Table 17. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f1* with results from sets 2 and 3.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0582136056*	.00142147089	.000
	10	-.0532099222*	.00142147089	.000
	25	-.0521562667*	.00142147089	.000
5	1	.0582136056*	.00142147089	.000
	10	.0050036833*	.00142147089	.000
	25	.0060573389*	.00142147089	.000
10	1	.0532099222*	.00142147089	.000
	5	-.0050036833*	.00142147089	.000
	25	.0010536556	.00142147089	.459
25	1	.0521562667*	.00142147089	.000
	5	-.0060573389*	.00142147089	.000
	10	-.0010536556	.00142147089	.459

Table 18. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f1* with results from sets 2 and 3.

From the results of the three ANOVA tests described in this section, this research concludes that each of the five factors, associated with the GA and CA, significantly affect the fitness values achieved in the GA process where *f1* is the objective function. However, when using the indirect representation, there was no significant difference when using 10 CA iterations compared to using 25 CA iterations. This means that the majority of the variance of effect on the fitness values occurs between 1, 5, and 10 CA iterations when using the indirect representation.

From the results of the visual comparison it was found that altering the factors associated with the GA and CA did have significant effects on the layouts that were produced. The chromosomes generated when using the higher mutation rate of 0.01 produced layouts with a better spread of negative space than the lower mutation rate of one over the chromosome length. However, the chromosome representation had the greatest effect on the visual appearance of the generated layouts, with the indirect representation producing layouts that were neater and more structured than those of the direct representation. The neighbourhood radius also had an effect on the layouts, with the larger radius generally producing more diagonal passageways than those of the lower neighbourhood radius. The visual comparisons also showed that some combinations of CA iterations and numbers of cell states performed better than others, with some combinations producing chromosomes with high fitness and other combinations producing chromosomes with lower fitness. While the majority of chromosomes that were assigned high fitness values produced layouts that were visually similar to the goal layout, the chromosomes that were assigned low fitness values did not.

Analysis of Fitness Function 2

During the visual comparison, it was discovered that the layouts generated from chromosomes that were evolved using fitness function 2 were not visually similar to their goal layout due to a grid size that was too small. Therefore no conclusions can be drawn between the analysis in this section and the visual results. However the analysis of variance in the fitness values has been included for completeness.

The first ANOVA that was performed for *f2* was to examine the impact that changes in the mutation rate, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 1. To do this, data from sets 1A and 1B were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 19 shows the results from a three-way between-subjects ANOVA test with two levels of mutation rate, three levels of cell states, and four levels of CA iterations. All effects were statistically significant. The main effect that this research is interested in is the interaction of the three independent variables. This interaction is, (F (6, 696) = 299.278, p < 0.05, Partial Eta Squared = 0.721).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	1.316[a]	23	.057	801.475	.000	.964[a]
Intercept	555.332	1	555.332	7778418.130	.000	1.000
MutationRate	.178	2	.089	1246.967	.000	.782
CellStates	.328	3	.109	1529.957	.000	.868
CAIterations	.122	1	.122	1713.153	.000	.711
MutationRate * CellStates	.168	6	.028	393.012	.000	.772
MutationRate * CAIterations	.129	2	.064	902.080	.000	.722
CellStates * CAIterations	.263	3	.088	1226.352	.000	.841
MutationRate * CellStates * CAIterations	.128	6	.021	299.278	.000	.721
Error	.050	696	7.139E-005			
Total	556.698	720				
Corrected Total	1.366	719				

Table 19. ANOVA Test for the fitness values achieved by the experiments in set 1A and 1B.

As the number of cell states and the number of CA iterations had a significant effect on the fitness values, two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the mutation rate as it had only two levels, and therefore the significant interaction is between those two levels. To determine where the significant interactions lie, the LSD post hoc test is performed. Table 20 shows the results from the post hoc test performed on the number of cell states, and Table 21 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 20, the p-value is less than 0.05 for all effects, meaning the effect of using any number cell states is significantly different to the effect when using any other number cell states. The results displayed in Table 21 show that the effects of using any number of CA Iterations is significantly different to the effect of using any other number of CA iterations due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0225104833*	.00077132964	.000
	4	-.0158147125*	.00077132964	.000
3	2	-.0225104833*	.00077132964	.000
	4	-.0383251958*	.00077132964	.000
4	2	.0158147125*	.00077132964	.000
	3	.0383251958*	.00077132964	.000

Table 20. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f2* with results from sets 1A and 1B.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	.0203621889*	.00089065475	.000
	10	.0413642722*	.00089065475	.000
	25	.0564442111*	.00089065475	.000
5	1	-.0203621889*	.00089065475	.000
	10	.0210020833*	.00089065475	.000
	25	.0360820222*	.00089065475	.000
10	1	-.0413642722*	.00089065475	.000
	5	-.0210020833*	.00089065475	.000
	25	.0150799389*	.00089065475	.000
25	1	-.0564442111*	.00089065475	.000
	5	-.0360820222*	.00089065475	.000
	10	-.0150799389*	.00089065475	.000

Table 21. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f2* with results from sets 1A and 1B.

The second ANOVA that was performed for *f2* was to examine the impact that changes in the chromosome representation, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 2. To do this, data from sets 1A and 2 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 22 shows the results from a three-way between-subjects ANOVA test with two levels of chromosome representation, three levels of cell states, and four levels of CA iterations. All effects were statistically significant. The interaction effect between the three independent variables is, (F (6, 696) = 353.392, $p < 0.05$, Partial Eta Squared = 0.753).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.749[a]	23	.033	723.029	.000	.960[a]
Intercept	553.597	1	553.597	12298033.764	.000	1.000
CellStates	.096	2	.048	1062.084	.000	.753
CAIterations	.123	3	.041	913.950	.000	.798
Representation	.149	1	.149	3319.719	.000	.827
CellStates * CAIterations	.129	6	.021	476.330	.000	.804
CellStates * Representation	.043	2	.021	474.809	.000	.577
CAIterations * Representation	.113	3	.038	838.659	.000	.783
CellStates * CAIterations * Representation	.095	6	.016	353.392	.000	.753
Error	.031	696	4.502E-005			
Total	554.377	720				
Corrected Total	.780	719				

Table 22. ANOVA Test for the fitness values achieved by the experiments in set 1A and 2.

Once again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the chromosome representation as it had only two levels. Table 23 shows the results from the post hoc test performed on the number of cell states, and Table 24 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 23, the p-value is less than 0.05 for all interactions, which means the effects of using any number of cell states are significantly different to the effects when using any other number of cell states. The same can be seen in Table 24, where the effects of using any number of CA iterations are significantly different to one another due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	-.0091942125*	.00061247487	.000
	4	-.0277103500*	.00061247487	.000
3	2	.0091942125*	.00061247487	.000
	4	-.0185161375*	.00061247487	.000
4	2	.0277103500*	.00061247487	.000
	3	.0185161375*	.00061247487	.000

Table 23. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f2* with results from sets 1A and 2.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0149003667*	.00070722506	.000
	10	.0050346778*	.00070722506	.000
	25	.0217651667*	.00070722506	.000
5	1	.0149003667*	.00070722506	.000
	10	.0199350444*	.00070722506	.000
	25	.0366655333*	.00070722506	.000
10	1	-.0050346778*	.00070722506	.000
	5	-.0199350444*	.00070722506	.000
	25	.0167304889*	.00070722506	.000
25	1	-.0217651667*	.00070722506	.000
	5	-.0366655333*	.00070722506	.000
	10	-.0167304889*	.00070722506	.000

Table 24. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f2* with results from sets 1A and 2.

The third ANOVA that was performed for *f2* was to examine the impact that changes in the neighbourhood radius, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 3. To do this, data from sets 2 and 3 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 25 shows the results from a three-way between-subjects ANOVA test with two levels of neighbourhood radius, three levels of cell states, and four levels of CA iterations. All effects were statistically significant with a p-value of less than 0.05. The interaction effect between the three independent variables is, (F (6, 696) = 4.426, p < 0.05, Partial Eta Squared = 0.037).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.026[a]	23	.001	51.329	.000	.629[a]
Intercept	539.150	1	539.150	24740442.802	.000	1.000
CellStates	.010	2	.005	223.024	.000	.391
CAIterations	.002	3	.001	24.683	.000	.096
NeighbourhoodRadius	.006	1	.006	275.906	.000	.284
CellStates * CAIterations	.005	6	.001	34.800	.000	.231
CellStates * NeighbourhoodRadius	.003	2	.001	65.182	.000	.158
CAIterations * NeighbourhoodRadius	.000	3	.000	6.281	.000	.026
CellStates * CAIterations * NeighbourhoodRadius	.001	6	9.646E-005	4.426	.000	.037
Error	.015	696	2.179E-005			
Total	539.191	720				
Corrected Total	.041	719				

Table 25. ANOVA Test for the fitness values achieved by the experiments in set 2 and 3.

Again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the neighbourhood radius as it had only two levels. Table 26 shows the results from the post hoc test performed on the number of cell states, and Table 27 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 26, the p-value is less than 0.05 for all interactions, which means the effects of using any number of cell states is significantly different to the effects when using any other number of cell states. Table 27 shows that there is no significant effect on the fitness value when changing between 5 and 10 CA iterations, and 10 and 25 CA iterations, but there is a significant effect between all other CA iterations due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	-.0074523583*	.00042614801	.000
	4	-.0080964458*	.00042614801	.000
3	2	.0074523583*	.00042614801	.000
	4	-.0006440875	.00042614801	.131
4	2	.0080964458*	.00042614801	.000
	3	.0006440875	.00042614801	.131

Table 26. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f2* with results from sets 2 and 3.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	.0026088889*	.00049207334	.000
	10	.0031195889*	.00049207334	.000
	25	.0040277333*	.00049207334	.000
5	1	-.0026088889*	.00049207334	.000
	10	.0005107000	.00049207334	.300
	25	.0014188444*	.00049207334	.004
10	1	-.0031195889*	.00049207334	.000
	5	-.0005107000	.00049207334	.300
	25	.0009081444	.00049207334	.065
25	1	-.0040277333*	.00049207334	.000
	5	-.0014188444*	.00049207334	.004
	10	-.0009081444	.00049207334	.065

Table 27. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f2* with results from sets 2 and 3.

From the results of the three ANOVA tests described in this section, this research concludes that each of the five factors, associated with the GA and CA, significantly affect the fitness values achieved in the GA process where *f2* is the objective function. However, when using the indirect representation there was no significant difference when using 5 CA iterations compared to using 10 CA iterations, or between using 10 CA iterations and 25 CA iterations.

Analysis of Fitness Function 3

The first ANOVA that was performed for *f3* was to examine the impact that changes in the mutation rate, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 1. To do this, data from sets 1A and 1B were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 28 shows the results from a three-way between-subjects ANOVA test with two levels of mutation rate, three levels of cell states, and four levels of CA iterations. All effects were statistically significant. The main effect that this research is interested in is the interaction of the three independent variables. This interaction is, (F (6, 696) = 173. 739, $p < 0.05$, Partial Eta Squared = 0.600).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	2.309[a]	23	.100	331.025	.000	.916[a]
Intercept	465.625	1	465.625	1535405.658	.000	1.000
MutationRate	.401	2	.200	660.615	.000	.655
CellStates	.982	3	.327	1079.342	.000	.823
CAIterations	.045	1	.045	147.879	.000	.175
MutationRate * CellStates	.129	6	.021	70.774	.000	.379
MutationRate * CAIterations	.291	2	.145	479.622	.000	.580
CellStates * CAIterations	.146	3	.049	160.035	.000	.408
MutationRate * CellStates * CAIterations	.316	6	.053	173.739	.000	.600
Error	.211	696	.000			
Total	468.145	720				
Corrected Total	2.520	719				

Table 28. ANOVA Test for the fitness values achieved by the experiments in set 1A and 1B.

As the number of cell states and the number of CA iterations had a significant effect on the fitness values, two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the mutation rate as it had only two levels, and therefore the significant interaction is between those two levels. To determine where the significant interactions lie, the LSD post hoc test is performed.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0418823833*	.00158970367	.000
	4	.0554174958*	.00158970367	.000
3	2	-.0418823833*	.00158970367	.000
	4	.0135351125*	.00158970367	.000
4	2	-.0554174958*	.00158970367	.000
	3	-.0135351125*	.00158970367	.000

Table 29. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f3* with results from sets 1A and 1B.

Table 29 shows the results from the post hoc test performed on the number of cell states, and Table 30 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 29, the p-value is less than 0.05 for all effects, meaning the effect of using any number of cell states is significantly different to the effect when using any other number of cell states. The results displayed in Table 30 show that the effects of using any number of CA Iterations is significantly different to the effect of using any other number of CA iterations due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	.0784378000*	.00183563168	.000
	10	.0829825056*	.00183563168	.000
	25	.0921431111*	.00183563168	.000
5	1	-.0784378000*	.00183563168	.000
	10	.0045447056*	.00183563168	.014
	25	.0137053111*	.00183563168	.000
10	1	-.0829825056*	.00183563168	.000
	5	-.0045447056*	.00183563168	.014
	25	.0091606056*	.00183563168	.000
25	1	-.0921431111*	.00183563168	.000
	5	-.0137053111*	.00183563168	.000
	10	-.0091606056*	.00183563168	.000

Table 30. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f3* with results from sets 1A and 1B.

The second ANOVA that was performed for *f3* was to examine the impact that changes in the chromosome representation, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 2. To do this, data from sets 1A and 2 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 31 shows the results from a three-way between-subjects ANOVA test with two levels of chromosome representation, three levels of cell states, and four levels of CA iterations. All effects were statistically significant. The interaction effect between the three independent variables is, (F (6, 696) = 90.543, $p < 0.05$, Partial Eta Squared = 0.438).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	2.459[a]	23	.107	747.458	.000	.961[a]
Intercept	506.945	1	506.945	3544441.069	.000	1.000
CellStates	.046	2	.023	162.549	.000	.318
CAIterations	.097	3	.032	225.424	.000	.493
Representation	1.320	1	1.320	9228.080	.000	.930
CellStates * CAIterations	.126	6	.021	147.382	.000	.560
CellStates * Representation	.036	2	.018	125.986	.000	.266
CAIterations * Representation	.756	3	.252	1760.855	.000	.884
CellStates * CAIterations * Representation	.078	6	.013	90.543	.000	.438
Error	.100	696	.000			
Total	509.503	720				
Corrected Total	2.558	719				

Table 31. ANOVA Test for the fitness values achieved by the experiments in set 1A and 2.

Once again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the chromosome representation as it had only two levels. Table 32 shows the results from the post hoc test performed on the number of cell states, and Table 33 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 32, the p-value is less than 0.05 for all interactions, which means the effects of using any number of cell states are significantly different to the effects when using any other number of cell states. The same can be seen in Table 33, where the effects of using any number of CA iterations are significantly different to one another due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	-.0026754667*	.00109173153	.015
	4	.0155512708*	.00109173153	.000
3	2	.0026754667*	.00109173153	.015
	4	.0182267375*	.00109173153	.000
4	2	-.0155512708*	.00109173153	.000
	3	-.0182267375*	.00109173153	.000

Table 32. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f3* with results from sets 1A and 2.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0254726667*	.00126062298	.000
	10	-.0071374444*	.00126062298	.000
	25	.0051311556*	.00126062298	.000
5	1	.0254726667*	.00126062298	.000
	10	.0183352222*	.00126062298	.000
	25	.0306038222*	.00126062298	.000
10	1	.0071374444*	.00126062298	.000
	5	-.0183352222*	.00126062298	.000
	25	.0122686000*	.00126062298	.000
25	1	-.0051311556*	.00126062298	.000
	5	-.0306038222*	.00126062298	.000
	10	-.0122686000*	.00126062298	.000

Table 33. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f3* with results from sets 1A and 2.

The third ANOVA that was performed for *f3* was to examine the impact that changes in the neighbourhood radius, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 3. To do this, data from sets 2 and 3 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 34 shows the results from a three-way between-subjects ANOVA test with two levels of neighbourhood radius, three levels of cell states, and four levels of CA iterations. With the exception of the interaction effect between number of cell states and number of CA iterations, all other effects were statistically significant with a p-value of less than 0.05. The interaction effect between the three independent variables is, (F (6, 696) = 6.969, p < 0.05, Partial Eta Squared = 0.057).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	1.115[a]	23	.048	211.399	.000	.875[a]
Intercept	554.120	1	554.120	2417094.309	.000	1.000
CellStates	.014	2	.007	29.612	.000	.078
CAIterations	1.061	3	.354	1543.395	.000	.869
NeighbourhoodRadius	.016	1	.016	67.654	.000	.089
CellStates * CAIterations	.003	6	.000	1.830	.091	.016
CellStates * NeighbourhoodRadius	.007	2	.003	14.258	.000	.039
CAIterations * NeighbourhoodRadius	.005	3	.002	7.937	.000	.033
CellStates * CAIterations * NeighbourhoodRadius	.010	6	.002	6.969	.000	.057
Error	.160	696	.000			
Total	555.394	720				
Corrected Total	1.274	719				

Table 34. ANOVA Test for the fitness values achieved by the experiments in set 2 and 3.

Again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the neighbourhood radius as it had only two levels. Table 35 shows the results from the post hoc test performed on the number of cell states, and Table 36 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 35, the p-value is less than 0.05 for all interactions, except between 3 and 4 cell states. Table 36 shows that the effect of using any number of CA iterations is significantly different than the effect when using any other number of CA iterations, due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0075600417*	.00138217912	.000
	4	.0102601417*	.00138217912	.000
3	2	-.0075600417*	.00138217912	.000
	4	.0027001000	.00138217912	.051
4	2	-.0102601417*	.00138217912	.000
	3	-.0027001000	.00138217912	.051

Table 35. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f3* with results from sets 2 and 3.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0943883889*	.00159600297	.000
	10	-.0875975944*	.00159600297	.000
	25	-.0823933500*	.00159600297	.000
5	1	.0943883889*	.00159600297	.000
	10	.0067907944*	.00159600297	.000
	25	.0119950389*	.00159600297	.000
10	1	.0875975944*	.00159600297	.000
	5	-.0067907944*	.00159600297	.000
	25	.0052042444*	.00159600297	.001
25	1	.0823933500*	.00159600297	.000
	5	-.0119950389*	.00159600297	.000
	10	-.0052042444*	.00159600297	.001

Table 36. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f3* with results from sets 2 and 3.

From the results of the three ANOVA tests described in this section, this research concludes that each of the five factors, associated with the GA and CA, significantly affect the fitness values achieved in the GA process where *f3* is the objective function. However, when using the indirect representation, the interaction effect between numbers of cell states and numbers of CA iterations, was not significant. There was also no significant difference to the fitness values when using 3 cell states compared to using 4 cell states, meaning the use of 2 cell states had the most significant effect on the fitness values.

When looking at the layouts during the visual comparison, it was discovered that chromosomes of the indirect representation produced more accurate layouts when they were evolved using a neighbourhood radius of 1 with 4 cell states and 5 CA iterations. Varying these factors greatly affected the visual accuracy of the generated level layouts.

Analysis of Fitness Function 4

The first ANOVA that was performed for *f4* was to examine the impact that changes in the mutation rate, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 1. To do this, data from sets 1A and 1B were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 37 shows the results from a three-way between-subjects ANOVA test with two levels of mutation rate, three levels of cell states, and four levels of CA iterations. All effects were statistically significant. The main effect that this research is interested in is the interaction of the three independent variables. This interaction is, (F (6, 696) = 394.070, $p < 0.05$, Partial Eta Squared = 0.773).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.807[a]	23	.035	1181.127	.000	.975[a]
Intercept	585.288	1	585.288	19712780.782	.000	1.000
MutationRate	.220	2	.110	3700.219	.000	.914
CellStates	.255	3	.085	2867.566	.000	.925
CAIterations	.030	1	.030	1007.011	.000	.591
MutationRate * CellStates	.078	6	.013	436.038	.000	.790
MutationRate * CAIterations	.028	2	.014	472.933	.000	.576
CellStates * CAIterations	.126	3	.042	1409.756	.000	.859
MutationRate * CellStates * CAIterations	.070	6	.012	394.070	.000	.773
Error	.021	696	2.969E-005			
Total	586.115	720				
Corrected Total	.827	719				

Table 37. ANOVA Test for the fitness values achieved by the experiments in set 1A and 1B.

As the number of cell states and the number of CA iterations had a significant effect on the fitness values, two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the mutation rate as it had only two levels, and therefore the significant interaction is between those two levels. To determine where the significant interactions lie, the LSD post hoc test is performed. Table 38 shows the results from the post hoc test performed on the number of cell states, and Table 39 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 38, the p-value is less than 0.05 for all effects except between 2 cell states and 4 cell states. This means the effects of using 2 cell states is not significantly different to the effects when using 4 cell states. The results displayed in Table 39 show that the effects of using any number of CA Iterations is significantly different to the effect of using any other number of CA iterations due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0375176667*	.00049741647	.000
	4	.0009375458	.00049741647	.060
3	2	-.0375176667*	.00049741647	.000
	4	-.0365801208*	.00049741647	.000
4	2	-.0009375458	.00049741647	.060
	3	.0365801208*	.00049741647	.000

Table 38. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f4* with results from sets 1A and 1B.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	.0247243722*	.00057436706	.000
	10	.0383265778*	.00057436706	.000
	25	.0507698333*	.00057436706	.000
5	1	-.0247243722*	.00057436706	.000
	10	.0136022056*	.00057436706	.000
	25	.0260454611*	.00057436706	.000
10	1	-.0383265778*	.00057436706	.000
	5	-.0136022056*	.00057436706	.000
	25	.0124432556*	.00057436706	.000
25	1	-.0507698333*	.00057436706	.000
	5	-.0260454611*	.00057436706	.000
	10	-.0124432556*	.00057436706	.000

Table 39. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f4* with results from sets 1A and 1B.

The second ANOVA that was performed for *f4* was to examine the impact that changes in the chromosome representation, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 2. To do this, data from sets 1A and 2 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 40 shows the results from a three-way between-subjects ANOVA test with two levels of chromosome representation, three levels of cell states, and four levels of CA iterations. All effects were statistically significant. The interaction effect between the three independent variables is, (F (6, 696) = 277.441, $p < 0.05$, Partial Eta Squared = 0.705).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.782[a]	23	.034	885.349	.000	.967[a]
Intercept	589.926	1	589.926	15370467.094	.000	1.000
CellStates	.021	2	.011	275.638	.000	.442
CAIterations	.228	3	.076	1979.708	.000	.895
Representation	.006	1	.006	155.430	.000	.183
CellStates * CAIterations	.039	6	.006	168.781	.000	.593
CellStates * Representation	.058	2	.029	756.787	.000	.685
CAIterations * Representation	.366	3	.122	3175.432	.000	.932
CellStates * CAIterations * Representation	.064	6	.011	277.441	.000	.705
Error	.027	696	3.838E-005			
Total	590.735	720				
Corrected Total	.808	719				

Table 40. ANOVA Test for the fitness values achieved by the experiments in set 1A and 2.

Once again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the chromosome representation as it had only two levels. Table 41 shows the results from the post hoc test performed on the number of cell states, and Table 42 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 41, the p-value is less than 0.05 for all interactions, which means the effects of using any number of cell states are significantly different to the effects when using any other number of cell states. The same can be seen in Table 42, where the effects of using any number of CA iterations are significantly different to one another due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0094841708*	.00056554181	.000
	4	-.0033063333*	.00056554181	.000
3	2	-.0094841708*	.00056554181	.000
	4	-.0127905042*	.00056554181	.000
4	2	.0033063333*	.00056554181	.000
	3	.0127905042*	.00056554181	.000

Table 41. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f4* with results from sets 1A and 2.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0486055444*	.00065303144	.000
	10	-.0330330889*	.00065303144	.000
	25	-.0204494167*	.00065303144	.000
5	1	.0486055444*	.00065303144	.000
	10	.0155724556*	.00065303144	.000
	25	.0281561278*	.00065303144	.000
10	1	.0330330889*	.00065303144	.000
	5	-.0155724556*	.00065303144	.000
	25	.0125836722*	.00065303144	.000
25	1	.0204494167*	.00065303144	.000
	5	-.0281561278*	.00065303144	.000
	10	-.0125836722*	.00065303144	.000

Table 42. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f4* with results from sets 1A and 2.

The third ANOVA that was performed for *f4* was to examine the impact that changes in the neighbourhood radius, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 3. To do this, data from sets 2 and 3 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 43 shows the results from a three-way between-subjects ANOVA test with two levels of neighbourhood radius, three levels of cell states, and four levels of CA iterations. All effects were statistically significant with a p-value of less than 0.05. The interaction effect between the three independent variables is, (F (6, 696) = 23.243, $p < 0.05$, Partial Eta Squared = 0.167).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.627[a]	23	.027	665.021	.000	.956[a]
Intercept	586.963	1	586.963	14322048.832	.000	1.000
CellStates	.015	2	.007	182.458	.000	.344
CAIterations	.540	3	.180	4395.153	.000	.950
NeighbourhoodRadius	.000	1	.000	6.372	.012	.009
CellStates * CAIterations	.005	6	.001	22.049	.000	.160
CellStates * NeighbourhoodRadius	.008	2	.004	92.433	.000	.210
CAIterations * NeighbourhoodRadius	.053	3	.018	427.373	.000	.648
CellStates * CAIterations * NeighbourhoodRadius	.006	6	.001	23.243	.000	.167
Error	.029	696	4.098E-005			
Total	587.619	720				
Corrected Total	.655	719				

Table 43. ANOVA Test for the fitness values achieved by the experiments in set 2 and 3.

Again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the neighbourhood radius as it had only two levels. Table 44 shows the results from the post hoc test performed on the number of cell states, and Table 45 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 44, the p-value is less than 0.05 for all interactions, which means the effects of using any number of cell states is significantly different to the effects when using any other number of cell states. Table 45 shows that there is no significant effect on the fitness value when changing between 10 and 25 CA iterations, but there is a significant effect between all other CA iterations due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0029772417*	.00058440271	.000
	4	.0108065333*	.00058440271	.000
3	2	-.0029772417*	.00058440271	.000
	4	.0078292917*	.00058440271	.000
4	2	-.0108065333*	.00058440271	.000
	3	-.0078292917*	.00058440271	.000

Table 44. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f4* with results from sets 2 and 3.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0658677111*	.00067481012	.000
	10	-.0623449611*	.00067481012	.000
	25	-.0612188944*	.00067481012	.000
5	1	.0658677111*	.00067481012	.000
	10	.0035227500*	.00067481012	.000
	25	.0046488167*	.00067481012	.000
10	1	.0623449611*	.00067481012	.000
	5	-.0035227500*	.00067481012	.000
	25	.0011260667	.00067481012	.096
25	1	.0612188944*	.00067481012	.000
	5	-.0046488167*	.00067481012	.000
	10	-.0011260667	.00067481012	.096

Table 45. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f4* with results from sets 2 and 3.

From the results of the three ANOVA tests described in this section, this research concludes that each of the five factors, associated with the GA and CA, significantly affect the fitness values achieved in the GA process where *f4* is the objective function. However, when using the direct representation there was no significant difference when using 2 cell states compared to using 4 cell states, which means using 3 cell states had the most significant effect on the fitness values. Also, when using the indirect representation, there was no significant difference when using 10 CA iterations compared to using 25 CA iterations, which means that the majority of the variance of effect on the fitness values occurs between 1, 5, and 10 CA iterations.

When looking at the layouts during the visual comparison, it was discovered that chromosomes of the indirect representation that were evolved using 10 and 25 CA iterations generally produced layouts that were more visually similar to the goal layout than chromosomes evolved using less CA iterations. Comparing this observation to the analysis in this section suggests that the fitness values achieved by the GA process peaked when using ten or more CA iterations with the indirect chromosome representation.

Analysis of Fitness Function 5

The first ANOVA that was performed for *f*5 was to examine the impact that changes in the mutation rate, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 1. To do this, data from sets 1A and 1B were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 46 shows the results from a three-way between-subjects ANOVA test with two levels of mutation rate, three levels of cell states, and four levels of CA iterations. All effects were statistically significant. The interaction effect between the three factors is, (F (6, 696) = 147.295, p < 0.05, Partial Eta Squared = 0.559).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.502[a]	23	.022	354.188	.000	.921[a]
Intercept	537.387	1	537.387	8722379.695	.000	1.000
MutationRate	.047	2	.024	384.632	.000	.525
CellStates	.120	3	.040	647.827	.000	.736
CAIterations	.054	1	.054	873.608	.000	.557
MutationRate * CellStates	.058	6	.010	156.205	.000	.574
MutationRate * CAIterations	.043	2	.021	345.105	.000	.498
CellStates * CAIterations	.126	3	.042	682.919	.000	.746
MutationRate * CellStates * CAIterations	.054	6	.009	147.295	.000	.559
Error	.043	696	6.161E-005			
Total	537.932	720				
Corrected Total	.545	719				

Table 46. ANOVA Test for the fitness values achieved by the experiments in set 1A and 1B.

As the number of cell states and the number of CA iterations had a significant effect on the fitness values, two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the mutation rate as it had only two levels, and therefore the significant interaction is between those two levels. To determine where the significant interactions lie, the LSD post hoc test is performed. Table 47 shows the results from the post hoc test performed on the number of cell states, and Table 48 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 47, the p-value is less than 0.05 for all effects, meaning the effect of using any number of cell states is significantly different to the effect of using any other number cell states. The results displayed in Table 48 show that the effects of using any number of CA Iterations is significantly different to the effect of using any other number of CA iterations due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0139188083*	.00071653191	.000
	4	-.0053253667*	.00071653191	.000
3	2	-.0139188083*	.00071653191	.000
	4	-.0192441750*	.00071653191	.000
4	2	.0053253667*	.00071653191	.000
	3	.0192441750*	.00071653191	.000

Table 47. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f5* with results from sets 1A and 1B.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	.0132579833*	.00082737978	.000
	10	.0250551278*	.00082737978	.000
	25	.0344036389*	.00082737978	.000
5	1	-.0132579833*	.00082737978	.000
	10	.0117971444*	.00082737978	.000
	25	.0211456556*	.00082737978	.000
10	1	-.0250551278*	.00082737978	.000
	5	-.0117971444*	.00082737978	.000
	25	.0093485111*	.00082737978	.000
25	1	-.0344036389*	.00082737978	.000
	5	-.0211456556*	.00082737978	.000
	10	-.0093485111*	.00082737978	.000

Table 48. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f5* with results from sets 1A and 1B.

The second ANOVA that was performed for *f*5 was to examine the impact that changes in the chromosome representation, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 2. To do this, data from sets 1A and 2 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 49 shows the results from a three-way between-subjects ANOVA test with two levels of chromosome representation, three levels of cell states, and four levels of CA iterations. Due to a p-value greater than 0.05, using different numbers of cell states did not have a significant impact on the fitness values. All other effects were statistically significant. The interaction effect between the three independent variables is, (F (6, 696) = 115.700, $p < 0.05$, Partial Eta Squared = 0.499).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.451[a]	23	.020	194.825	.000	.866[a]
Intercept	547.254	1	547.254	5436125.430	.000	1.000
CellStates	.000	2	.000	1.146	.319	.003
CAIterations	.172	3	.057	568.023	.000	.710
Representation	.000	1	.000	4.034	.045	.006
CellStates * CAIterations	.038	6	.006	62.768	.000	.351
CellStates * Representation	.053	2	.027	265.271	.000	.433
CAIterations * Representation	.118	3	.039	389.742	.000	.627
CellStates * CAIterations * Representation	.070	6	.012	115.700	.000	.499
Error	.070	696	.000			
Total	547.775	720				
Corrected Total	.521	719				

Table 49. ANOVA Test for the fitness values achieved by the experiments in set 1A and 2.

In this case only the number of CA iterations had a significant effect on the fitness values. Therefore one post hoc test was performed to see where the significant interactions were between pairs of CA iterations. No post hoc test is performed on the chromosome representation as it had only two levels. Table 50 show the results of the post hoc test performed on the number of CA iterations, and as can be seen the p-value is less than 0.05 for all interactions, which means the effect of using any number of CA iterations is significantly different to the effect when using any other number of CA iterations.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0421533667*	.00105761708	.000
	10	-.0270906833*	.00105761708	.000
	25	-.0157345333*	.00105761708	.000
5	1	.0421533667*	.00105761708	.000
	10	.0150626833*	.00105761708	.000
	25	.0264188333*	.00105761708	.000
10	1	.0270906833*	.00105761708	.000
	5	-.0150626833*	.00105761708	.000
	25	.0113561500*	.00105761708	.000
25	1	.0157345333*	.00105761708	.000
	5	-.0264188333*	.00105761708	.000
	10	-.0113561500*	.00105761708	.000

Table 50. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f5* with results from sets 1A and 2.

The third ANOVA that was performed for *f5* was to examine the impact that changes in the neighbourhood radius, number of cell states, and number of CA iterations had on the fitness values, as stated in ANOVA Test 3. To do this, data from sets 2 and 3 were used, where each set contained 12 experiments that were each run 30 times giving a total number of 720 data points to be used in the analysis. Table 51 shows the results from a three-way between-subjects ANOVA test with two levels of neighbourhood radius, three levels of cell states, and four levels of CA iterations. All effects were statistically significant with a p-value of less than 0.05. The interaction effect between the three independent variables is, (F (6, 696) = 11.652, $p < 0.05$, Partial Eta Squared = 0.091).

Dependent Variable: Fitness Value

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	.364[a]	23	.016	129.690	.000	.811[a]
Intercept	539.001	1	539.001	4417756.246	.000	1.000
CellStates	.052	2	.026	211.390	.000	.378
CAIterations	.230	3	.077	627.092	.000	.730
NeighbourhoodRadius	.025	1	.025	201.799	.000	.225
CellStates * CAIterations	.013	6	.002	17.208	.000	.129
CellStates * NeighbourhoodRadius	.007	2	.004	29.576	.000	.078
CAIterations * NeighbourhoodRadius	.030	3	.010	81.572	.000	.260
CellStates * CAIterations * NeighbourhoodRadius	.009	6	.001	11.652	.000	.091
Error	.085	696	.000			
Total	539.450	720				
Corrected Total	.449	719				

Table 51. ANOVA Test for the fitness values achieved by the experiments in set 2 and 3.

Again, the number of cell states and the number of CA iterations had a significant effect on the fitness values. Therefore two post hoc tests were performed to see where the significant interactions were between pairs of cell states, and pairs of CA iterations. No post hoc test is performed on the neighbourhood radius as it had only two levels. Table 52 shows the results from the post hoc test performed on the number of cell states, and Table 53 show the results of the post hoc test performed on the number of CA iterations. As can be seen in Table 52, the p-value is less than 0.05 for all interactions, which means the effects of using any number of cell states is significantly different to the effects when using any other number of cell states. Table 53 shows that there is no significant effect on the fitness value when changing between 10 and 25 CA iterations, but there is a significant effect between all other CA iterations due to a p-value that is below 0.05.

Dependent Variable: Fitness Value

(I) Number of Cell States	(J) Number of Cell States	Mean Difference (I-J)	Std. Error	Sig.
2	3	.0109186792*	.00100833145	.000
	4	.0207229583*	.00100833145	.000
3	2	-.0109186792*	.00100833145	.000
	4	.0098042792*	.00100833145	.000
4	2	-.0207229583*	.00100833145	.000
	3	-.0098042792*	.00100833145	.000

Table 52. LSD Post hoc test results for number of cell states. Post hoc performed as part of ANOVA for *f5* with results from sets 2 and 3.

Dependent Variable: Fitness Value

(I) Number of CA Iterations	(J) Number of CA Iterations	Mean Difference (I-J)	Std. Error	Sig.
1	5	-.0456001000*	.00116432087	.000
	10	-.0384362556*	.00116432087	.000
	25	-.0378433833*	.00116432087	.000
5	1	.0456001000*	.00116432087	.000
	10	.0071638444*	.00116432087	.000
	25	.0077567167*	.00116432087	.000
10	1	.0384362556*	.00116432087	.000
	5	-.0071638444*	.00116432087	.000
	25	.0005928722	.00116432087	.611
25	1	.0378433833*	.00116432087	.000
	5	-.0077567167*	.00116432087	.000
	10	-.0005928722	.00116432087	.611

Table 53. LSD Post hoc test results for number of CA iterations. Post hoc performed as part of ANOVA for *f5* with results from sets 2 and 3.

From the results of the three ANOVA tests described in this section, this research concludes that each of the five factors, associated with the GA and CA, significantly affect the fitness values achieved in the GA process where *f5* is the objective function. However, when analysing the results from sets 1A and 2, there was no significant difference when different numbers of cell states. Also, when using the indirect representation, there was no significant difference when using 10 CA iterations compared to using 25 CA iterations, a discovery that was also made when analysing results from *f4*.

When looking at the layouts during the visual comparison, it was discovered that chromosomes of the indirect representation that were evolved using 5 CA iterations generally produced layouts that were more visually similar to the goal layout than chromosomes evolved using other numbers CA iterations. Comparing this observation to the analysis in this section suggests that the fitness values achieved by the GA process peaked when using 5 CA iterations and then levelled out over 10 and 25 iterations.

4.4 Summary

This chapter viewed the results achieved through the proposed approach starting with a visual analysis of the layouts produced using this approach, followed by analysis of variance tests which details the factors associated with GA and CA which had a significant impact on the results. The visual analysis was performed by comparing generated level layouts to their goal layout against a set of criteria. The majority of these results demonstrated that the chosen chromosome representation had a big impact on the appearance of the generated layouts, with the indirect representation producing neater and more structured layouts than the direct representation. This comparison also demonstrated that layouts which are assigned high ASM values have a stronger visual similarity to their goal layout.

The analysis of variance tests showed that the five factors, associated with GA and CA, which were varied in the experiments that were conducted in this research all had a significant effect on the fitness values achieved by the GA process. However, there were also significant interaction effects between these factors, which mean varying one factor may only have a significant effect on the fitness values when the other factors were set to specific values. The results attained from this study show promise for future research. Possible future research directions based on these results are discussed in the next chapter.

Chapter 5. Conclusion and Future Work

Content creation is an important process in the development of video games, but also one of the most time consuming, which is causing game development companies to spend a large portion of their annual revenue on developers who manually create the content. PCG is the process of algorithmically generating media content which can be used in video games. This is a useful tool to game developers as the process of PCG is much faster than the manual creation of content, reducing development time, and decreasing development costs.

A new approach to the procedural generation of maze-like game level layouts through the use of evolved cellular automata has been introduced in this thesis. The approach uses a genetic algorithm to evolve CA rules which, when applied to a perfect maze configuration, produce level layouts with desired maze-like properties. Most other PCG methods that use evolution employ the evolution to generate the level layout directly. Such approaches are limited as the evolutionary process can be slow. In contrast CAs are fast and simple, so that once rules for generating a desired level style are evolved, many instances of that style can be produced in a short space of time, as was demonstrated in section 4.3.1 where rule tables with high fitness were selected to produce a number of similar layouts. This makes CA ideal for run time content generation, as was used in Johnson et al.'s (2010) approach. However, unlike Johnson's approach which generated cave-like levels using simple, manually designed CA rules, the approach presented in this study uses a GA to automatically find CA rules that are capable of generating level layouts with a number of different styles.

The results achieved from this research demonstrated that CA are capable of generating game level layouts with desired maze-like properties. These results were attained by running a series of experiments, which varied the chromosome representation, GA mutation rate, and various CA properties, with eight different objective functions. The CA properties that were varied include the number of cell states, the number of CA iterations, and the radius of the CA's neighbourhood. During this research it was discovered that the chromosome representation had the largest affect on the visual appearance of the generated level layouts, while all of the varied factors had a significant effect on the fitness values achieved by the GA process.

The approach developed during this study answered the primary research question, "How can rule sets for cellular automata be evolved so as to produce maze-like game level layouts?" with the contributions that are summarized below.

1) Two chromosome representations for cellular automata rule sets were explored to experiment with different CA parameters including number of CA iterations, number of cell states, and size of the neighbourhood radius. This exploration was performed to examine how these parameters affected the generated level layouts. It was discovered that all of these parameters had an effect on the visual appearance of the generated layouts, although the chromosome representation had the greatest impact.

2) The level layouts used in this approach were made up of cells that could be in one of two states, traversable or non-traversable. This meant that the number of CA cell states could only be two. To explore using more than two cell states the idea of "flavours" was used. The principle of this idea divides a single cell state into a set of sub states for the purpose of rule set application. This allowed the use of additional cell states where each additional state mapped to one of the two original states, traversable or non-traversable.

3) In order to evolve the CA rule sets towards producing layouts with desired attributes, attributes from generated layouts had to be extracted for the purpose of evaluation. Due to a lack of literature in this area, a unique approach was developed to extract particular attributes from generated 2D level layouts, using a collection of image analysis techniques.

4) Another important aspect of CA that can greatly impact the results that it produces is its initial configuration. This research used a modified graph traversal algorithm to generate a collection of perfect mazes, which were used as initial configurations, and fed as input into the CA process. This effectively combined cellular automata, a genetic algorithm, and maze generation into a method capable of developing CA rules with the ability to produce maze-like game level layouts with desired properties.

Due to the time constraints that were imposed on this project, its scope was limited to using a weighted aggregation of level attributes in the fitness functions and two chromosome representations. There are some key avenues open for future work that build on the concepts introduced in this study. These are given below.

1) The fitness functions used in this research evaluated chromosomes by applying them to initial configurations to generate a collection of level layouts. Each layout contributed to the chromosomes fitness based on a weighted aggregation of the layouts attributes and their similarity to the desired values, which were determined by the fitness function. However, the level attributes that were evaluated can interact with one another, and therefore there may be other options to evaluating these attributes than a weighted aggregation, such as a multi-objective evaluation using Pareto fronts.

2) During this research it was discovered that the chromosome representation had the largest effect on visual appearance of the generated level layouts, as the indirect representation produced neater and more structured layouts than the direct representation. Therefore experimentation with other chromosome representations that use different encoding schemes, or allow exploration of other CA parameters, could be performed to further research in this area.

To conclude, this research aimed to contribute to the increasing need of PCG techniques by developing an approach to generating game level layouts with maze-like properties through combining GA, CA, and maze generation techniques. This research proved that it is possible to use genetic algorithms to evolve cellular automata rules that are capable of generating maze-like game level layouts when applied to an initial configuration in the form of a perfect maze.

References

Aldous, D. (1990). "The Random Walk Construction of Uniform Spanning Trees and Uniform Labelled Trees." *SIAM Journal on Discrete Mathematics* **3**(4): 450-465.

Ashlock, D. (2010). “Automatic generation of game elements via evolution”. *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*.

Ashlock, D. Lee, C. McGuinness, C. (2011). "Search-Based Procedural Generation of Maze-Like Levels." Computational Intelligence and AI in Games, IEEE Transactions on **3**(3): 260-273.

Ashlock, D. McGuinness, C. Ashlock, W. (2012). “Representation in Evolutionary Computation”. *Advances in Computational Intelligence*. J. Liu, C. Alippi, B. Bouchon-Meunier, G. Greenwood and H. Abbass, Springer Berlin Heidelberg. **7311:** 77-97.

Bäck, T. (1993). “Optimal Mutation Rates in Genetic Search”. *Proceedings of the 5th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers Inc.**:** 2-8.

Basili, V. (1993). “The Experimental Paradigm in Software Engineering”. Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions, Springer-Verlag**:** 3-12.

Blizzard North. (1996). “Diablo”. Cross-platform. Blizzard Entertainment.

Broder, A. (1989). “Generating random spanning trees”. Foundations of Computer Science, 1989., 30th Annual Symposium on.

Buck, J. (2011). “Maze Generation: Growing Tree algorithm”. Retrieved from the buckblogs website http://weblog.jamisbuck.org/2011/1/17/maze-generation-aldous-broder-algorithm

Chopard, B. Droz, M. (1998). “Cellular automata modeling of physical systems”. *University of CAMBRIDGE*.

De Jong, K (1975). “An analysis of the behavior of a class of genetic adaptive systems”, *University of Michigan***:** 266.

Golomb, S., Baumert, L. (1965). "Backtrack Programming". *J. ACM* **12**(4): 516-524.

Grupe, F. Jooste, S. (2004). "Genetic algorithms: A business perspective." *Inf. Manag. Comput. Security* **12**(3): 288-297.

Guðlaugsson, B. (2006). "Procedural Content Generation". *New Technology*.

Hendrikx, M. Meijer, S. Van Der Velden, J. Iosup, A. (2013). "Procedural content generation for games: A survey". *ACM Trans. Multimedia Comput. Commun. Appl.* **9**(1): 1-22.

Holland, J. (1992). "Genetic Algorithms". *Scientific American*. 66-72.

Johnson, L. Yannakakis, G. Togelius, J. (2010). "Cellular automata for real-time generation of infinite cave levels". *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. 1-4.

Koeneke, R. Todd, J. (1994). "Moria", Cross-platform.

Kruskal, J. (1956). "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem." *Proceedings of the American Mathematical Society* **7**(1): 48-50.

Mandelbrot, B. (1982). "The Fractal Geometry of Nature". Amazon.

Mička, P. (2013). "Jarník-Prim algorithm". Retrieved from the Algoritmy website http://en.algoritmy.net/article/43764/Jarnik-Prim-algorithm

Mitchell, M. Crutchfield, J. Das, R. (1996). "Evolving cellular automata with genetic algorithms: A review of recent work." *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96)*.

NetHack Dev Team. (1987). "NetHack". Cross-platform. NetHack Dev Team.

Pedersen, C. Togelius, J. Yannakakis, G. (2009). "Modeling player experience in Super Mario Bros". *Computational Intelligence and Games*, 2009. CIG 2009. IEEE Symposium on.

Perlin, K. Hoffert, E. (1989). "Hypertexture." *SIGGRAPH Comput. Graph.* **23**(3): 253-262.

Piwonska, A. Seredynski, F. (2010). "Discovery by genetic algorithm of cellular automata rules for pattern reconstruction task". *Proceedings of the 9th international conference on Cellular automata for research and industry*. Ascoli Piceno, Italy, Springer-Verlag**:** 198-208.

Pullen, W. (2010). "Maze Classification". Retrieved from the Think Labyrinth website http://www.astrolog.org/labyrnth/algrithm.htm.

Roth, J. Ausbury, L. Fisk, S. Scott, J. (2010). “Free iPADS!!! – Project Report”.

Russ, J. (2006). “The Image Processing Handbook”. 777.

Serra, J. (1983). “Image Analysis and Mathematical Morphology”. Academic Press, Inc.

Shaker, N. Yannakakis G. Togelius, J. (2010). “Towards Automatic Personalized Content Generation for Platform Games”. *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE’10)*.

Togelius, J. Preuss, M. Yannakakis, G. (2010). “Towards multiobjective procedural map generation”. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. Monterey, California, ACM**:** 1-8.

Togelius, J. Yannakakis, G. Stanley, K. Browne, C. (2011). "Search-Based Procedural Content Generation: A Taxonomy and Survey." *Computational Intelligence and AI in Games, IEEE Transactions on* **3**(3): 172-186.

Toy, M. Wichmann, G. (1997). “Rogue”. Cross-platform. Epyx.

Visualization, Interactive Data. (2003). "SpeedTree". Retrieved from the SpeedTree website http://www.speedtree.com/

Von Neumann, J. Burks, W. (1966). “Theory of Self-Reproducing Automata”. *University of Illinois Press*.

Wilson, D. (1996). “Generating random spanning trees more quickly than the cover time”. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. Philadelphia, Pennsylvania, USA, ACM**:** 296-303.

Wolfram, S. (1983). “Statistical mechanics of cellular automata”. *Reviews of Modern Physics*. 601-644.

Yannakakis, G. Togelius, J. (2011). "Experience-Driven Procedural Content Generation." *Affective Computing, IEEE Transactions on* **2**(3): 147-161.