An Investigation into Procedural Dungeon Generation

Alfie Thomasson
40312683

Edinburgh Napier University
School of Computing

Submitted in partial fulfilment of the requirements of Edinburgh Napier University for the Degree of BSc (Hons) Games Development

April 2021

Authorship Declaration

I, Alfie Thomasson, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed: Alfie Thomasson, 21st of April 2021

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

Signed: Alfie Thomasson, 21st of April 2021

The University may not make this dissertation available to others.

Abstract

The concept of dungeons, an explorable space filled with random encounters, monsters, and treasure, has been around for a long time in video games and pop culture. As part of this, procedural generation techniques are commonly employed to provide unique layouts and replayability to them. One such genre that has a big emphasis on procedural generation is on roguelike video games, and developers need to ensure they are up to date with key techniques and methods available for accurate content generation.

This paper discusses the techniques and methods commonly employed in procedural dungeon generation, their suitability for use within video games, and detail key background research into the topic.

We will look into a variety of algorithms used for procedural dungeon generation, and explore the advantages, disadvantages, limitations, and applications of each, as well as identify key elements that might favour one algorithm over another.

Analysis of these techniques will be included alongside methodology of the discussed algorithms, and discussion on improvement of these algorithms. As part of this paper, we will also assess suitability for video games, specifically roguelikes, using data gathered from the implementation of methods.

Keywords: Procedural· Dungeon · Generation. · Evolutionary · Algorithms · Genetic · Cellular · Automata · Generative · c++ · Roguelike · BSP · Agent · Constraint · Games

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to thank my supervisor Babis Koniaris for his continued support, feedback, and patience throughout the project, without this, the project likely would not have been possible.

I would also like to thank all of my friends for supporting me both mentally and emotionally over this experience.

My final thanks goes to my parents for their unwavering support throughout this project and all of my studies.

1 Introduction

1.1 Motivations

Procedural content generation has become increasingly prevalent in recent years, and has been employed in many popular video games for years now. In addition, as video games become more complex the ability to generate levels and content in place of game designers is more valuable than ever, to save on man power, time, and effort.

While procedural dungeon generation makes up a smaller portion of the overall procedural content generation, it is vitally important to certain genres of games. Namely, roguelikes. For these games, the dungeon and layout generation can be critical and are essential to the overall longevity of the game.

Research Question The following research question has been identified for the project:

How effective are various methods of procedural content generation at creating dungeon layouts? What are the differences and similarities between approaches, and how effective and desirable are they?

1.2 Aims and Objectives

The aim of this project is to investigate and present multiple methods and algorithms for generating dungeon layouts and maps. As part of this, the following objectives have been identified:

1. Research and investigate into the algorithms and approaches into procedural generation techniques, with further emphasis on dungeon generation and how procedural generation can enhance this field.

2. Implementation and design of algorithms within a simple software, based on the findings from the research stage. Preparation of outputs for evaluation.

3. Critical evaluation of findings from implementation, and analysis of the results using findings from the work and the literature.

4. Draw conclusions and present findings of the study, along with recommendations of future research.

1.3 Scope

Deliverables As part of this project, the following elements will be submitted.

1. A review and introduction of notable and well used techniques used for dungeon generation, in particular for roguelike video games.

2. A description of how one would implement such techniques, and how they work in practice.

3. A software showcasing varying algorithms in practice, with which we can see how each produces different layouts as well as the scope for customisation.

4. An evaluation of the outputs from previously discuss algorithms, discussing the quantitative differences, and a comparison of the potential advantages, limitations, and applications for different generation types.

Boundaries As part of this project, the boundaries are in the complexity of algorithms as the focus is on comparison and suitability of methods as opposed to the technical aspect. Additionally, a simple graphical interface will be provided, which is an additional boundary to ensure ease of use.

1.4 Constraints

There are a number of hurdles and constraints to overcome for this project. Mainly, the constraint is time. This is due to the amount of time required to gather all the relevant information and implement versions into the sample software.

1.5 Context for the Project

What are Roguelikes? Stemming back to the dungeon crawler game *Rogue*, roguelikes have been an increasingly popular genre of video games since their inception. The term roguelike is hard to pin down as to exactly what it means, but at its core it is a genre that is like the game *Rogue* in many ways, hence the term roguelike. To give an accurate overview on what the genre contains nowadays, we must first examine the history of where it came from.

Pre Rogue Although a lot of people hail Rogue as being the forerunner in these types of games, it was not the first to employ some of the key technologies. Of note, *Pedit5*, released in 1975, and *dnd*, released in 1975, were some of the early examples of dungeon crawling in video games. Both games employed simple gameplay elements inspired by the popular pen and paper RPG Dungeons and Dragons, as well as separated rooms connected via corridors for players to explore.
Subsequently, *Colossal Cave Adventure*, released in 1976, while not employing

some of the traditional roguelike elements such as procedurally generated map layouts, was a game that really set the overall theme and tone of what the genre would become, with adventure, loot, monsters to fight, and permadeath. *Beneath Apple Manor*, released in 1978, pulled a lot of the previous rogue like elements together. It generated random dungeons with multiple floors, and had different rooms, items, monsters to fight, and treasure to claim. As far as the technology goes, this was one of the first to successfully employ procedurally generated content, however all of this was paving the way to the game that would define the genre, *Rogue*.

Rogue Created in 1980 by students Glenn Wichman and Michael Toy, *Rogue* was created to gather elements from previously successful games into something totally new. Much like Colossal Cave Adventure, *Rogue* would have RPG elements such as an inventory, collectable items, random layouts for dungeons and rooms, and more. It was built using UNIX, with help from a programming library known as Curses, which allowed for the drawing of images on screen using a native terminals character set.
The game was released into the public domain as free software, and exploded in popularity. Soon after it became available on many different PC's and was even included in the 1984 BSD (Berkley Standard Distribution) Unix and was available on university computers around the world. It became the definitive most popular game for students everywhere, and marked its place in history.
Following on from Rogue's success, many games tried to step up to emulate and improve upon it. *Hack*, 1980 to 1982, and *Moria*, released in 1983, were two games that emerged from Rogue's success, both trying to replicate Rogue from memory as the original source code wasn't released until 1986.
A game that exploded in popularity, and was worked on and expanded long after its release, was *Nethack*, released in 1987. Building on Rogue's initial design concept, *Nethack* included a massively rich encyclopedia of items, objects, pop culture references, and traditional concepts seen commonly in roguelikes even today. Another notable mention is *Angband*, released in 1992, a game based on the writings of J.R.R Tolkien. *Angband* was a truly massive roguelike for its time, with 100 floors, permadeath, and procedurally generated levels. Later on, in 1994, *Ancient Domains of Mystery*, or *ADOM*, was released, which added a new layer of complexity to the roguelike formula. This game expanded upon the successes of previous games in the genre to create something totally new, and in this game, actions had long lasting effects and may come into effect later down the line. With an overworld of towns and quests to complete, this game took the best elements of previous roguelikes and created something new. Released in 1995, *Linley's Dungeon Crawl* was the next notable release. This game again took all the best elements from previous games and incorporated them, and introduced a new skill and levelling system. It was a few years after Angband's release that the term roguelike as we now know it came to be as in 1993, the USENET newsgroups coined the term (Zapata, 2017). Nowadays, there is a large variety of games that could claim to be roguelike in nature. For a long period, confusion over what exactly defined a roguelike game was still unclear however,

and so in 2008 at a conference in Berlin, an official definition was created known as the *Berlin Interpretation* (*Berlin Interpretation*, n.d.).

1.6 Dungeon Design

Definitions In order to properly define and understand some of the terms used in this paper, presented here are some of the key terms used, with some inspiration for descriptions taken from Viana and Dos Santos (2019).

Room – An area of a level that is connected to other rooms via doorways and corridors.

Corridors - Connects rooms and areas, usually a smaller area with little content, used for transport between rooms.

Dungeon – A collection of smaller areas with a specific objective, an example of this would be to get from one side of the area to the other.

Item - A game element that can be used by the player for mechanical change. An example of this would be a health potion that can be used to regain health, or a weapon that increases player attack.

Barrier – A feature that blocks player access to certain regions, areas, or rooms. These can usually be removed by accomplishing some sort of task elsewhere in the dungeon, such as picking up a key to a locked door.

Differences Throughout the category of dungeons, a variety of different designs, layouts, and definitions of the term exists. Within the topic of procedural generation, four different main methods of generation exist (Dahlskog et al., 2015). The four methods are Connected Rooms, Rooms and Corridors, Labyrinths and Mazes, and Open Area. The first method we will discuss utilises less procedural generation than the other, but is nonetheless important to discuss. Known as Connected Rooms, this method essentially uses pre-designed rooms, and generates a layout for how the rooms will be connected to each other. This approach more closely resembles dungeons in games such as *The Legend of Zelda* (1986), where the layout is static and the player enters and exits rooms, with a clear path between rooms from start to exit, see figure 1.

A modern game that demonstrates this method with procedural generation, is *The Binding of Isaac* (2011). This game uses the same room based approach previously seen, but generates the layout of each level on a per-play basis.

The second type of dungeon layout, is called Rooms and Corridors. A traditional example of this is the game *Rogue* (Toy & Arnold, 1980), that utilised this approach, and was very popular with early roguelike games from the 1980s. Typically, these dungeons are scarce and consist of few rooms connected to each other with non branching linear corridors. The rooms are typically a procedurally

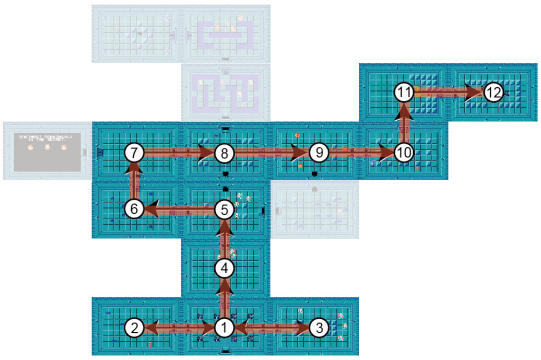


Fig. 1: Layout and path from *The Legend of Zelda* (1986), image from Stout (2012)

Fig. 2: A room layout from *The Binding of Isaac* (2011), utilising the Rooms approach. Image from Wikipedia (n.d.-a)

generated size and shape, and the corridors act as a means to connect them and are playable game space much like the rooms, and viewable on the same screen.

The third approach to dungeon design is Labyrinths and Mazes. These are two separate concepts but are very similar in nature. Games like the *Ultima* Series, starting with *Ultima 1* (1981), use these to great effect. Labyrinths are structures with one route from start to finish, and often include many red herrings and dead ends, as shown in Figure 4. These structures are typically very dense with lots of enclosed spaces, and require 2D matrices to map efficiently. Maps, are similar to labyrinths but have multiple paths from start to finish, and are also often dense but can have less dead ends and more variety in paths.

A fourth and final approach is the Open Area design. These layouts often consist of large open spaces, with walls separating areas off from each other,

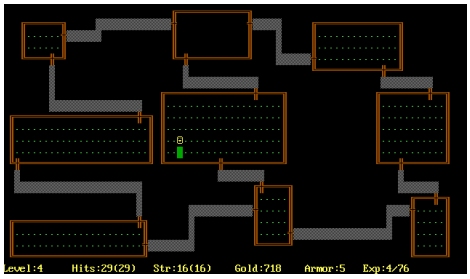


Fig. 3: Layout of a level from *Rogue*, utilising Rooms and Corridors. Image from Wikipedia (n.d.-c)

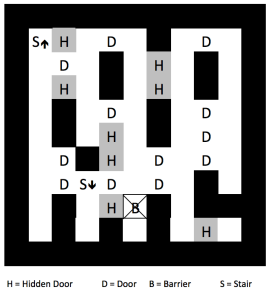


Fig. 4: Layout of a Maze dungeon from *Ultima 1* (1981). Image from Dahlskog et al. (2015)

but in comparison to other dungeon layout the movement around the area is much more important. A good example of this sort of approach would be *Diablo* (1997), which utilises an isometric camera view and this approach to allow for a lot of freedom of player movement.

Dungeon Features

What's In a Dungeon? As part of this study, it is important to define what exactly we mean by certain terms, in reference to dungeons and procedural generation. Dungeons are the main area and play space for a wide variety of game genres. Typically, they will include an entrance or a starting point, a variety of intermediately rooms and areas, and an objective to end the dungeon that will often include challenging a boss monster or similar. It is these rooms in the middle of the beginning and the end that make up the core of the dungeon.
Often times these middle rooms contain some sort of challenging element that will slow the player down, challenging them with mechanical skill or an objective that needs to be completed elsewhere in the dungeon to proceed. These encounters can be monsters to fight, puzzle elements (e.g press buttons in a certain order to move to the next room), or collectible loot such as armour, weaponry,

Fig. 5: A view of the players screen from *Diablo* (1997). Image from Wikipedia (n.d.-b)

or keys that will open areas elsewhere. A good example of traditional dungeon design and a mixture of challenging puzzles is the *Baldur's Gate* series shown in figure 6.

For this project, we will be focusing on the generation of the overall dungeon structure, including a variety of factors and adjustable variables that will change the room layout, and what challenges those rooms might contain.

2 Background

2.1 The Case for Procedural Generation

Throughout the world, procedural generation has widely become used for many different kinds of programs and can be used in multiple fields of work, including music, animation, simulations, and of course video games. It is this field of video games that it has exploded in, and is very popular nowadays as often it can enthrall users in the power to create whole levels or even worlds from scratch, and can create very efficient and varied levels for games (Bermejo, 2018). As Wainer et al. (2010) says, in order to maintain player interest and provide continuity, new content and scenarios must be added to the virtual world. Procedural generation provides this, generating new and often unique content on each play through. In fact, there are many popular games nowadays in which procedural generation is a core element to what makes the game successful. One such game is No Mans Sky (Hello Games, 2016), which is a game built entirely around exploring a universe of unique procedural generated planets, each potentially differing from each other in terms of landscape, wildlife, and a large variety of other features.

Roguelike games heavily incorporate procedural generation into their gameplay, and the quality of the generated content play a vital role. As is the study of this paper, often the most pivotal element of procedural generation in these games is the dungeon generation (King, 2015). By itself, procedurally generating dungeons can produce interesting and varied results such as those seen on

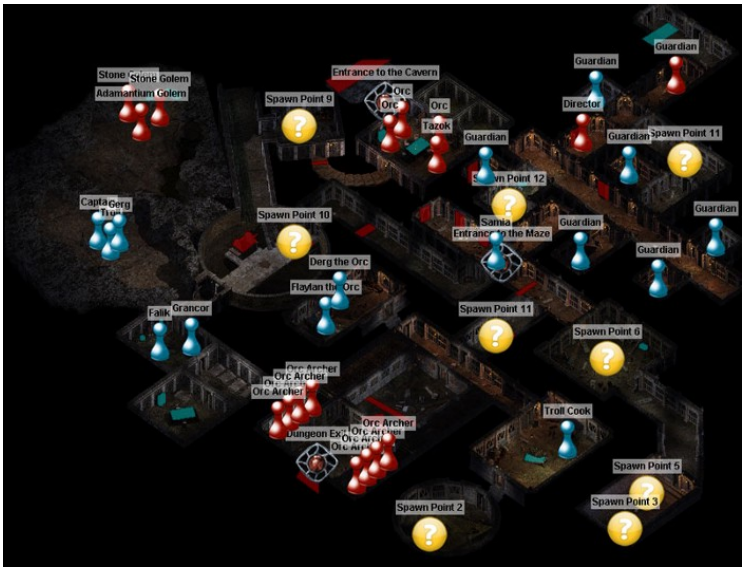


Fig. 6: Dungeon layout taken from the game *Baldurs Gate 2* (Pepe, 2020)

donjon; Random Dungeon Generator (Copyright 2009-2020) and those seen on the original Rogue. These types of generators tend to generate content such as room sizes, connecting halls, monster locations, trap locations, and treasures, and while they may use constraints to ensure it is not totally random, often the results can be too varied or inconsistent. Inspired by games such as *Spelunky*, created by Mossmouth Llc, the popular roguelike game *Dead Cells*, created by Motion Twin, uses a variant of this, utilising chunks of pre-built area to create its levels (Benard, 2017). This approach uses procedural generation to piece together puzzle piece-esque tiles of the level, each with a set of connections and other tiles that can connect to it. This is a slightly alternative approach to procedurally generating dungeons, but worked well, at least for *Dead Cells*, as it allowed for the game to capture the feeling of each run being unique and for the player to rely on skill rather than learning the map, but allowed them to ensure the levels would be consistent by controlling the tilesets that were used to generate the levels.

On the subject of procedural generation for dungeons, Togelius et al. (2016) defines the process as the generation of the overall topology, geometry, and gameplay-related objects of a level. With this in mind, they propose a typical dungeon generation method consists of three elements:

- A simplified, abstract representation of the dungeon, providing a simple overview of the structure.
- A method of constructing the representational model.
- A method to create the geometry of a dungeon from its representational model.

In the book by Togelius et al. (2016), they propose a variety of methods to generate dungeons. In the rest of the background information section, we will discuss each of the methods presented in the context of dungeon generation, in addition to search-based methods using evolutionary algorithms, which will be discussed in more detail in Section 2.7.

2.2 Dungeon Taxonomy

As has been the case with procedural content generation since it's inception, there are multiple different ways and constructors with which to generate content. This is true with dungeon generation as well. Here we outline the general taxonomy of procedural content generation outlined by Togelius et al. (2016) with simplified headings and descriptions to improve readability within our project area (Viana & Dos Santos, 2019).

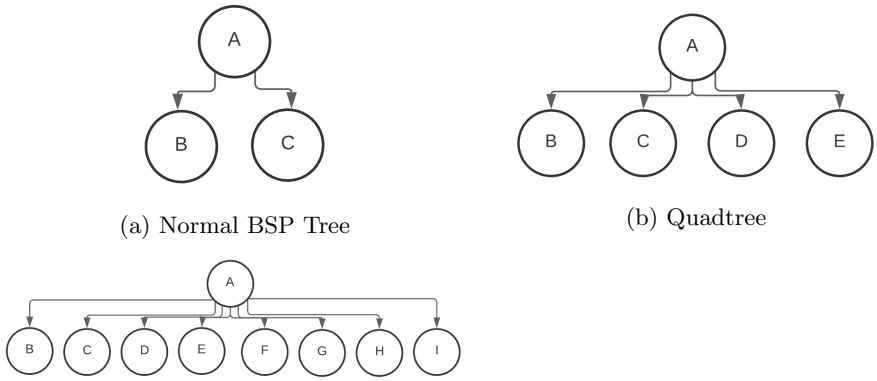
1. Content Need - This defines whether or not the generated content is necessary for the game to function, or is optional. For example, this might be levels in a game which are necessary, or multiple weapon types which typically are optional.
2. Generation Time - This defines when the content is generated. There is an offline approach, where the content is generated before gameplay begins, and the online approach, where content is generated as the game is running.
3. Generation Control - This defines how the content is generated and the degree of control over the creation. For this there is the approach of random seeds to generate content, or the approach of using a set of constraints to allow more control.
4. Generality - This defines the target audience for the content. One such approach is generic, where content is targeted for several different players, and adaptive, where content is generated for a specific player.
5. Random Choice - This defines how decisions are made when content is being generated. There is deterministic approaches and stochastic approaches, where content is generated the same given the same set of parameters, or new content for the same parameters respectively.
6. Generation Method - This defines how the actual content generation is performed. The constructive option generates in one go, and the generate-and-test option switches between generating content and testing it for consistency.
7. Content Authorship - This defines how much the designer can interfere with the generation of the content. Automatic Generation is where the computer generates all of the content with no intervention, whereas mixed-initiative is where the designer has a degree of control over the content generation and is aided by the computer, rather than the computer generating everything.

The algorithms discussed in this paper all have an element from each of these categories. A generated dungeon will be necessary the vast majority of the time as it provides a level on which the game is played on, and so is necessary for

the content need section. With regards to generation time, the dungeons could potentially be offline or online, but will typically be able to be generated at run time and so are usually online approaches. The exception to this is with evolutionary algorithms, which will need time to run and so it is considered an offline approach. Within generation control, cellular automata, constraint based, and agent based utilise the random seed approach as on each new generation they will generate a new result using the new seed. For the others, spatial partitioning, and evolutionary algorithms, they use a defined set of parameters to allow for more control over the process. For the generality section, all approaches are generic as they are not targeted for a specific player. The exception to this may be evolutionary algorithms, as the approach outlined using evolutionary cellular automata could be targeted towards a specific player. For the random choice, everything noted here is deterministic. With generation method, everything is a constructive algorithm except for evolutionary algorithms, which by their nature use the generate-and-test option as they are constantly testing outputs for their fitness value. Finally, for content authorship, everything discussed is automatic generation, although it is important to note that the evolutionary algorithms do have an element of mixed-initiative where the user provides the goal input to achieve.

2.3 Space Partitioning

The first method proposed for dungeon generation is Space Partitioning, a constructive method that allows for content generation at run-time. Binary space partioning is the most popular method for this, allowing for generation of a geometric data structure, which is obtained by using a recursive partitioning method (Toth, 2005). This technique originally arose within the field of computer graphics, in order to gain spatial information about objects within a scene, around the time of 1980 (Fuchs et al., 1980). With this method, the level space is divided into a series of points, and is so that each point lies within a cell on the grid. Traditional space partitioning algorithms often recursively subdivide these cells using the same algorithm each time to create the finished outputs. This method allows the partitions to be arranged in what is called a space partitioning tree, which can be useful for efficient graphical calculations, ray-casting, collision detection, and more, as it allows for querying specific points within the space (Figueiredo et al., 2002). Binary Space Partitioning can be used to subdivide 2D and 3D spaces, and multiple types of spatial partitioning tree exist and may be applicable to different environments. Of note are BSP trees, Quadtrees, and Octree's. The differences between these are shown in figure 7. A typical BSP tree has two children from the root node, and subdivides a 2D space into two subsections. A Quadtree is similar to this, and has four children from the root node. This tree divides a 2D space into four sections, which may have equal or varying size. An Octree can be used for 3D spaces, and would divide a 2D face into four sections, and then along the Z axis additionally into four, to create eight subsections within a 3D environment.



(a) Normal BSP Tree

(b) Quadtree

(c) Octree

Fig. 7: Different examples of space partitioning trees.

The implementation and details on the BSP algorithms themselves can be found in section 18.

2.4 Agent-Based

Another approach to dungeon generation is to use a an agent based approach, using an AI agent of varying complexity to carve out rooms and corridors in our dungeon. This method is constructive, allowing for fast generation at runtime if required. This approach is often more chaotic than the outputs generated by Spatial Partitioning in section 2.3. Due to the nature of using an agent, this method is often far more unpredictable than other methods, and depending on the stochastic nature of the agent, may sometimes generate unusual or potentially unplayable outputs, such as overlapping rooms stuck in the corner of the map and not using the full area Togelius et al. (2016). While this can be a downside to the method, if the agent can be fine tuned and the erratic behaviour stabilised to act in a more rational way, the levels can begin to feel very organic and even planned (Blomberg et al., 2018). Additional information on the implementation of this can be found in section 3.3.

2.5 Cellular Automata

Cellular Automata is typically defined by a grid of cells, each with different states they can be in. Cells start with a certain value, and mathematical functions are applied to cells to potentially change the state of cells and their neighbours, this is typically a rule that is defined before the operation begins. In order to understand Cellular Automata, we must first understand the four classes that make up the different methods of generation.

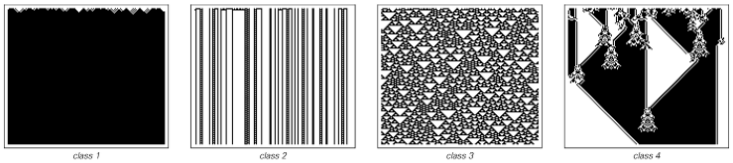


Fig. 8: The four classes that Cellular Automata results can be classified as (Retrieved from Wolfram (2002))

In Class 1, the behaviour of the cells is very simplistic and they do not evolve much from the initial state. Class 2 does have some randomness in its behaviour, but the cells tend to evolve into simple structures that repeat indefinitely once they are stable. In Class 3, the cell behaviour gets more complicated. However it is very chaotic, and can be almost completely random in what is generated. As a result, what is generated at the end is often nonsensical as the emerging cell structures are merged and destroyed by each other. Class 4, is typically the most interesting and the most relevant to this project. The structures created from this include a mixture of randomness and order in evolution, and complex patterns can evolve as a result. Local and individual cell structures are often quite simple, but can interact with each other in interesting ways.

One particularly famous example of Cellular Automata is Conway's Game of Life, a class 4 cellular automata algorithm. The Game of Life presents two cell states, alive and dead, and follows a series of rules (Bellos, 2014).

1. An alive cell with zero or one live neighbours dies – Under-population
2. An alive cell with two or three live neighbours will remain alive – Lives on
3. An alive cell with four or more neighbours dies - Overpopulation
4. A dead cell with exactly three live neighbours becomes alive, in all other cases a dead cell stays dead - Reproduction

Cellular Automata has been explored in detail to develop interesting and practical layouts for cave structures and rooms. Johnson et al. (2010) used CA to create levels for games that they described as "Cave-Like". For this, they used a moore neighbourhood (8 neighbours per cell) with two states, wall and floor. Each cell would be initialised randomly, and a cell would be set to be a wall if at least 5 of its neighbours were also a wall, else the cell would be a floor cell. This presented a simplistic approach to cave generation that would be extremely fast to process during run time and could form separate sections of an area that could be connected for a seemingly infinite game level.

2.6 Grammar-Based

Generative Grammars were originally used to describe structures within languages. It follows the format of structures such as phrases, sentences, or similar,

being modelled by using a set of rules that describe how exactly these larger structures are created from smaller ones, using individual words or characters as symbols. They are called generative grammars as the way they describe language structures, also includes descriptions on how best to generate them. We can apply this technique to other domains, in particular dungeon generation.

One of the first examples of using generative grammars to produce dungeon layouts, was that done by Adams (2002). In his paper, he uses grammars to generate levels for a first person shooter game, which have a similar layout to traditional dungeons that the levels he created could qualify as such. He uses graph grammar rules to create graphs that describe the levels overall topology, although it doesn't include details such as room sizes. This could be an advantage however, as having a high level representation of a level could allow for parameters such as level size, difficulty, etc, to be controlled through other methods to match specific scenarios. Adam's work has limits however, his grammar rules are hard coded and inflexible. However it still shows the importance of mapping dungeon generation to gameplay elements.

Dormans (2010) was the first to introduce gameplay based controls into generative grammars, using a mission grammar. He used generative grammars to create dungeons for adventure games, and using graph grammars, generated missions first in the form of a graph of tasks the player had to complete. From this, the mission was abstracted to a tree of nodes and edges and then subsequently used to generate a game space and level. While he did successfully implement this, it did still have the drawback of a lack of real control parameters, which is something required by most game designers as graph and shape grammars are far from intuitive.

Expanding on the work of others, Van Linden et al. (2013) proposed using gameplay grammars to create dungeons. With this, designers express design constraints using gameplay vocabulary, typically consisting of actions players are able to perform in game. Using these constraints created by designers, a generative graph grammar is created which is called a gameplay grammar. This is then used to create layouts for levels, based on the actions the player is to perform. They also proposed methods to map this graph into the game space and to generate room and corridor geometry. This method was aiming to provide a more generalised approach to gameplay controlled grammars, providing designers with tools to create grammar graphs of player actions which could then be turned into levels. Although the methodology and implementation of grammars will not be covered within this paper, it is a big field and can be highly utilised in a various fields to great success.

2.7 Evolutionary Algorithms

Evolutionary algorithms (EAs) have become a very important and influential part of procedural content generation. There are many different subsections and variants of evolutionary algorithms and are widely used in all types of computing. Charles Darwin, one of the naturalists best known for his contributions to evolutionary science, once said "This preservation of favourable variations and

the destruction of injurious variations, I call Natural Selection, or the Survival of the Fittest." (Darwin, 1869):

The roots of evolutionary algorithms and computation can be traced back to a combination of machine learning, and natural selection being used as an inspiration (Coello, 2005). How this works in nature, is that as generations of people are born, reproduce, and die, the strongest and most fitting traits of each person are passed down to their offspring. This ensures that as the generations go by, the species will adapt and the weak genetic traits are not passed down. These traits are controlled by different chromosomes in the body, which are made up of genes. It is these genes that are passed on during sexual reproduction and recombined, during a process called crossover.

Within Evolutionary Algorithms, there are 3 main subsections that were each developed and created independently from each other, and are each suited to different areas of evolutionary computation. These 3 subsections are Evolution Strategies (ES), Evolutionary Programming (EP), and Genetic Algorithms (GA). There is also Genetic Programming which could be considered another subsection, although this could also be seen as a part of genetic algorithms. Within the scope of procedural dungeon generation, genetic algorithms are the most important subsection of EA's to note, and will be discussed further within the rest of this paper.

Genetic Algorithms Evolutionary Algorithms have many subsections and sub algorithms that have come about as a result of them. Initially conceptualised in the 1960's, evolutionary algorithms have a deep grounding and focus on natural selection, reproduction, and evolution of chromosomes.

One such application of Genetic Algorithms was taken by Ashlock et al. (2011). They used a GA with game levels as chromosomes, and a fitness function which utilised checkpoint placement among other methods, to evaluate fitness and suitability for created level layouts. This proved to create successful layouts and had a lot of control during the creation process, however it took up to 20 minutes at a time to run, and so, was suitable for generating levels during development but not during run time.

Another application of genetic algorithm application to create levels was undertaken by Hartsook et al. (2011). They took an approach of generating levels based on a storyline of events that were essential and non essential throughout the game. This approach allowed the game designer to create a series of events beforehand and have a suitable dungeon layout created for them, represented in a tree structure using Island nodes and Bridge nodes, representing essential locations to the storyline and non essential respectively. This method of generation allowed for a lot of control over bridge length, uniqueness, and the number compared to island nodes.

Pech et al. (2016) utilised a genetic algorithm in tandem with cellular automata to generate outputs similar to a goal input. What this meant was that they could create a simple level via other means, set it as the goal input, and the algorithm would attempt to break down that level into individual elements and

create other levels that share similar qualities with the input. This approach is covered in more detail in section 3.5, as a showcase of how evolutionary algorithms can be used to generate dungeon outputs.

3 Methodology

In this section, we will discuss different methods of dungeon generation and the theory behind how they work. In each section, a description of how each algorithm is used will be presented, and if an implementation within the software has been created, this will also be discussed here.

The Software As part of this project, a software has been created that will demonstrate some of the procedural generation techniques discussed here. This software has support for cellular automata techniques, binary space partitioning, agent based, as well as a constraint based implementation, which will all be discussed here, as well as additional techniques which were not implemented.

3.1 Cellular Automata

What is Cellular Automata? Cellular Automata is a type of cave and dungeon generation technique, that uses a map of cells to produce outputs that may look similar to figure 11. These outputs can be an example of the Mazes and Labryinths type of dungeon, as presented in section 1.6.

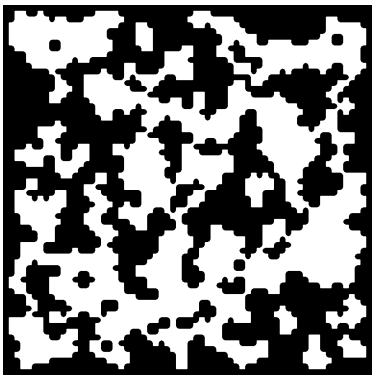


Fig. 9: An example CA output (Kun, 2017)

Firstly, in order to understand cellular automata, we must first understand what elements make up a map. A typical CA model consists of a grid of "Cells", which make up the individual elements.

– A cell can be in one of two states, dead or alive, or represented as 0 and 1.
– A cell has a neighbourhood of adjacent cells, meaning each cell can have up to 8 neighbours.

These neighbourhoods that cells can have can be categorized into two common patterns, the Moore neighbourhood and the von Neumann neighbourhood, although more exist these are the most commonly referenced.

The Moore neighbourhood is categorised by having neighbours in all directions of the cell, completely surrounding it in all directions. This is shown by the red cells surrounding the center in the left image of figure 10. In addition, this neighbourhood can sometimes span two layers, shown by the lighter pink cells (Shaker et al., 2016).

The von Neumann neighbourhood consists of the four neighbouring cells in the cardinal directions, north, east, south, and west. An example of this is shown in the right hand side of figure 10.

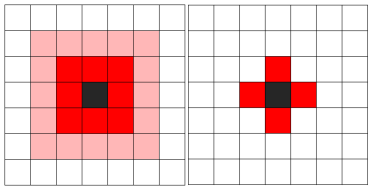


Fig. 10: Left: A Moore Neighbourhood. Right: A von Neumann neighbourhood.

The CA process first begins by initializing a grid of cells. To initialise the cells, by randomly determining if a cell will start alive or dead. This is not always a 50% chance, and may be higher or lower depending on parameters. An example grid a CA algorithm may start with is shown below, with each cell being either dead (0), or alive (1).

Following on from this, the next step for CA is to begin the main generation process, where each cell is checked to determine if it should remain its current value, or switch. An entire CA generation process may be multiple cycles, and this step is typically repeated multiple times. This is done by comparing its neighbours values, and calculating what value the cell should become based on its neighbours states. Each cell has its value calculated, and once each cell has been checked, this generation step is finished and the cycle can repeat. An example cell neighbourhood is shown in figure 11.

In each iteration, a cell can have its value changed depending on the values of each of its neighbours in the previous cycle. Of note here, is how many neighbours are alive and how many are dead. To create the cave and dungeon type structures we follow the principle of Conway's game of life, which is outlined in section 2.5. However, there a few differences which will allow us to generate our structures. Notably, we can adjust our rules and values to produce more accurate representations of caves and dungeon areas. We need two values here, the death

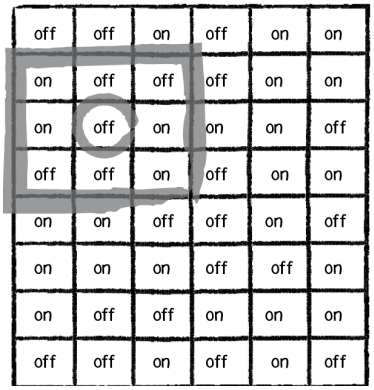


Fig. 11: An example inital CA layout (Shiffman, n.d.)

Limit, and the birth Limit. These represent how many neighbouring cells are needed to change each cells value. If in an iteration, a cell is alive, then it checks how many neighbours are alive, and if it is less than the death limit then the current cell is also killed. If a cell is dead, then it checks how many neighbours are alive, and if its more than the birth limit, the cell is born.

As an example of how a typical CA iteration would appear within code, presented here is a pseudo code example.

Algorithm 1: Cellular Automata Dungeon Generation

```
   Result: A map generated using cellular automata
 1 Initialise grid of cells using alive chance;
 2 for each cell do
 3 |   count number of alive neighbours as Nb;
 4 |   if cell is alive then
 5 |   |   if Nb is less than death limit then
 6 |   |   |   set cell to dead;
 7 |   |   else
 8 |   |   |   set cell to alive
 9 |   |   end
10 |   else
11 |   |   if Nb is bigger than birth limit then
12 |   |   |   set cell to alive;
13 |   |   else
14 |   |   |   set cell to dead;
15 |   |   end
16 |   end
17 end
18 if iterations are not complete then
19 |   return to step 2
20 end
```

Implementation In the software used to represent dungeon generation algorithms, cellular automata is included with a number of variables to adjust how the outputs will generate. This implementation is based off Conway's game of life, discussed in section 2.5, but alters the rules somewhat to allow cave like levels to be generated, which we can use for dungeons. The software can produce outputs based on the following variables, which are explained below:

- Alive Chance - This represents the percentage chance that a cell will start alive at the beginning of the CA process.
- Death Limit - The number of neighbouring cells that must be alive in order for a cell to keep on living.
- Birth Limit - The number of neighbouring cells that must be dead for a cell to stay dead, and not change to alive.
- Iterations - This is the number of iterations that the CA cycle will run for.

As examples of what these parameters can do, the following outputs were generated. For the purposes of these figures, white cells represent traversable space and cells that end up alive, and the black cells represent walls and cells that end up dead. Additionally, standard parameters were set in order to ensure consistency between the generations. These were as follows:

- Map Size - 50 x 50
- Alive Chance - 42%
- Death Limit - 4
- Birth Limit - 4
- Iterations - 2

Using these standard parameters, the following output was produced, shown in figure 12, which will act as a control for looking at how altering the parameter can change the output of the CA algorithm.

In figure 13 we can see the development of a CA dungeon map over three iterations, and using the default settings. We can see how the final result came to be, with the cells applying the rules to them on each iteration to determine their new state, and ending with a result that resembles a cave and could be used for a playable dungeon space.

In figure 14, we can see a comparison of how a low initial alive chance compares to a higher value. We can clearly see that this leads to less cells starting alive versus a higher value, as we expected. The outputs using the default parameters also reflect this very accurately, with the lower alive chance leading to a small amount of alive cells after the iterations, and these small pockets of space. In comparison, a higher alive chance leads to a much more open map, with almost platform like cell clusters forming.

Looking at the death limit, we can draw a few conclusions as to how the death limit affects the output of the dungeon. The lower death limit, shown in figure 15a, leads to an output that is somewhat similar to the control output, with a cave like structure being formed. One thing to note here however, is that there is a large amount more cells alive here than in the control, due to the lower

Fig. 12: A CA output using the default parameters.

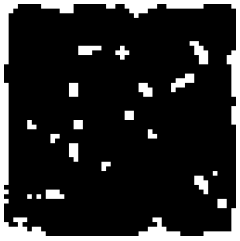
Fig. 13: A CA output progressing over three iterations.

death limit allowing cells to survive for longer with less alive neighbours. With a higher death limit of 6, shown in figure 15b, we can see that the output is nothing like the lower death limit and control outputs. Almost every cells has been killed and only a small amount remain. Here, the high death limit means that cells require a high number of neighbours that are also alive in order to keep surviving, which in turn leads to this output and almost every cell will be dead if further iterations were run.

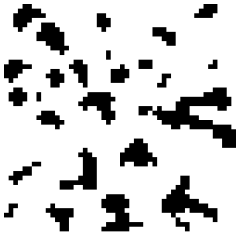
For the birth limit, a lower value leads to the interesting output shown in figure 16a. We can see that a lower birth limit leads to this, as the dead cells are changed to alive a lot more frequently due to the lower birth limit. This is a stark contrast to figure 15b with the high death limit, as in this case most cells end up alive as opposed to dead. A higher birth limit leads to the output shown in figure 16b. Here we can see that a lot less dead cells are changed to alive, and is still far away from our control sample, although is more resembling the control sample than the lower birth limit in figure 16a.

(a) A CA initialisation with alive chance of 30%.

(b) A CA initialisation with alive chance of 60%.



(c) A CA output with alive chance of 30%.



(d) A CA output with alive chance of 60%.

Fig. 14: CA Outputs with lower and higher initial alive chances.

3.2 Spatial Partitioning

One method to generate dungeons is spatial partitioning. This method may produce a result that will be in the rooms and corridors category of dungeons, as presented in section 1.6. This method recursively splits the dungeon space up into boxes, and then fills these boxes with a room. Although this method has multiple ways of subdividing the rooms, the one we will be focusing on in this paper will be binary space partitioning.

This method uses a tree-like node structure to track its room division. To begin, a parent node is created that is the size of the map. Then, for each step of the process, each node on the end of the chain creates two children from it, which is the division of rooms.

The room division works like this; firstly, a random direction is picked, vertical or horizontal. Then, each nodes space is sliced along this axis. The two resulting areas become new nodes, and from here the process is repeated. With each of these two new nodes and new spaces, they are cut into two new spaces and child nodes created to represent these.

Figure 17 is a series of figures detailing the BSP process of dungeon generation. At each step, the current map state is shown and the node tree attached.

Once the subdivisions of the BSP tree have been completed, the next step is to generate rooms within each section. To do this, we generate a room of random x and y size within the space, typically with upper and lower boundaries to stop rooms getting too small or too large. This can be seen in figure 16c.

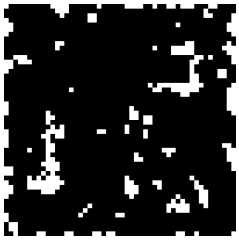
The pseudocode implementation for BSP dungeon is shown below.

(a) A CA output with death limit of 2.

(b) A CA output with death limit of 6

Fig. 15: CA Outputs with lower and higher death limits.

(a) A CA output with birth limit of 2.



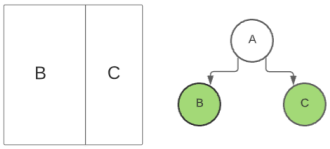
(b) A CA output with birth limit of 6

Fig. 16: CA Outputs with lower and higher birth limits.

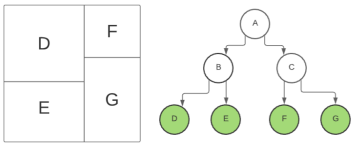
Algorithm 2: BSP Dungeon Generation

Result: A map generated using space partitioning.

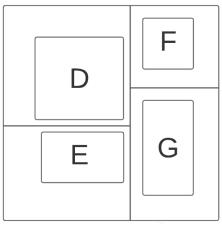
```
1  initialise minimum split value MinSV;
2  initialise maximum split value MaxSV;
3  initialise root node Nd;
4  for each leaf Lf of Nd do
5  |  generate random value Rd between MinSV and MaxSV;
6  |  generate randomly horizontal or vertical;
7  |  split Lf across horizontal or vertical axis at Rd value;
8  |  create two children from current leaf node with the spaces created;
9  end
10 if maximum depth is not reached then
11 |  go to line 4;
12 end
13 for each leaf Lf of Nd do
14 |  create room of random size within space;
15 end
16 for each pair of nodes do
17 |  create a corridor between the two nodes centers;
18 end
```



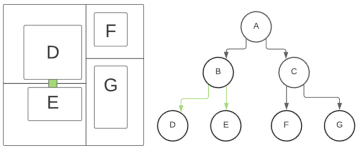
(a) Room is split into two sub divisions.



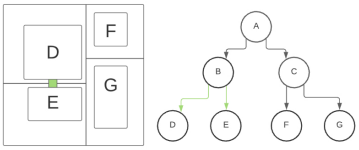
(b) Room is split further into four sub divisions total.



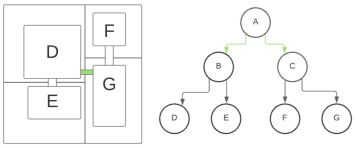
(c) Each section has a room generated within it, of random size.



(d) A corridor is created between the first pair of nodes.



(e) A corridor is created between the second pair of nodes.



(f) A corridor is created between the final pair of parent nodes, and the core of the dungeon is complete.

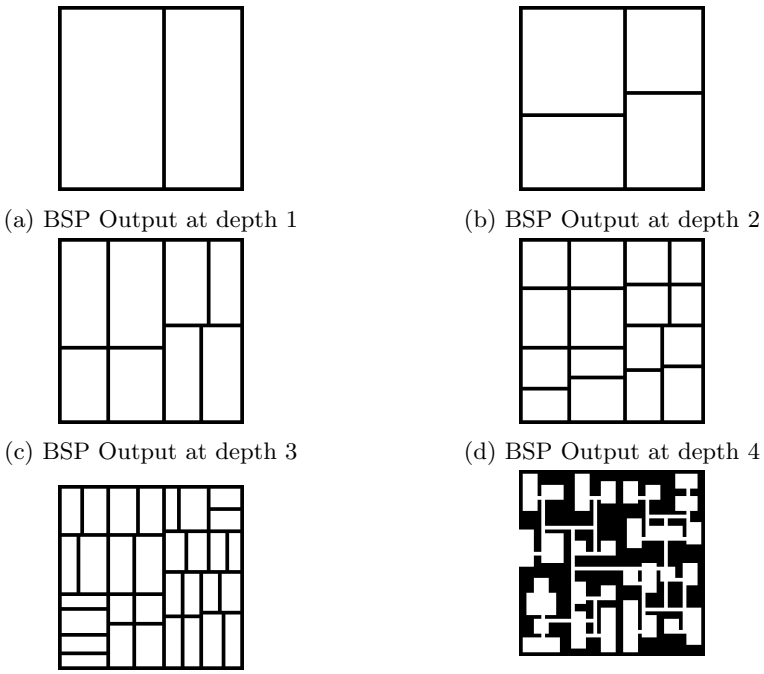
Fig. 17: The BSP process of dungeon generation, including room generation, corridors between rooms, and the node tree generated alongside the dungeon.

Implementation In the software, spatial partitioning has been implemented based on Roguebasin's article (*Basic BSP Dungeon generation*, n.d.). There are multiple adjustable variables to influence the output of the algorithm, and these are explained below.

- Depth - This is how deep the BSP tree will go, and how many rooms the final result will end up with.
- Minimum Split Value - This is the minimum value that the room division can occur on. For example, if this was 40, then when each room is split in half then it will split on a random value between 40% and the maximum split value.
- Maximum Split Value - This is the maximum value that the room division can occur on. For example, if this was 60, then when each room is split in half then it will split on a random value between the minimum split value and 60%.

The following outputs were generated using the software, similarly to the cellular automata section, the black cells represent impassable objects such as walls, and the white cells represent traversable space. For the purposes of visibility, rooms smaller than 2 cells in both dimensions are scaled up to 2 cells in that dimension. The following parameters were set to use unless otherwise stated:

- Size - 100 x 100.
- Depth - 5,
- Minimum Split Value - 40.
- Maximum Split Value - 60.



(a) BSP Output at depth 1

(b) BSP Output at depth 2

(c) BSP Output at depth 3

(d) BSP Output at depth 4

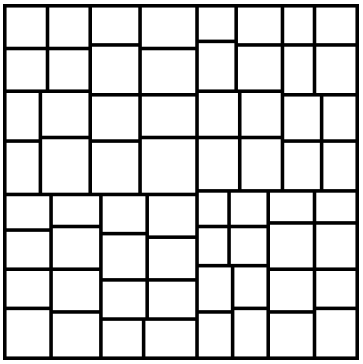
(e) BSP Output at depth 5

(f) Final BSP Dungeon with rooms and corridors.

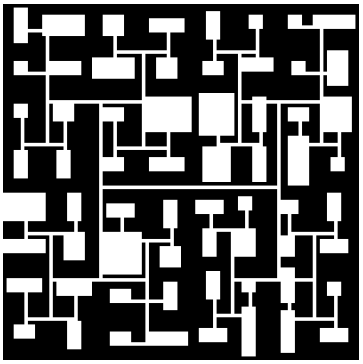
Fig. 18: The development of a BSP Dungeon.

In figure 18 we can see the development of a BSP dungeon using the software. From this, we can clearly see the process of how the dungeon develops, with the subdivisions breaking down the space into smaller areas which can then be used to generate rooms and the final output. This is a slightly deeper depth than the explanation of how the BSP tree is used shown in figure 17, but we can see how this output is similar and how the BSP tree would develop.

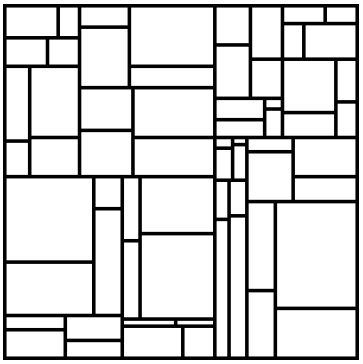
In figure 19, we can see a comparison of how minimum and maximum split values can affect the overall output of the dungeon. Figure 19b shows the dungeon



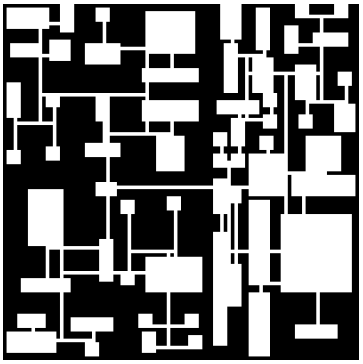
(a) Minimum split value of 45, maximum of 55. Before adding rooms and corridors.



(b) Minimum split value of 45, maximum of 55. After adding rooms and corridors.



(c) Minimum split value of 20, maximum of 80. Before adding rooms and corridors.



(d) Minimum split value of 20, maximum of 80. After adding rooms and corridors.

Fig. 19: Comparison of how minimum and maximum split values affect the BSP dungeon output.

generated using a high minimum split value and a low maximum before rooms and corridors are added. Since the gap between the two values is so tight, the subdivides have a very limited space they can be split upon, and so we end up with this grid array that looks very much like a series of alike cells, without much variation of size between them all. When this is turned into our final dungeon and rooms and corridors are added, shown in figure 19b, we can see that the dungeon overall has decisive sub-dungeons, with corridors from the center leading off into their own areas which do not connect with each other from anywhere else. Looking at the dungeon generated using a low minimum split value and a high maximum, figure 19c, we can see that the room sizes are a lot more varied and the overall placement of rooms is a lot more sporadic and clustered in some areas compared with others. This translates to the output shown in figure 19d. From this we can see that the dungeon has a lot less obvious sub-dungeons, and the areas where there is a large concentration of rooms appears very chaotic

with a lot of corridors and traversable space. Although there is still sub-dungeons created, such as in the top left corner, there is a less clear path between rooms and more options to travel from area to another.

3.3 Agent Based

The agent based method is another one that can produce very varied and interesting results, and is one we have implemented to showcase. This method may produce a result that will be in the rooms and corridors category of dungeons, as presented in section 1.6. At its core, this method is one of the most simplistic, but has a lot of opportunity for different approaches and adjustments to drastically change the results.

The most simple version of the agent based approach, is the drunkard walk. With this, first we find a random point in the map, and then give our agent a random direction from north, east, south, or west. Then our agent moves one tile in that direction, and carves out that tile. Then we choose another direction, move one tile, carve out, and repeat.

Whilst this method is chaotic and random, we can adapt on it to control the output more. One way to do this is to make it so instead of switching each iteration, it is instead a low chance to switch so the output will produce long corridors as opposed to more cavelike systems from the drunkard walk.

To adapt this into a dungeon generator instead of just carving long corridors across the map, we add a low chance to place a room of random size at every step. With this, we can also ensure that each room is connected to each other in some form.

The pseudocode implementation of this method is shown below, as outlined by Togelius et al. (2016), with some minor variations.

Algorithm 3: Agent Based Dungeon Generation

Result: A dungeon dug out by an agent
initalize change direction chance Pc = 5;
initalize add room chance Pr = 5;
place agent on random tile and random direction;
dig one cell along direction;
roll random number Nc between 0 and 100;
if *if Nc smaller than Pc* **then**
  randomize agent direction;
end
roll random number Nr between 0 and 100;
if *if Nr smaller than Pr* **then**
  randomize room width and length between set values;
  place room on agent position;
  randomize agent direction;
end
if *Dungeon is not large enough size* **then**
  go to step 4
end

Drunkard Walk The drunkard walk is a very simple implementation of the agent based algorithm. This works by starting the agent in a random cell on the board, and then each iteration the agent changes direction. By doing this, it can create some outputs similar to those generated by cellular automata. These can be seen below in figure 20.

Fig. 20: A sample output from the Drunkard Walk.

The pseudocode implementation for the drunkard walk is shown below.

Algorithm 4: Drunkard Walk Dungeon Generation

```
Result: A dungeon dug out by the drunkard walk agent
place agent on random tile and random direction;
set agent direction to random direction;
move agent along one tile in set direction;
dig out tile at agent location;
if cells dug is less than desired number then
    return to step 2
end
```

Implementation The agent system has been implemented in the software to demonstrate the agent algorithm. As part of that, the following variables can be adjusted to change how the algorithm performs.

- Switch Chance - This is the percentage chance that the agent will switch direction each step.
- Room Chance - This is the percentage chance that the agent will drop a room at the current location each step.
- Iterations - How many steps the agent will take to carve out the dungeon space.
- Switch Weight - When enabled, the agent will look ahead a variable number of cells to see if there is a carved area ahead, and if so increases the chance to switch to try and avoid digging the same area twice.
- Minimum Straight Lines - When enabled, this means that the agent must move a certain number of steps in each direction before it is allowed to switch direction.

In order to look into the effectiveness of the agent based algorithm, a set of default control parameters were used to generate outputs. These were used for all tests unless otherwise specified, and are as follows:

- Size - 100.
- Switch Chance - 5%.
- Room Chance - 5%.
- Iterations - 500.
- Switch Weight - Not enabled.
- Minimum Straight Lines - Not enabled.

The figure 21 shows the default agent's output. We can see from this that it does create room-like areas and connecting corridors, and this map could be used as a playable dungeon space. It is not without it's issues however, and some potential negatives to this output are the overlapping rooms that are created, and the agent doubling up on itself. Additionally, the agent does not travel to a large part of the map, and ends up concentrated on certain areas as opposed to others.

Fig. 21: An agent based dungeon using default settings.

(a) Room Place chance of 20%.

(b) Switch chance of 20%.

Fig. 22: Agent based dungeons using higher room values and switch values.

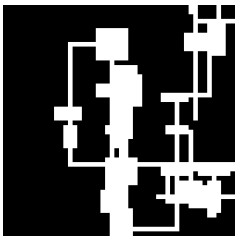
In order to understand how the room chance and switch chance affect the overall outputs, both variables were raised to 20% individually, and the results of this is shown in figure 22. We can see in figure 22a, that a higher room place chance, leads to the output being very heavy on empty space, and the corridors are not as visible as the default control output, due to rooms being placed on top of them. With the higher switch chance, shown in figure 22b, the output ends up very concentrated on one side of the map, and leaves a large area untouched. The agent seems to struggle to break out of the higher likelihood of switching direction, ends up very chaotic and without much variation of structure.

As we can see from the outputs shown in figure 21 and figure 22, the agent has a number of issues that we can attempt to mediate. The three main characteristics that could be considered issues, and potentially unwanted in the output are as follows:

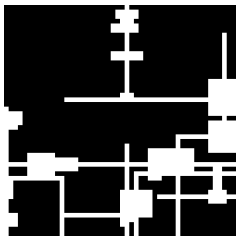
- A number of dead ends are created.
- Rooms overlap with each other.

– The agent gets stuck in one area of the map, and does not travel to parts of the map.

In order to attempt to solve and alleviate some of these issues, two techniques were implemented in the software. The first is a minimum straight lines variable, which attempts to solve the agent getting stuck in one area and spread across the rest of the map. This is a simple fix that means when the agent switches direction, it must then travel a minimum number of tiles in that direction before it can switch again. The second technique that was implemented was a look ahead algorithm, which looks ahead a variable number of cells in the direction the agent is travelling, and if it sees a cell that is already dug out, it increments the switch value by a variable value in order to increase the chance that the agent switches direction away from the already dug out area.

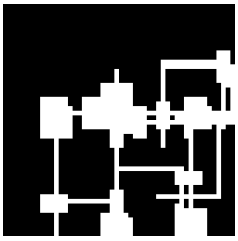


(a) Minimum straight line of 5 tiles.



(b) Minimum straight line of 10 tiles.

(c) Switch weight of 2, and look ahead value of 2.



(d) Minimum straight line of 10 tiles, switch weight of 1, and look ahead value of 5.

Fig. 23: Agent based dungeons using minimum straight lines and a look ahead algorithm to check for overlaps.

In figure 23, we can see the results of the two implemented techniques. In figure 23a, a minimum straight line of 5 tiles was used. This yielded fairly consistent results, with the agent spreading out across the map, and generating a central area with some offshoots. A higher value of 10 tiles, shown in figure 23b, the map is somewhat similar to the lower value of 5 tiles, but with longer corridors and space between rooms. Although the issue with overlapping rooms still exists, this technique also seems to mitigate this by creating longer spaces between the rooms and allowing for defined rooms and corridors to be created.

In figure 23c we can see the results of the look ahead technique, with a switch weight of two and a look ahead value of two. This means that for every movement the agent makes, it will look ahead two cells and for every dug out cell it sees, the chance to switch will be increased by 2%. This did not have the impact that we were hoping for, and the result is still very clustered in one part of the map and with overlapping rooms. As a technique, this is theoretically a good idea, but does need refinement and some fine tuning in order to function more effectively. The lack of success of this could be in part to the room placement, as rooms are placed on top of the agent, if the room is large enough then the look ahead algorithm will see the cells nearby as dug out, already, and could affect the effectiveness of the algorithm. A potential improvement to this could be to check if placing a room in the current position would overlap with any other rooms, and if so it will move to an area that the agent could place a room without overlapping on any others. This is the approach outlined by Togelius et al. (2016), and although not implemented in the software, their approach is shown in figure 24. This technique does solve the issue with overlapping rooms, but runs into the issue of the agent becoming unable to move to a suitable space, and prematurely ending the algorithm.

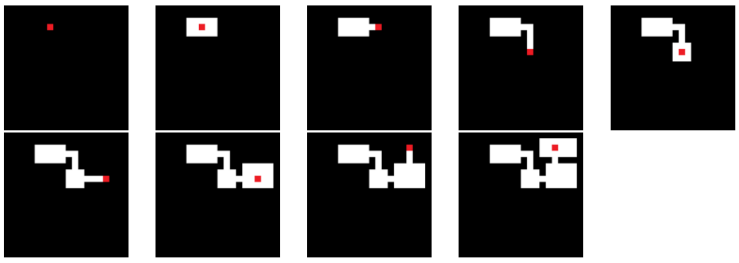


Fig. 24: An agent based dungeon with the agent checking for open spaces before placing a room (Togelius et al., 2016).

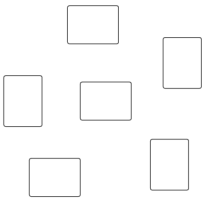
3.4 Constraint Based

Another method that can be used to create dungeons is the constraint based method, which is based off a web article by Adonaac (2015) with minor changes. This method may produce a result that will be in the rooms and corridors category of dungeons, as presented in section 1.6. This method starts by placing rooms of random height and width, and placing them within the boundaries of a circle so that they may overlap. It then separates the rooms so none of them overlap anymore, and connects them with corridors.

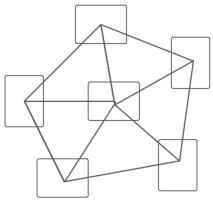
In order to separate the rooms, Adonaac uses a simple physics engine to push the rooms apart based on their boundaries, and to make sure they do not collide. While this did prove to be effective, we opted for an alternative approach

to this which iterates through each room and pushes it apart if it intersects with another room. First it checks if the room is overlapping, and if so it moves the room in a direction away from the previously overlapping room, and repeats this process for each overlapping room. Following on from this, the next step is to connect the rooms with corridors. To do this, Adonaac (2015) used delauney triangulation and a minimum spanning tree. These techniques are commonly used in combination with each other to create corridors between rooms, and will be discussed below in more detail.

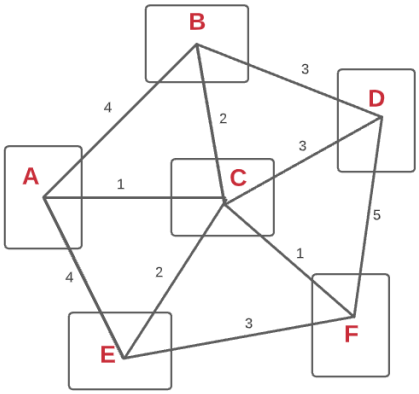
Connecting Rooms One method of connecting rooms within a dungeon is to use a combination of delauney triangulation and minimum spanning trees, to connect rooms and ensure a path is available from every room.



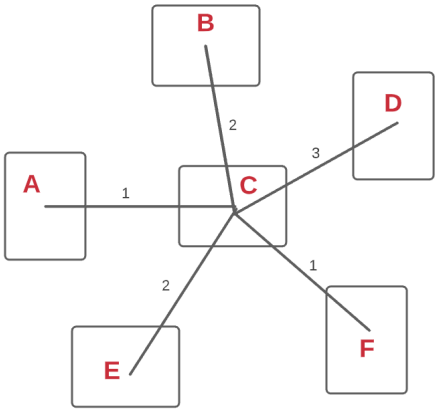
(a) A collection of unconnected rooms.



(b) Rooms connected via delauney triangulation.



(c) A set of weights is given to the edges of the graph.



(d) A completed minimum spanning tree.

Fig. 25: The process of using delauney triangulation and minimum spanning trees to connect rooms.

Delauney triangulation takes a set of points on a map, and connects them via triangles. In figure 25a we can see a small collection of rooms, which we will connect using delauney triangulation. In order to do so, we are going to take the

center points of each of our rooms, and connect them via triangles. An example of the output from delauney triangulation is shown in figure 25b, with all of the rooms connected. Here we can see that each node, a room center, is connected to other nodes via an edge. These edges will form the basis of our corridors, but first we must reduce the amount of them in order to ensure every rooms cannot be accessed from all nearby areas.

Following on from this graph of connected points that has been created by delauney triangulation, we can use a minimum spanning tree to cut create paths to every room and ensuring every point of the map is reachable, whilst cutting down on the amount of connections we have. To do this, we must first assign a weight to each of the edges on the graph. This weight can be assigned based on many variables, but a common example and one that can be applied to our room connection is distance. In figure 25c we can see a graph with edges given weights, and rooms labelled. For the purposes of this example the edges have been given random weights. From this point, there are two main approaches we can take to create our minimum spanning tree, they are Kruskal's algorithm and Prim's algorithm, although it should be noted that the two approaches often produce the same result.

Looking at Kruskals's algorithm, the first step is to arrange all of the edges in ascending order by their weight. For our graph, this will look like table 1.

1	2	3	4	5
AC	BC	CD	AE	DF
FC	EC	DB	AB	
		EF		

Table 1: Edge weights organised in preparation for Kruskal's algorithm.

Now that we have our edges organised by weight, we can go through each of the edges and decide where or not it will be included in the finished graph. We start with the lowest weight edges, so in this case the edges between AC and FC. To decide whether each edge will make it to the finished product, we check if including the edge would form a complete cycle if it was included. A complete cycle is formed if placing the edge would create a triangle between rooms. So in the case of our example in figure 25c, if AC and EC are included, then AE would not be used when it is checked. Kruskals iterates through all of these edges, checking lowest weights first, and produces a minimum spanning tree as a result.

The other algorithm that can be used to create minimum spanning trees is Prim's algorithm. This method works by first selecting any node on the graph, it does not matter which. For example, we will pick the A node. Once we have selected the root node, now we look for the lowest weight edge attached to the node, and include that in our final generation. Each time that we find the lowest

weight node, we will also check and make sure that including it will not form a cycle, as discussed in the previous section. Once we have validated the lowest length node, we can include it and extend our root node to include the node we have added. As an example, the lowest weight edge from our root node A is AC, so we include it. Then, we look for the lowest edge attached to both nodes A and C, as they are both now included in the final generation. The lowest weight node attached to A or C is CF, and so we include this edge. Now, we look for the lowest weight node from node A, C, or F and include this in the graph, and continue like this until we have a completed minimum spanning tree.

An example of a result from this process is shown in figure 25d. Each room is connected in some way, and there is no cycles formed. From this tree we can create corridors between those rooms, and connect rooms in a similar visual style as BSP.

A simple pseudocode representation of the constraint based algorithm is shown here.

Algorithm 5: Constraint Based Dungeon Generation.

```
Result: A dungeon created using constraint based algorithm
initialise radius Rad;
initialise number of rooms Rms;
for number of Rms do
    generate a room of random size within Rad;
end
separate all rooms from each other with no overlaps;
create corridors between rooms;
```

Implementation As part of the software, constraint based was included as a dungeon generation technique. In order to connect the rooms, a simple algorithm is run to check a certain radius near each rooms center to find other rooms, and connects them if so. This ensures that the dungeon is fully connected provided each room is within a certain radius of each other. Additionally, the following adjustable variables were included and descriptions of what they are for can be found here:

- Number of Rooms - This is the number of rooms that will initially be spawned.
- Radius to Spawn - This is the radius within which rooms can spawn, a larger radius means rooms will start spread further apart.
- Minimum Room Size - The Minimum size a room can spawn to be.
- Maximum Room Size - The Maximum size a room can spawn to be.
- Distance For Corridors - This is the maximum distance that corridors can be spawned from a room, a shorter distance means less rooms will be connected.

For the purposes of looking into the results of this algorithm, the following parameters were used, unless otherwise stated:

- Size - 100, image cropped for ease of viewing.

- Number of Rooms - 20.
- Radius to Spawn - 10.
- Minimum Room Size - 2.
- Maximum Room Size - 8.
- Distance For Corridors - 10.

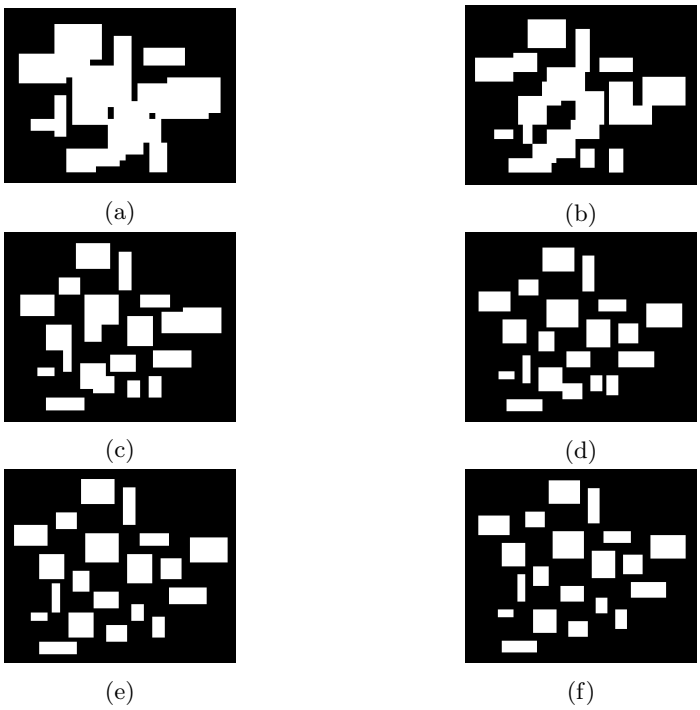


Fig. 26: The progression of the constraint based dungeon development, as rooms are separated from each other.

In figure 26, we can see how this algorithm creates the basis of a dungeon, with spread out rooms of varied size. Although no corridors have been added yet, we can see that this could be turned into a playable space by connecting these rooms. With this technique, the rooms all end up in relatively close proximity to each other, and there are no clear outliers or unique spaces generated, and is very consistent in its approach.

In figure 27 we can see an output with initial radius 15, larger than the radius 10 used by the previous example. Increasing the initial radius means rooms will start more spread out, and will take less iterations to separate all of the rooms. By increasing the initial radius, we can aim to create more spaced out dungeons and have longer corridors between rooms.

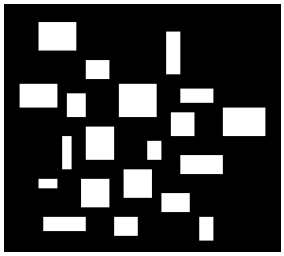
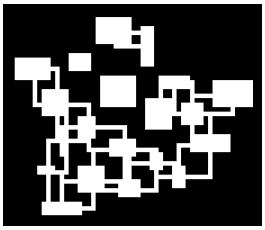
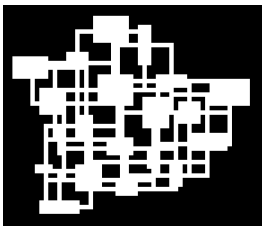


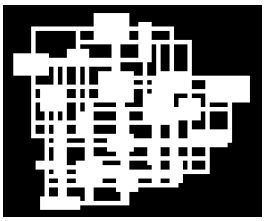
Fig. 27: A constraint based dungeon with initial radius 15, with no movement apart applied.



(a) Radius 10.



(b) Radius 15.



(c) Radius 20.

Fig. 28: The progression of the constraint based dungeon development, as rooms are separated from each other.

As part of looking into this algorithm, we needed a way to create corridors between the rooms and turn this map into a playable dungeon space. To do this, we used a simple algorithm that took the center point of every room, and searched a variable distance of cells nearby for all other rooms. This was used over minimum spanning tree's due to time limitations. After this, for every room found, a corridor is created between the two rooms, either in a straight line or in an L-shape dependent on the positioning of the rooms. In figure 28 we can see the results of this technique, with differing distances. Figure 28a shows the default distance 10, and we can see that the dungeon ends up moderately well connected, although the further away rooms do not end up with a connection due to them being too far away from any other rooms. In figure 28b we can see a higher radius, this time 15, and here every room does end up connected. Due to how close some rooms are however, we end up with this intricately connected maze of sorts, with corridors between a lot of rooms and a significant amount of unnecessary connections between rooms as there are already ways to travel from one to another. Figure 28c shows a higher radius again of 20, and here the dungeon becomes even more of a maze, with the corridors in some cases overlapping and blurring the line between what is a room, and what is overlapping corridors.

3.5 Evolutionary Algorithm(s)

Evolutionary Algorithms are the broader term for a series of evolutionary processes that aim to replicate natural selection and reproduction but applied to specific computational problems. They are stochastic meta-heuristics, and due to being based around real life reproduction, has a lot of basis grounded in Darwin's theory of evolution. Essentially, these work by keeping a list of structures, and testing each individual in the population to assess its fitness for the environment. The fitness is calculated using a function that is manually defined to assess suitability for the hypothetical output. After each iteration, the highest fitness states are selected to evolve and carry forward to the next iteration, where the cycle begins anew. This four step process is summarised by the terms Evaluation, Selection, Recombination, and Mutation. Once the population reaches the point where its result meets a criteria set, be it a set number of iterations or meeting constraints set manually, it ends, and a final result is output.

Following the approach proposed by De Jong (2006), Wong (2015) breaks down Evolutionary Algorithms into 6 distinct sections of the overall cycle. These are as follows:

- Representation – How the data or variables to be mapped are represented. For example, the number "19" can be represented in Binary as 10011. If the left-most bit is mutated, we have 00011, or "3". This is known as Genotype representation and genotype-phenotype mapping.
- Parent Selection – Selects good parent individuals based on a fitness function. Aims to produce higher fitness children so typically selects most worthy candidates from previous batch.
- Crossover Operators, also known as Recombination – This represents reproduction in nature. Alongside Mutation operators, these are known as reproductive operators. Crossover operators combine two individuals to create a new one, by splitting those two individuals into two parts and recombining them.
- Mutation Operators - Simulates mutations in genes such as what happens in nature. Changes parts of a genome to create something new. This is particularly useful for exploration in the algorithm to ensure varied results.
- Survival Selection – Selects a good subset of individuals, usually related to fitness. This is very similar to parent selection. In a framework such as "EC4" (De Jong, 2006), most of the parent selection mechanisms can be used here.
- Termination Condition – The condition by which the algorithm will end.

Below in figure 29, we can see a general representation of the evolutionary algorithm cycle.

Evolutionary Cellular Automata Evolutionary algorithms can be utilised to create dungeon layouts in combination with cellular automata, with a technique known as evolved cellular automata. Pech et al. (2016) uses this to generate map outputs, utilising all of the usual CA techniques but using a GA to generate the

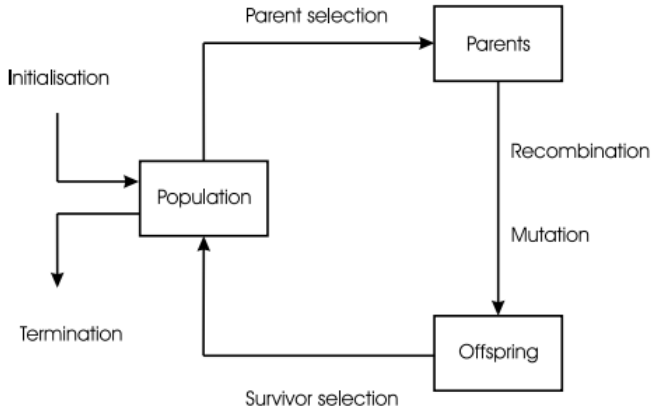


Fig. 29: General Cycle of an Evolutionary Algorithm, image from Eiben and Smith (2003)

rules. With this approach, they replace cellular automata parameters with an evolutionary algorithm generated set in order to achieve a result that have a similar visual style to a "goal" layout. This means they do not use parameters such as the birth limit and death limit, but instead use their own rule set for establishing what value a cell should be based on its neighbours, which will be discussed in this section. Evolved cellular automata was not able to be implemented in the software discussed previously, however here we will outline and discuss the approach proposed by Pech et al. (2016).

In order to identify this "goal" layout, Pech et al. (2016) identified 9 attributes of a level to identify the visual style. These are shown below:

1. Number of traversable areas.
2. Size of the largest traversable area.
3. Average size of all traversable areas.
4. Number of dead ends.
5. Number of rooms.
6. Average size of rooms.
7. Number of corridors.
8. Average length of corridors.
9. Numbers of culs-de-sac (rooms with only one way leading in/out).

Their approach then follows three key steps to generate final outputs. These are as follows:

- A way to extract the level attributes from a given input.
- A level layout generator using CA.
- A GA to evolve the CA rules.

Extracting level attributes The first step for Pech et al. (2016) was to extract information on the nine level attributes from a "goal" level provided as an input. To do this, they used an image processing technique for region growing which was not named, and an erosion operator based off a technique used in image processing Serra (1986) (as cited in Pech et al. 2016), amongst other techniques which are beyond the scope of this paper.

CA level generator The second step here was to define the form that the CA rule sets would take. For this, they used lookup tables where the neighbourhood of a cell is taken as the input, and the associated GA would return the output state that the cell would take once the rules have been applied (Pech et al., 2016). In order to represent these rule tables, they explored two representations, a direct representation, and an indirect representation.

The direct representation is suitable for a CA where each cell can have one of two values. This approach differs from the indirect representation on this key aspect, as the indirect representation allows for complex CA to be explored, where a cell can have more than two possible values. With the direct approach, an output state it stored for every combination of neighbourhood, and the output state is looked up based on what neighbourhood configuration it has. This is shown in figure 30 with a simple 1D representation of this process, comparing the neighbourhoods unique index to the rule table in order to calculate the cells new output state.

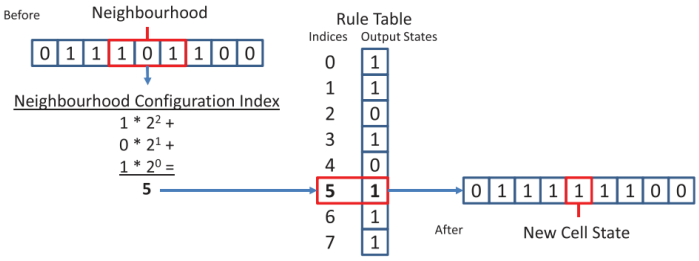


Fig. 30: Direct representation of a CA rule table. Source (Pech et al., 2016).

As discussed previously, the indirect representation is more suitable when a cell might have multiple states or a complex neighbourhood. An example of this is shown in figure 31 using a 1D array of cells. From this, we can see that the key difference with this representation is that cells are given a value, and in order to calculate the output state of an individual cell, the total sum of its neighbourhood is entered into the rule table. This approach is much more suited to multiple cell values and complex neighbourhoods as it significantly cuts down on the lookup table size. If a lookup table was to be used for a CA where each cell could have 5 values for example, the lookup table would be a significant amount larger with the direct approach where every combination

of neighbourhoods would have to be accounted for, as opposed to the indirect approach where only the summed values have to be accounted for.

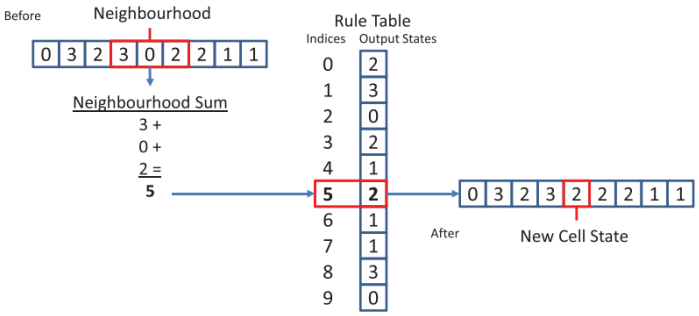


Fig. 31: Indirect representation of a CA rule table. Source (Pech et al., 2016).

Genetic Algorithm The third step to the approach laid out by Pech et al. (2016) is to utilise a GA to generate outputs, based on the previous steps. Based on the outline of an EA by Wong (2015), this can be described as follows:

- Representation - The chromosomes are represented by CA rule sets, as described in the second step of Pech et al.'s approach.
- Parent Selection - First, the fitness function will evaluate each chromosome to determine its fitness. It does this by first generating 100 initial map layouts, using these for every individual in the generation. Then, it applies the rule table associated with each chromosome to the 100 layouts. For every layout after the rule table is applied, its attribute similarity measure is calculated by extracting the level attributes from the layout, and comparing this to the extracted attributes of the goal layout. After this, each chromosomes fitness is calculated by getting the average similarity measure of the 100 layouts generated. With this fitness, measure, tournament selection with a group size of 5 is used to select the parents.
- Crossover Operators - Crossover is performed with a certain probability, meaning not every pair of chromosomes has crossover applied. When it does happen, single point crossover is applied to generate new chromosomes.
- Mutation Operators - Uniform mutation is applied, again with a certain probability so it may not always occur. The mutation operator replaces genes with random values between 0 and the total number of cell states, creating new CA look up tables.
- Survival Selection - This GA employs an elitism survival selection, where the top five individuals are carried over to the next generation.
- Termination Condition - The GA will continually iterate through the previous steps until one of three termination conditions occur. The first is convergence, where the fitness of the highest ranked individual has not been

increased by 0.0001 after 100 consecutive generations. The second is a maximum fitness value, where the highest fitness reaches 1.0 and cannot be improved further. The third is a maximum number of generations, if no other termination condition occurs before the 5000th generation, then this termination condition is reached.

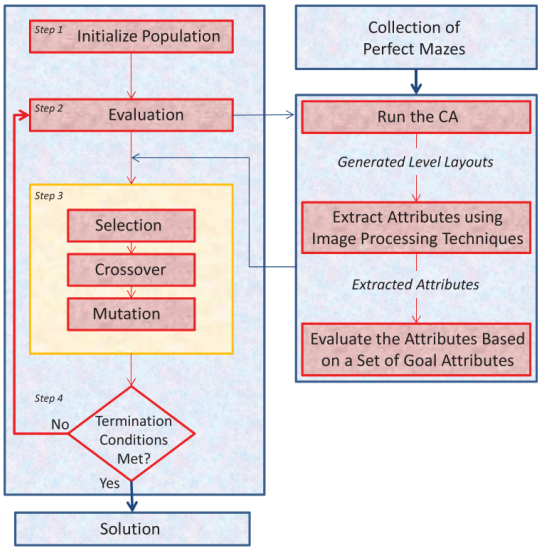
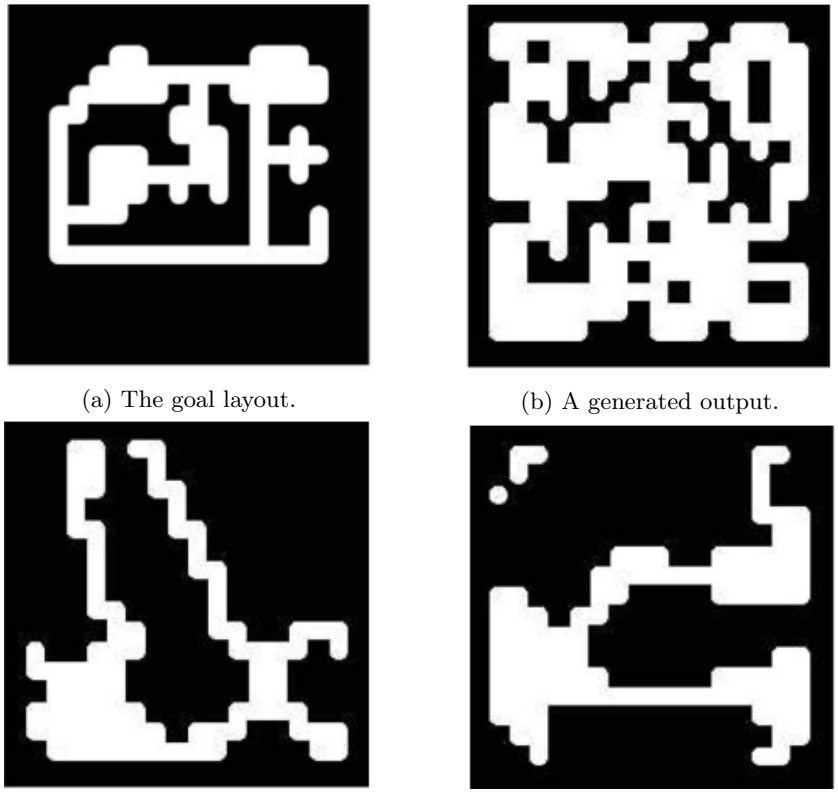


Fig. 32: The GA cycle for evolutionary cellular automata. Source (Pech et al., 2016).

The total cycle for this GA is shown in figure 32. From this we can see how this fits into the cycle of an evolutionary algorithm as shown in figure 29, with the GA working alongside the attribute extractor. To initialise the population, each chromosome has a rule table generated with a random value from the possible cell values. We can see from this figure how each of the described elements fit into the overall structure of the GA, with the input mazes becoming the set of goal attributes, and at each step, new layouts are generated and evaluated against these.

In figure 33 we can see a series of generated outputs by the evolutionary cellular automata, alongside the goal layout that was originally input. As we can see, the outputs are resemblant of the original, and this method could be used to create playable dungeon-like spaces. Although this concept is still relatively infantile, this approach does prove the theory behind this does work, and can be used to create similar looking level layouts given a goal input.



(a) The goal layout.

(b) A generated output.

(c) A generated output.

(d) A generated output.

Fig. 33: Generated outputs using evolutionary cellular automata. Source (Pech et al., 2016)

3.6 Future Approaches

Whilst the software created does effectively display some of the algorithms discussed here, it does not include other methods that may produce interesting and varied dungeons. Other techniques that can produce interesting results and/or include work relevant to the field are not discussed in this paper. Some examples of these are as follows:

- Generative Grammars - Van Linden et al. (2013).
- Relative Placement - Valtchanov and Brown (2012).
- Mixed Initiative Genetic Algorithm - Liapis et al. (2013).
- 3D Constraint Based - Roden and Parberry (2004)
- Cyclic Generation - Dormans (2017)

3.7 Evaluation

After looking at the implemented and discussed methods within the previous section 3, we can now look into the potential applications, limitations, and comparisons that we can draw between the dungeon generation algorithms. References are given where the findings of this paper agree with the literature.

Advantages	Disadvantages
Fast to run	Parameter impact is hard to understand
Small number of intuitive parameters	No connection to gameplay
Large number of possible variations	Lack of direct control of output
Versatile	Difficult to adjust for 3D space
	Connectivity between areas not guaranteed

Table 2: Advantages and disadvantages of Cellular Automata. (Togelius et al., 2016); (Van Linden et al., 2013); (Johnson et al., 2010); (Williams, 2014).

Cellular Automata Looking at cellular automata, its main attraction is in the natural and unstructured looking outputs it produces. The type of outputs it can produce are very organic looking and can create very varied spaces. Due to the low level nature of the algorithm however, one of the main issues it can have is the lack of connectivity between the spaces, and control over what is created. Often areas may be created that cannot be accessed from other parts of the dungeon, and external techniques such as a flood fill algorithm may have to be employed to ensure the map is fully playable. This also could lead to issues when looking at fleshing out the dungeon into a playable space within a game, as additional attributes such as treasure rooms, monster placement, and other elements could be challenging to implement. Additionally, it is hard to have any control over this output due to the random nature of its generation, and the user has no ability to control elements such as ensuring certain room spaces are connected or clear paths from one area to the next.

Looking at how a user may use this algorithm, the parameters are easy to understand and simple to adjust, with very clear differences between the extremes of each parameter. When trying to fine tune the algorithm however, these parameters can be very hard to understand the impact each is actually having as the results may look very different simply due to the nature of the algorithm. This could be a potential advantage of the algorithm as well, as it can produce a large variety of outputs that may look very different each time the algorithm is run, so it is unlikely outputs will be similar to each other.

Spatial Partitioning Spatial partitioning has a number of advantages that set it aside. Of note here are its relative simplicity and ease of understanding for the

Advantages	Disadvantages
Easy to understand	Repetitive generation
Guaranteed connectivity	Looks unnatural and man made
Little overlapping rooms	Limited control
Potential application in 3D environment	

Table 3: Advantages and disadvantages of Spatial Partitioning. (Togelius et al., 2016); (Niemann & Preuß, 2015); (Blomberg et al., 2018).

user, and that the output of this algorithm guarantees every room is connected without external techniques needing to be applied. The outputs this algorithm creates have a very traditional and formulaic style to them, with every room in its own respective space meaning little to no room overlap. This could also be considered a disadvantage however, due to the relatively basic generation of the rooms, the outputs may not have much variety and could end up predictable. Additionally, these outputs can end up looking very unnatural and man made, which may be desirable if thats the style the user is aiming for, but often can detract from the dungeon and lower immersion. If additional gameplay features such as special rooms or monster rooms are to be implemented, this approach does provide a good basis to create these, due to the connectivity of the dungeon and access to the node tree for dungeon structure.

Another thing to note about spatial partitioning is the potential application for 3D spaces as well, as octree's allow for access to a 3D BSP tree.

Advantages	Disadvantages
Flexible to requirements	Lots of parameter tuning
Lots of potential	Lack of direct control
Application in other PDG areas	

Table 4: Advantages and disadvantages of Agent Based. (Togelius et al., 2016); (Blomberg et al., 2018).

Agent Based Looking at the agent based algorithm, one of the standout advantages is in the potential of this technique. By using an agent to create the dungeon for us, a lot of this potential lies in the user developing the agent, and how much work that user is willing to put into the agent to tune parameters to desired requirements and to adjust how the agent operates so that it functions as the user wants it to. This is also one of the major downsides of this algorithm, simply due to the amount of tuning and time that needs to be put into the agent in order for it to create a map that satisfies the users requirement. Additionally,

due to the random nature of the agent, it can be hard to test whether it is correctly responding to small changes to the parameters.

One thing that has not yet been discussed in this paper, is the potential for the agent to be used to generate alternative elements such as feature placement of items, doors, or treasure. Another possibility is for the agent to simulate a player, and travel through a level looking for dead ends or impassable areas. These are just speculative ideas, but shows the agent approach could have a variety of uses and be a potentially very useful tool in the dungeon creation process.

Advantages	Disadvantages
Simple parameters	Lack of Control
Potential varied sizes	Lack of variation in results
Consistent size	

Table 5: Advantages and disadvantages of Constraint Based. (Adonaac, 2015).

Constraint Based As this constraint based approach is not a well known algorithm, the analysis of this algorithm will be based on the results from the implementation in the software. This approach shares a lot of similarities with space partitioning in its advantages and limitations. The algorithm has simple parameters and is intuitive to use, but this is also one of the disadvantages as results can often end up very similar to each other, and can be predictable with little variety to the output. This technique is very consistent overall, with maps ending up very similar in size shape. A potential change that could be applied to create more variety in the dungeons is to create different shapes of room, such as L-shaped rooms or circular shaped rooms which could replace an amount of rooms in the dungeon.

Advantages	Disadvantages
All advantages of normal CA	Specific visual attributes hard to replicate
Control over output	Time to generate
No CA rule tuning	

Table 6: Advantages and disadvantages of evolutionary cellular automata. (Pech et al., 2016); (Niemann & Preuß, 2015).

Evolutionary Cellular Automata Evolutionary Cellular Automata is arguably the most unique out of the algorithms presented in this paper, and as

such has a number of of advantages and disadvantages. Firstly, this approach incorporates all of the major elements of cellular automata and so also has a lot of the advantages associated with this. Mainly, a major advantage of this is in CA's versatility and near limitless variety of outputs. It can be used to create a large amount of outputs depending on the requirements of the user and what parameters the CA is run with. However, the evolutionary cellular automata requires a significant more time to generate than the basic CA due to the evolutionary algorithm needing time to run. Using the evolutionary CA we can cut down on time required to tune parameters and the trial and error required to produce a suitable CA output however, as the evolutionary CA does this for the user.

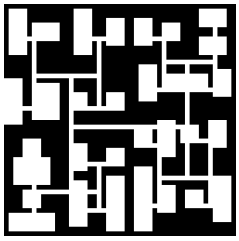
This method allows for a lot of control over the final output, as the outputs are trying to replicate the input map, meaning if the user wants a style of map to be created, if they provide a suitable input then the evolutionary CA will generate similar outputs. A potential limitation of this algorithm is that it does struggle to replicate specific visual attributes, if the user wants the output to contain a certain variation of cells or smaller attributes then the evolutionary CA will struggle to accurately reproduce this.

Comparisons Now that we have looked at some of the potential advantages and limitations that each algorithm might have, we can make some comparisons between them and look into the situations and applications that one method might have more suitability than others. In figure 34 we can see a sample output from each of the five main methods presented in this paper, cellular automata, binary space partitioning, agent based, constraint based, and evolutionary cellular automata. Here we will analyse these algorithms and compare them to each other under the following categories:

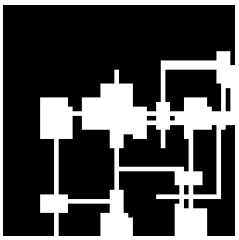
- Control - The amount of control each algorithm gives the designer, and how much control is given over each element of the output.
- Ease of use - How simple the algorithm is to understand, and how easy the algorithm is to generate quality results.
- Consistency - Whether the algorithm gives consistent results and could be relied upon to generate similar outputs if required.
- Suitability for a game level - How easy the outputs produced could be translated into a game level.
- Potential for the future - If improvements can be made that the algorithm could benefit from, and whether the algorithm provides a good basis for a designer to adapt for more complex levels.

Control Looking at the amount of control each algorithm gives the designer, overall, it seems that evolutionary CA gives the most amount of control as it attempts to replicate goal layouts provided by the designer, and the parameters of the EA such as the crossover and mutation probability can also be adjusted to change how the output turns out. However, this is a different type of control to

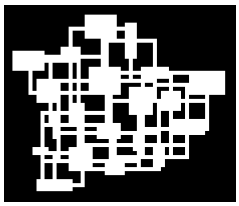
(a) Cellular Automata output.



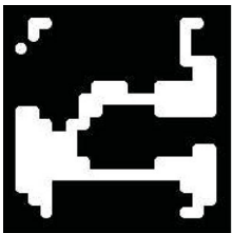
(b) Binary Space Partitioning output.



(c) Agent Based output.



(d) Constraint Based output.



(e) Evolutionary Cellular Automata output.

Fig. 34: The outputs of the methods discussed in this paper.

the rest of the algorithms, and this method does use an evolutionary algorithm that is ultimately out of the designers control whilst it was running. It's challenging to properly assess the amount of control that evolutionary CA gives the designer in comparison to the other algorithms discussed, simply due to the fact that its method of content generation differs so greatly from the others. Normal cellular automata on the other hand, we can look at in more detail. With this method, the outputs tend to be very varied the designer has little to no control on the details of the output. The parameters used to control can often have unclear impact on the final result, and it can be a challenge for a designer to ensure a CA output map will have elements such as clear rooms and spaces to move between areas, without using additional techniques. Overall, cellular automata has very limited control over its outputs, with only a small number of parameters which may not have obvious impact on the output. On the other hand, BSP is a stark contrast with CA in terms of room generation and clear paths between areas. The outputs produced by BSP tend to be very straightforward, with rooms

connected by corridors, and areas split into smaller sub dungeons with limited ways in or out. Despite this, BSP still struggles from some of the same issues with control as cellular automata. Limited parameters are available to adjust the algorithm with, and small issues with the output can be hard to have their cause identified. The agent-based method is an interesting algorithm on the topic of control. Overall, it allows for perhaps the highest amount of control over how the algorithm actually performs, as the designer can adjust all of the individual parameters of how the agent will act and tune these to their satisfaction. However, this can be an incredibly difficult feat to achieve due to the amount of testing that will need to be done to assess the viability of a setting. Additionally, there is a distinct lack of direct control whilst the agent is running, and its viability is largely dependent on the tuning the designer has done beforehand. Finally, the constraint based algorithm falls into the same category as BSP, very limited parameters available to adjust the algorithm.

On the whole, the agent-based method allows for the most control over the output, but this assumes that the designer is willing to put the time in to tune the individual parameters in order to allow the agent to produce an output that matches their requirements. It's worth mentioning here that evolutionary cellular automata does also allow for a significant amount of control over the output, through the specification of a goal that the evolutionary CA will try to create similar layouts to. However both of these methods do still suffer from a lack of direct control and require parameter tuning to function well.

Ease of use The ease of use of an algorithm is mostly dependent on how simple an algorithm is to setup and produce good quality outputs with. Starting with cellular automata, this is a very low level concept and once implemented, can be simple to produce interesting and varied results with. BSP is also not particularly complex in implementation, and is very similar to cellular automata in its ease of use once it has been implemented due to a low number of parameters. The constraint based approach is perhaps slightly more complex than CA and BSP, simply due to the fact that it's corridor creation is not implemented alongside the rest of the algorithm, and must be achieved through additional means such as distance based connections or a minimum spanning tree. The agent based method is perhaps one of the simplest of them all to set up in its basic format, although requires much more investment to produce quality results compared to the previous methods discussed. Evolutionary Cellular Automata is the most complex out of the methods outlined here, and requires significant setup and tuning in order to produce quality results.

Out of the methods discussed, cellular automata and binary space partitioning are the most accessible and provide the highest ease of use, due to their relative simplicity in implementation and parameter setup.

Consistency Here we are looking for how consistent an algorithms outputs are, and if a designer could rely on an algorithm to provide a map that is very similar to the last each time it is run. Looking at cellular automata first, there is not a high level of consistency with this approach as each output may look drastically

different from the last, and there is no guarantee that two outputs with the same parameters would share similar elements or structures due to the random nature of the initial generation. Moving on, both binary space partitioning and the constraint based approach produce consistent results, and will rarely, if ever, produce outputs that are extreme outliers or differ greatly from other results. The agent based method does have issues with consistency, since it is heavily reliant on random elements such as its chance to switch direction. Some outputs may be drastically different to others as a result of this, although it should be noted that the designer could implement approaches to mitigate the random element and potentially improve consistency, such as the approach of minimum straight lines or a look ahead technique, as discussed in section 3.3. Finally, the evolutionary cellular automata additionally can have significant issues with consistency, due to the nature of creation and evolutionary algorithms not always reaching high fitness values.

With regards to consistency, overall the BSP algorithm and constraint based approach produce the most consistent results. There are few factors that could cause these two algorithms to produce outputs that are significantly different from the rest, and the majority of the time they will produce similar quality results.

Suitability for games Although each of these outputs could be translated into a game level, it may be easier to translate some outputs into a game space than others. Firstly, looking at cellular automata, there are a number of differentiating features that set this algorithm apart from others. One distinct difference is that CA produces cavelike outputs, without defined rooms and corridors, as opposed to the outputs created by BSP, agent based, and constraint based. This may make the outputs produced by cellular automata more suitable for a 2D side view game, where the player is affected by gravity and can jump or move from platform to platform. Additionally, the outputs generated by CA may not always have defined boundaries and clear paths from area to area, and in order to translate a CA output into a playable level, additional techniques such as a flood fill would have to be used. Evolutionary cellular automata is similar to CA in its suitability for games, and all of the points made for normal CA can also be applied to this. Additionally however, the EA needing time to run in advance and not at runtime makes evolutionary CA less suitable than normal CA for most games. Looking at BSP, the outputs have very defined rooms and corridors, and have clear paths between all areas of the maps. For a simple level within which the player can move from room to room and must travel from one area of the map to another, BSP seems well suited. If the designer wanted to include additional gameplay features, it would not be too challenging to simply change rooms generated into special rooms or to add features such as barriers or enemies. For example, since the output produces a number of sub dungeons with a limited number of entry and exit points, a barrier could be placed across one of the entrances to a sub dungeon and a key placed somewhere else on the map that will unlock this area. This is a simplified example of course, but shows how the algorithm could be used to create a game level. Additionally, this algorithm

allows for access to the node tree that makes up the structure of the dungeon, which could potentially be useful for identifying key areas of the map or which areas have a higher concentration of rooms. The constraint based algorithm falls into a similar category to BSP, as it provides a framework of connected rooms and corridors. However, it does not have the sub dungeons that BSP generates, and thus could encounter more issue when trying to incorporate gameplay features. The agent based method is interesting when it come to its suitability for games. In theory, when the agent is active and digging out the dungeon, it could also populate it with gameplay elements such as treasure or monsters. This could be based on random chance, or elements such as total number of placed rooms or length of the dungeon so far for example. In practice, it's difficult to gauge exactly how successful this would end up being without further research, and would take a lot of work to ensure that the dungeon does not end up chaotic and with no order. As previously discussed, with refinement of its parameters the agent can produce quality results, but adding yet another element on top of its existing parameters in the form of gameplay features may push this to a point that the dungeons end up unrefined and with no order, however it definitely has potential for gameplay elements to be added and could be done so during dungeon generation.

Overall, the BSP method stands out as an algorithm that allows for gameplay elements to be incorporated with relative ease compared to other methods. The constraint based is similar in this regard, but falls short due to its lack of sub dungeons and no node tree to access, although this could be improved by alternative means of connecting rooms such as a minimum spanning tree which could improve this method.

Potential Finally, we look at the potential for improvement that each of these algorithms has, and how limited they are in scope. Looking at cellular automata first, there is limited potential for cellular automata by itself, due to the limited parameters and simplistic nature of the algorithm. There may be potential in looking at the way each cell is determined to be dead or alive, potentially employing additional methods of checking the current structure of the dungeon as a whole and using this as a factor in determining individual cells value, although this may be going beyond the scope of basic cellular automata and moving into more complex algorithms. Evolutionary cellular automata seeks to improve upon normal CA by developing its own rulesets for each cell, and this is an algorithm that has much higher potential for improvement that normal CA. While the evolutionary CA algorithm does produce outputs similar to its goal layouts, there is no doubt that this could be improved further still, and in an ideal world being able to consistently produce map layouts that are similar to a goal layout could trivialise a lot of work that game designers have to do to produce maps via other methods. We can see the potential that evolutionary CA has, and the nature of the EA used to produce outputs could allow for potential improvement as the field of evolutionary algorithms is expanded going into the future. With the spatial partitioning method, there is a limited scope for potential improvement, but some examples could be dungeons created using alternative node tree types,

improvement to how the algorithm generates rooms, as it could be improved to generate gameplay elements such as special rooms or barriers alongside its basic room and corridor generation. Constraint based is similar to the BSP method, with limited scope for potential improvement. As discussed in this paper, utilising alternative methods of corridor connection such as minimum spanning trees could improve this method, and allow it to create outputs similar to BSP but potentially with more variation. It could also be improved by allowing for generation of gameplay elements when it is run, such as by replacing a certain number of rooms with special rooms. Finally, the agent based method has a high amount of potential, as it is mostly limited by the parameters and rules the designer is willing to give it, however the time and cost of improvement for this method could potentially be very high as it requires a significant amount of tuning and refinement in order to function well.

Out of the algorithms discussed, evolutionary cellular automata will have the most scope for improvement going into the future. The field of evolutionary algorithms is constantly growing, and could greatly benefit dungeon generation. Although there are still limiting factors such as time to run, we can see that the algorithm can produce comparable outputs to a goal input, and this could potentially be employed to great effect. A notable mention also goes to the agent based method, which has a high scope for potential but is limited by testing capabilities.

Summary To summarise the comparisons between algorithms, a table has been created that shows an approximate order for each of the algorithms and how they compare to each other on the five areas discussed, control, ease of use, consistency, suitability for games, and potential.

Comparison of Algorithms					
Rank (Best to worst)	Control	Ease of Use	Consistency	Use in Games	Potential
1st	Agent	CA	BSP	BSP	ECA
2nd	ECA	BSP	Constraint	Constraint	Agent
3rd	BSP	Constraint	CA	Agent	BSP
4th	Constraint	Agent	Agent	CA	CA
5th	CA	ECA	ECA	ECA	Constraint

Table 7: A summary of the comparisons discussed between the algorithms.

In table 7 we can see a summary of the algorithms discussed, and what their strengths and weaknesses are. From this table, we can see that every algorithm discussed has strong points and weak points, and no algorithm provides the best in every category. Notably, the BSP method ranks well and does not have any low ranking places within the categories discuss. This table provides a quick visual summary of situations where one algorithm might be preferred against another, and summarises what was discussed in this section.

4 Conclusion

In this section we will discuss the success and failures of this project from a critical standpoint. Additionally, we will look at the authors thoughts on the project and where it could be taken in the future.

4.1 Overall Conclusion

In this paper we looked into a selection of procedural dungeon generation algorithms, the background information behind them, methodology, their advantages, and their limitations.

The main goal of this paper was to research and look into different ways to procedurally generate dungeons, and to analyse the effectiveness and potential applications for each. Overall there is a large variety of dungeon generation algorithms that can be employed, each with their own advantages and disadvantages that may set them apart from each other. Whilst it was found to be difficult to draw comparisons between algorithms due to the unique nature and suitability of each algorithm, we can assess the positives and negatives of each algorithm to determine in what situation an algorithm might be preferred.

At the beginning of this paper, in section 1.1, we identified the research question this paper would strive to answer. Overall, in this paper we have answered this question by looking into the effectiveness and advantages of procedural dungeon generation methods, and discussed the key limitations and applications for each.

Although not every algorithm discussed was implemented, we have managed to discuss the methodology behind some of the core algorithms. However, an improvement to the project would have been to implement all of the described algorithms within the software, as evolutionary cellular automata was not able to be implemented. The existing algorithms could also have been tuned more to allow for more accurate testing, as the software is rudimentary and has a lot of limitations to how an output can be used. Additionally, the inclusion of the additional algorithms discussed in section 3.6 would have been an improvement to the project, as although a large portion of the major procedural dungeon generation techniques were discussed, some techniques such as generative grammars did not get a full description.

4.2 Personal Analysis

Overall, I would consider this project to have been a success. I set out to investigate the various methods of procedural dungeon generation, and in this paper have discussed a large variety of available algorithms and provided insights on their applications. However, more information on large fields such as generative grammars and their applications within dungeon generation would have provided a useful addition and allowed for more depth of discussion.

A significant element that could have been improved to increase the quality of the report is the software used to test algorithms and provide results. Although

four of the methods discussed in this paper, CA, BSP, agent based, and constraint based, were included in the software, not every element was as tuned as I would like, for example including delauney triangulation and minimum spanning tree's would have improved the quality of discussion available for the constraint based algorithm a lot. It also does not have support for evolutionary cellular automata and this is another element that was missing from the project. The software is a significant element that I feel did not match the expectations I had set for it initially, and if I was to attempt this project again, this is an element I would focus more time on and ensure it allowed for quality results for each discussed method. Both time management and project management could have been improved throughout the project as well, as there were periods of time where the direction of the project was unclear, and a more focused project plan would have allowed more time to focus on the important and time consuming aspects of the project, such as the software.

Additionally, the way the algorithms were going to be evaluating changed form multiple times over the course of the project. Initially, the idea behind the evaluation of algorithms was going to be a heavy focus on the quantitative differences behind each algorithm, for example the number of rooms created, dead ends, scalability, and additional information surround special rooms and elements such as treasure rooms, monsters, or barriers. However upon further research and investigation into the various algorithms, this idea had to be scaled back as it became clear that comparing purely quantitative elements of the outputs will be difficult due to the clear differences in generation and what style of map each algorithm created, for example the difference in output between the cave-like cellular automata and the structured BSP rooms and corridor approach, makes direct comparison of quantitative elements challenging. As an alternative approach to analysis, the project moved in a slightly different direction, choosing to include a qualitative analysis of each algorithm focusing more on factors such as perceived difficulty, level variety, and how engaging a level is to play, using results from a survey sent to the public. However, this was also unable to be included due to time constraints and a lack of support from the software with gameplay elements in order to properly test this. Based on these two approaches, in the end a hybrid approach between the two was included, discussing algorithms with a game developer in mind, and focusing on elements such as how much control an algorithm gives the designer and simplicity of use. This was an approach I was happy with, and feel allows for discussion of the key differences between approaches.

4.3 Future Work

Following on from this project, future work could be done to expand upon what was discussed and to look further into the effectiveness of procedural dungeon generation. One key piece of work that could be improved is the software, as implementation of all generation methods would allow for direct comparison of results, instead of relying on the works of others in the literature.

Further research into the application of evolutionary algorithms for procedural dungeon generation could also yield interesting results. Looking into the agent based method and how it can be improved is another field that could prove insightful. Finally, the 3D application of procedural dungeon generation techniques has not been explored in as much depth as 2D, and research into this with techniques such as octree space partitioning may have good results.

Additionally, a key concept that was not covered within this paper was that of filling a dungeon with features such as special rooms, monsters, keys, and locks. This was discussed early on in the paper, however implementation of this was not covered. It is a large area of the dungeon generation process, and discussion of how these features could be procedurally implemented would allow for a whole new avenue of discussion and evaluation within the field.

References

Adams, D. (2002, 01). Automatic generation of dungeons for computer games.

Adonaac, A. (2015, Mar). *Procedural dungeon generation algorithm.* Retrieved from https://www.gamasutra.com/blogs/AAdonaac/20150903/252889/Procedural_Dungeon_Generation_Algorithm.php

Ashlock, D., Lee, C., & Mcguinness, C. (2011). Simultaneous dual level creation for games. *IEEE Computation Intelligence Magazine*, 26-37.

Basic bsp dungeon generation. (n.d.). RogueBasin. Retrieved from http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation

Bellos, A. (2014). *The game of life: a beginner's guide.* Retrieved from https://www.theguardian.com/science/alexs-adventures-in-numberland/2014/dec/15/the-game-of-life-a-beginners-guide

Benard, S. (2017, Mar). *Building the level design of a procedurally generated metroidvania: a hybrid approach.* Retrieved from https://www.gamasutra.com/blogs/SebastienBENARD/20170329/294642/Building_the_Level_Design_of_a_procedurally_generated_Metroidvania_a_hybrid_approach.php

Berlin interpretation. (n.d.). RogueBasin. Retrieved from http://roguebasin.roguelikedevelopment.org/index.php?title=Berlin_Interpretation

Bermejo, A. P. (2018). *Design and development of a roguelike game with procedurally generated dungeons and neural network based ai.*

Blomberg, J., Jemth, R., Lennar, A., Lilius-lundmark, R., Petterson Johnsson, M., & Svensson, T. (2018). An Exploration of Procedural Content Generation for Top-Down Level Design.

Coello, C. A. C. (2005). An introduction to evolutionary algorithms and their applications. *Advanced Distributed Systems Lecture Notes in Computer Science*, 425–442. doi: https://doi.org/10.1007/11533962$_3$9

Dahlskog, S., Bjork, S., & Togelius, J. (2015). Patterns, dungeons and generators. *Foundations of Digital Games*.

Darwin, C. (1869). *On the origin of species.*

De Jong, K. (2006). Evolutionary computation: A unified approach. *GECCO 2020 Companion-Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 327-342.

donjon; random dungeon generator. (Copyright 2009-2020). Retrieved from https://donjon.bin.sh/fantasy/dungeon/

Dormans, J. (2010). Adventures in level design: generating missions and spaces for action adventure games. *PCG'10: Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 1-8.

Dormans, J. (2017, May). *A handcrafted feel: 'unexplored' explores cyclic dungeon generation.* Retrieved from http://ctrl500.com/tech/handcrafted-

feel-dungeon-generation-unexplored-explores-cyclic-dungeon-generation/

Eiben, A., & Smith, J. (2003). *Introduction to evolutionary computing.*

Figueiredo, M., Marcelino, L., & Fernando, T. (2002, 01). A survey on collision detection techniques for virtual environments. *Proc. of V Symposium in Virtual Reality, Brasil*, *307*.

Fuchs, H., Kedem, Z. M., & Naylor, B. F. (1980). On visible surface generation by a priori tree structures. *SIGGRAPH '80 Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, 124–133.

Hartsook, K., Zook, A., Das, S., & et al. (2011). Toward supporting stories with procedurally generated game worlds. *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, 297-304.

Hello Games. (2016). *No man's sky.* Wikimedia Foundation. Retrieved from https://en.wikipedia.org/wiki/No_Man's_Sky

Johnson, L., Yannakakis, G., & Togelius, J. (2010, 09). Cellular automata for real-time generation of infinite cave levels.
doi: https://doi.org/10.1145/1814256.1814266

King, A. (2015, Apr). *The key design elements of roguelikes.* Envato Tuts. Retrieved from https://gamedevelopment.tutsplus.com/articles/the-key-design-elements-of-roguelikes–cms-23510

Kun, J. (2017, Jan). *The cellular automaton method for cave generation.* Retrieved from https://jeremykun.com/2012/07/29/the-cellular-automaton-method-for-cave-generation/

Liapis, A., Yannakakis, G. N., & Togelius, J. (2013). Sentient sketchbook: Computer-aided game level authoring. *Proceedings of the 8th International Conference on the Foundations of Digital Games (FDG 2013)*, 213–220. Retrieved from http://julian.togelius.com/Liapis2013Sentient.pdf

Niemann, M., & Preuß, M. (2015). Constructive Generation Methods for Dungeons. , *49*(176), 1–40.

Pech, A., Masek, M., Lam, C. P., & Hingston, P. (2016). Game level layout generation using evolved cellular automata. *Connection Science*, *28*(1), 63–82. Retrieved from https://doi.org/10.1080/09540091.2015.1130020 doi: https://doi.org/10.1080/09540091.2015.1130020

Pepe, F. (2020). *What makes a good rpg dungeon?* Retrieved from https://medium.com/@felipepepe/what-makes-a-good-rpg-dungeon-505180c69d00

Roden, T., & Parberry, I. (2004). From artistry to automation: A structured methodology for procedural content creation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3166*(September 2004), 151–156. doi: https://doi.org/10.1007/978-3-540-28643-$1_1$9

Serra, J. (1986). Introduction to mathematical morphology. *Computer Vision, Graphics, and Image Processing*, *35*(3), 283-305. Retrieved from https://www.sciencedirect.com/science/article/pii/0734189X86900022
doi: https://doi.org/https://doi.org/10.1016/0734-189X(86)90002-2

Shaker, N., Liapis, A., Togelius, J., Lopes, R., & Bidarra, R. (2016). *Constructive generation methods for dungeons and levels.* doi: https://doi.org/10.1007/978-3-319-42716-4$_3$

Shiffman, D. (n.d.). *The nature of code.* Retrieved from https://natureofcode.com/book/chapter-7-cellular-automata/

Stout, M. (2012). *Learning from the masters: Level design in the legend of zelda.* Retrieved from https://www.gamasutra.com/view/feature/6582/learningfromthemasterslevel.php

Togelius, J., Shaker, N., & Nelson, M. (2016). *Procedural content generation in games.*

Toth, C. D. (2005). *Binary space partitions: Recent developments* (Vol. 52). MSRI Publications.

Toy, M. C., & Arnold, K. C. (1980). A guide to the dungeons of doom. Retrieved from https://docs.freebsd.org/44doc/usd/30.rogue/paper.pdf

Valtchanov, V., & Brown, J. A. (2012). Evolving dungeon crawler levels with relative placement. *ACM International Conference Proceeding Series*(June 2012), 27–35. doi: https://doi.org/10.1145/2347583.2347587

Van Linden, R. D., Lopes, R., & Bidarra, R. (2013). Designing procedurally generated levels. *AAAI Workshop - Technical Report*, *WS-13-20*(October 2013), 41–47.

Viana, B. M. F., & Dos Santos, R., S. (2019). A survey of procedural dungeon generation. *Brazilian Symposium on Games and Digital Entertainment, SBGAMES*, *2019 October*, 29-38.

Wainer, G., Liu, Q., Dalle, O., & Zeigler, B. P. (2010). Applying cellular automata and devs methodologies to digital games: A survey. *Simulation and Gaming*, *41*(6), 796–823. doi: https://doi.org/10.1177/1046878110378708

Wikipedia. (n.d.-a). *The binding of isaac (video game).* Retrieved from https://en.wikipedia.org/wiki/The_Binding_of_Isaac_(video$_game$)

Wikipedia. (n.d.-b). *Diablo (video game).* Retrieved from https://en.wikipedia.org/wiki/Diablo_(video$_game$)

Wikipedia. (n.d.-c). *Rogue (video game).* Retrieved from https://en.wikipedia.org/wiki/Rogue_(video_game)

Williams, N. (2014). An Investigation in Techniques used to Procedurally Generate Dungeon Structures. , 1–60. Retrieved from http://www.nathanmwilliams.com/files/AnInvestigationIntoDungeonGeneration.pdf

Wolfram, S. (2002). *A new kind of science.* Wolfram Media. Retrieved from https://www.wolframscience.com

Wong, K.-C. (2015, 08). Evolutionary algorithms: Concepts, designs, and applications in bioinformatics. *Nature-Inspired Computing: Concepts, Methodologies, Tools, and Applications*.

Zapata, S. (2017, Nov). *On the historical origin of the "roguelike" term.* Retrieved from https://blog.slashie.net/on-the-historical-origin-of-the-roguelike-term/

Appendices

Appendix A

Project Overview

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: An Investigation into Procedural Dungeon Generation

Overview of Project Content and Milestones

The project will focus on looking at different methods to procedurally generate dungeon layouts, with a specific focus on the technology for use in Roguelike/Roguelite video games. A roguelike video game, in this instance, is described as a sub-genre of role-playing video games which typically contains procedurally generated levels, permanent death, and grid-based graphics for levels. Roguelites are very similar to Roguelikes, but with a focus on carrying progress over from one run to the next. From here on in, both terms will be referred to as "Roguelikes" for ease of use. These typically use procedurally generated dungeons as their levels, and such will be the focus for the project. There will be a descriptive review of algorithms and methods already in use, and research into how these things affect finished dungeon layouts and use experience. In addition, a sample software will be delivered with which to showcase some of the algorithms and techniques talked about. The methods will be compared to each other, and conclusions drawn regarding the most successful techniques.

1. To Research into various methods of Procedural Dungeon Generation (PDG) and compare how each may be used in a practical setting. Additionally, to investigate what makes a dungeon successful and how.

2. To Develop a sample software to test and showcase some of the methods and algorithms discussed.

3. To Compare and draw conclusions based on research and practical tests.

The Main Deliverable(s):

- A review of multiple famous and well used techniques used for dungeon generation, in particular for Roguelike video games.

- A comparison of how these techniques are used, and how successful they are compared to each other based on varying criteria and setting.

- A software showcasing varying algorithms in practice, with which we can see how each produces different layouts as well as the scope for customization.

- A review of the results from tests done by groups of people, and how successful they viewed the software and algorithms in practice.

The Target Audience for the Deliverable(s):

- Researchers working with procedural content generation algorithms

- Game Developers working with/on Roguelike games

The Work to be Undertaken:

- Investigation into different procedural generation algorithms and techniques.

- Investigation into what makes a dungeon successful, and the core elements that need to be included.
- Discussion on success and use of algorithms.
- Comparison between algorithms and techniques for varying requirements and settings.
- Design of a basic framework to showcase dungeon generation in action.
- Implementation of algorithms within framework to produce a visual representation.
- Testing of algorithms within framework based on conclusions drawn previously.
- Evaluation of success through survey-based approach, using feedback on software use and algorithm results.

Additional Information / Knowledge Required:

For this project, there will be a variety of additional information needed. I will need to firstly investigate various techniques and algorithms commonly used in Procedural Content Generation and acquaint myself with how they are used in practical settings. Then, I will need to find papers on more detailed techniques and information regarding the history and future of the subject. Additionally, I will need to most likely learn some new skills to create the sample software using Unity and implement some of the algorithms within it.

Information Sources that Provide a Context for the Project:

https://journal.stuffwithstuff.com/2014/12/21/rooms-and-mazes/ - This site briefly explains a simple implementation, but the What's in a dungeon? part is specifically interesting as it talks about the core elements of dungeon design.

https://www.researchgate.net/publication/260800341_Procedural_Generation_of_Dungeons - This paper is of specific interest to the project as it goes into more detail on certain algorithms and techniques, as well as providing a useful background for Proc Gen in general.

https://www.semanticscholar.org/paper/A-Logical-Approach-to-Building-Dungeons%3A-Answer-Set-Smith-Bryson/f69bf76b77da89bfd7135b9c60d2b9b10fc1ac20#citing-papers – Again, a little background on Proc Gen and discussion on Answer Set Programming in dungeon generation.

https://www.researchgate.net/publication/288320122_Designing_procedurally_generated_levels - A look into a Grammar Based method of generating dungeon layouts.

https://dl.acm.org/doi/abs/10.5555/3144605.3144638 - Discussion on Tile based span and growth procedural dungeon generation.

https://ieeexplore.ieee.org/abstract/document/8924832 - A recent conference paper, exploring the current state of PDG (Procedural Dungeon Generation). Talks about which areas are underdeveloped, and the state of PDG at the moment.

https://www.teses.usp.br/teses/disponiveis/55/55134/tde-25032019-144917/en.php - A masters dissertation from a Portuguese student on a similar area, using an evolutionary algorithm.

http://pcg.wikidot.com/pcg-algorithm:dungeon-generation – Procedural Content Generation wiki, has lots of useful resources and a short description of how dungeon generation is typically done, as well as linking to a lot of other useful sources.

The Importance of the Project:

This project is of some significance, Roguelike games have been increasingly popular ever since their inception and the ways that they have done proceduralily generated content is vital to the success of the game. The topics covered will not be of new technology, but rather giving insight and comparing existing methods, as well as discussing suitability for purpose and ease of use.

The Key Challenge(s) to be Overcome:

- Research and gather information on what makes successful procedurally generated dungeons.
- Research and gather information on PDG methods and papers surrounding them.
- Collate and write up review using sources previously gathered, exploring the circumstances you may use certain techniques and how methods have evolved recently.
- Create framework for basic software to test algorithms and methods.
- Test different PDG methods using software and gather test results.
- Evaluate success of tests compared to each other, based on results gathered from feedback of groups of people using the software and examining algorithm outputs.

Is the project of the required standard? (if not, what changes are required?)	Yes, I think this will project is of a good standard, and has the possibility to be a little overcomplex depending on the number of algorithms chosen. My only comment would be to make sure there is some development work undertaken by the student and it isn't all just library use. (I don't think this would be the case, however!)
Is the project viable? (if not, what changes are required?)	Yes
Does the project provide a realistic challenge for the student? (if not, what changes are required?)	Yes, I believe so.
Is the project appropriate for the student's programme? (consult project guidance - if not, what changes are required?)	Yes, very!
Is the student adequately supported? (if not, what changes are required?)	Yes
Any other feedback?	I have two small comments: 1. Picking a list of algorithms quickly would be sensible, with perhaps a narrowing down of two techniques to compare first, to make sure you have a achievable minimal viable product for writing the honours project! 2. How the evaluation will take place is somewhat vague at the moment. I suspect this might be the challenging part, so attention should be given to what method(s) will be used (qualitative vs quantitative etc) sooner rather than later! In particular, if you are gathering feedback from users, I'd see if there are any appropriate pre-written/designed surveys you can use!

Appendix B

Diary Reports

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 22/09/2020 **Last diary date: N/A**

Objectives:

- First project meeting
- Discussion on suitability of project
- Ensure correct documentation is submitted so far
- Discuss what is needed for project start up, from a technical and literature standpoint
- Look for pointers on recommended reading

Progress:

- Basic game created over summer
- Initial idea thought up
- Project has been registered with uni
- Supervisor and Second marker assigned

Supervisor's Comments:

Project seems suitable, needs fine tuning into specifics.
Specifics include what algorithms and/or techniques are relevant and can be included, how in depth the project is going to focus on the technical side and how much on the comparison and evaluation instead. Documentation is good so far, need to start thinking about and filling in Initial Project Overview, should give better idea of project as a whole when filled in.
Gave a couple of links to interesting articles and GitHub repositories.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 06/10/2020 **Last diary date: 29/09/2020**

Objectives:

- Ensure IPO was submitted successfully
- Ensure project is on track
- Discuss moving into creation of software, and what languages/libraries may be suitable

Progress:

- Discussion on techniques such as Cellular Automata
- Collating papers and sources
- Background reading

Supervisor's Comments:

IPO is in a good state, some small tweaks but nothing major. Project is on track and on time. Moving into software, options could include using Unity as an easy to use graphical interface displaying images, potentially from c# or an external c++ program. Recommended looking into graphical options for c++ as there are some good options.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 13/10/2020 **Last diary date: 06/10/2020**

Objectives:

- Go over IPO feedback from second marker
- Discuss plan for dissertation, word document
- Ask questions about various topics, such as intellectual property and algorithm suitability
- Discuss plan for coming week, what to prioritise

Progress:

Made a draft for chapters and overall contents of the dissertation.
Written up a small amount for introduction on background to Roguelikes
Received feedback from second marker on IPO

Supervisor's Comments:

IPO feedback is good! Only real comment/change is the evaluation section, it is still too vague and needs properly defined. Such as with previous weeks, discussed some ideas for evaluation and to take ideas from other papers.
Plan for dissertation, for the most part, is good. The sections need tightening up a little, some sections are still a bit vague on what they may contain. In addition, the methodology and implementation sections may be a bit too similar to each other, could possibly roll them into one section.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 20/10/2020 **Last diary date: 13/10/2020**

Objectives:

- Discuss beginnings of lit review and what is required
- Review plan for dissertation in case anything needs changing
- Discuss introductory code for procedural dungeon generation application

Progress:

Have started writing dissertation properly, and have more on the background of roguelikes written up.

Started work and looking into creation of software, decided on using c++ for the backend and most likely an integrated graphical interface for ease of use.

Started creation of a basic dungeon generation implementation, simple cave generation using drunkard walk. In working state, but simplistic.

Supervisor's Comments:

Background on roguelikes is looking alright, but probably belongs in the introduction/background rather than literature review. Perhaps focus more on the more technical side in future weeks, more important to get this out of the way.

Software creation is off to a good start, good choice using c++ as is lightweight, perfect for games. Recommended imggui as a library for graphical interface.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 27/10/2020 **Last diary date: 20/10/2020**

Objectives:

- Review plan and see if additional changes are needed at this time.
- Go over added detail to literature review, check quality.

Progress:

More in depth more with comments taken on board drafted.

Literature review added more detail, descriptions on how EA's and Cellular Automata work have been added.

Supervisor's Comments:

Draft plan looking much better, but still need to tighten up sections and the difference between methodology and implementation.
Written work, while good enough descriptions, perhaps should not be in the literature review but in the method section, as they are maybe a bit too technical? Keep it simpler in the literature review and more high level.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 03/11/2020 **Last diary date: 27/10/2020**

Objectives:

- Review full draft paper in preparation for interim review, get feedback on paper as a whole
- Discuss the sections and chapter headings, now there is writing for initial sections, potentially change things around.
- Get feedback on new sections added, such as dungeon taxonomy.

Progress:

Switched dissertation over to LaTeX for ease of writing and better presentation.

Fleshed out the introduction section a lot more, added key information from the IPO such as aims and objectives into dissertation.

Added more writing to the literature review and paper as a whole, detail on dungeon taxonomy, definitions, and descriptions of algorithms added.

Supervisor's Comments:

Paper is much better than previous weeks, main feedback is on the literature review section. Suggested using the 5W system, and keep it simple in this section, moving other things to introduction or method.
Can potentially roll methodology and implementation into one section.
Feedback given on paper, including small comments such as spelling, wording, and more.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 10/11/2020 **Last diary date: 03/11/2020**

Objectives:

Discuss additions to sections, information that has been added to blank sections and assess suitability.
Go over referencing, how to reference some of the more niche websites, and sites with no author, date like Wikipedia. Also go over referencing with images, and how best to reference images in figures.
Go over overall layout, might need to move some sections around or delete if they are obsolete.

Progress:

Have acted on comments from last weeks draft, made small changes, fixed typos etc.

Added information to each blank section, detailing what will be included in that section and how it ties to the rest of the project.

Added referencing file to Latex, now set up for future use.

Added additional information to introduction/context.

Changed the layout of some sections in the project to read better.

Supervisor's Comments:

Few comments on the draft, with the "rooms" section in methodology, need to clear up its purpose and what is actually discussed there.
On same topic, linking between sections and introducing sections in the previous one is a good idea, paper will flow better as a result.
Two other sections, Cellular Automata and Filling the Dungeon in the method section can be moved, they are not quite right where they are now.
Gone over latex code to reference images using ref and labels, and gave pointers on referencing images and quotes.
In future, when on the practical side, look into identifying shared classes and libraries for ease of use, and look into things such as pathfinding and noise functions, alongside other useful techniques.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 17/11/2020 **Last diary date: 10/11/2020**

Objectives:

Plan for interim review, do you fill in the document as it happens with comments etc, similar to diary stuff?

Thoughts on algorithms based on popular games, e.g the way Binding of Isaac does its generation, rogue's implementation etc. Worth including, can I talk about it etc.

For survey stage, is there any official university protocol or can I setup a survey website, send out the application and questions, and be okay? Also data protection, I assume keep everything anonymous

This semester, what should I be aiming for in your opinion

Posters??

Progress:

Supervisor's Comments:

Enquired about Posters
End of this trimester, done with dissertation - Know what you are going to do, hone in, implement and write what has been planned.
If you plan everything ahead and execute everything, bad idea. Don't use waterfall, work as you write.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 24/11/2020 **Last diary date: 17/11/2020**

Objectives:

Clarify research question, how much detail this should be.
Go over viability of some aims/objectives

Progress:

Supervisor's Comments:

Gave feedback on research question and link to web page with additional information.
Advice on aims/objectives.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 01/12/2020 **Last diary date: 24/11/2020**

Objectives:

Clarify term dates and how long project will last.
Receive feedback from draft of paper sent to supervisor.

Progress:

Draft sent to supervisor for feedback.
Added information on the evaluation and how this will be approached.
Changed some formatting and the order of sections, will need feedback on this.

Supervisor's Comments:

Discussed term dates.
Sent feedback back from draft, specifically focus on the evaluation and tighten up what you are doing, gave notes on some of the criteria and ideas to improve it.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 08/12/2020 **Last diary date: 01/12/2020**

Objectives:

Not much to do discuss this meeting due to external commitments.

Progress:

This week not much achieved due to external coursework commitments, next week will send draft to receive feedback on methodology section.

Supervisor's Comments:

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 15/12/2020 **Last diary date: 08/12/2020**

Objectives:

Go over comments from draft feedback sent.
Ask about repetition and how to avoid this within the paper.
How best to deal with positioning in latex, as figures are moving out of place

Progress:

Lots of literature review progress, added sections on generative grammars, space partitioning, and agent based. Added to evaluation section but still unclear of what to include here.

Supervisor's Comments:

Advice on literature review sections given, focus more on BSP and not space partitioning.
Give a heads up on where agent method fits in to the grand scheme of things, focus more on description of how it works and what it does than evaluation at this point.
Add figures! Images help convey ideas better than text.
Answered questions on referencing, reference company website rather than something like steam page.
Split content up into 3 sections, lit review, methodology, and evaluation. Gave advice on where certain elements would fit into the paper and what category they would fall into.
Don't focus on latex figure positioning etc at the moment, do this at the end.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 19/01/2021 **Last diary date: 15/12/2020**

Objectives:

Return from Christmas break, discuss work done on the code and next steps going forward

Progress:

Basics of the software created over Christmas break, basic CA implemented and initial look into BSP implementation.

Supervisor's Comments:

Code looks good so far, just watch time management to ensure everything you would like to get impelemented can be done.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 26/01/2021 **Last diary date: 19/01/2021**

Objectives:

Not much to be discussed this meeting.

Progress:

Work made on code this week; basics of space partitioning done.

Supervisor's Comments:

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 02/02/2021 **Last diary date: 26/01/2021**

Objectives:

Discuss best method for viewing the software, currently using SFML but potentially not the best. Also discuss what I would like to include in the software overall once it is finished.

Progress:

Additional work on code made, space partitioning nearly finished and basics of the agent based method implemented. Cellular Automata also mostly finished.

Supervisor's Comments:

Look up using Imgui, could be useful for adjustable parameters.
Plan looks solid as far as what is going to be included in software, but look into EA's.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 09/02/2021 **Last diary date: 02/02/2021**

Objectives:

Lots of small aspects of the code to go over, potential fixes and issues so far. Look at strange dead ends and diagonal corridors being created,
Review all current results with supervisor to receive feedback and advice on what next steps should be.

Progress:

Agent based basics are implemented, spatial partitioning almost finished, cellular automata almost finished.

Supervisor's Comments:

Number one advice is to animate the generation for dungeons, implement a step function of sorts to step through the algorithm steps as it created in order to see how the process as a whole comes together.

Potentially make a player agent that could move through the maps and ensure everything is connected, could be useful for evaluation as well.

Keep fixed seeds so that you can adjust parameters and see their direct impact on results, hard to tell impact with new seed each generation.

Build all parts of the code with evaluation in mind, also focus on the dungeon from a players perspective, where are the start and end points?

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 16/02/2021 **Last diary date: 09/02/2021**

Objectives:

Look into best way to build survey, does Napier have support?
Discuss evaluation, is qualitative and quantitative a good idea or

Progress:

Progress on software, spatial partitioning finished, cellular automata finished, agent-based needs work still.

Supervisor's Comments:

Napier does have survey site, worth looking into alternatives.
Analysis of both elements is good approach, be aware of issues that may come up when using survey though.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 23/02/2021 **Last diary date: 16/02/2021**

Objectives:

Discuss new constraint-based approach, viability, and supervisor thoughts on this.

Progress:

Basic implementation of new constraint-based approach finished. Agent based method has received some improvements.

Supervisor's Comments:

Constraint approach is a good idea to include, be aware of difficulties implementing as original uses physics engine however. Your implementation will be different but should function in a similar way.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 02/03/2021 **Last diary date: 23/02/2021**

Objectives:

Won't be including EA's in implementation,
however field is still worth discussing in paper.
4 main implementations: CA, BSP, Agents, Constraint Based (Bounce)

Rough word count I should be aiming for in paper.
Issues surrounding sending survey out, what should I be aware of.

Progress:

Constraint based implementation mostly finished, but still has issues with separating rooms as no physics engine.

Added basics of a character that can move around environment and spawns in one corner of map.

Supervisor's Comments:

4 main implementations if fine but still include examples of EA's in paper, make sure you can find comparable results.
Discussed rough word count and survey issues that may come up.
Will have access to the survey software arranged for next week.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 09/03/2021 **Last diary date: 02/03/2021**

Objectives:

Discuss work that has been done, constraint based, partial implementation of minimum spanning trees. Still have issues with constraint based and would like advice on where to proceed with this next. Discuss save function and the ability to import custom maps, is this worth including?

Progress:

Partial implementation of MSP's
Constraint based mostly working
Save and load working.
Start and end point finished
Ability to import custom maps is mostly done.

Supervisor's Comments:

Software is looking good on the whole, gave ides on the corridors, linked to library that could be used for delauney triangulation.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 16/03/2021 **Last diary date: 09/03/2021**

Objectives:

Not much to discuss this week, go over timeframe with supervisor and ensure no major issues with time management.

Progress:

Added ability to view outputs with a clean black background, currently not appealing to look at and this will help.

Tweaked and changed some of the BSP code behind the scenes as it wasn't working as intended.

Got files ready for use on other PC's, dll's etc.

Supervisor's Comments:

Looks much better with black background, BSP producing better outputs now.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 23/03/2021 **Last diary date: 16/03/2021**

Objectives:

Discuss comments and feedback on paper.

What best to use for diagrams?

Best way to represent the difference each parameter has on the outputs.

Display multiple images in latex, subfigures?

Progress:

Added a lot to methodology section of paper.

Methods implemented in code now have dedicated sections.

Supervisor's Comments:

Paint.net could be good for diagrams, they can be simple as long as they convey the right message.

Figures are very important to include.

Could potentially include things like time to generate.

Discuss why you have picked the analysis that will be included.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 30/03//2021 **Last diary date: 23/03/2021**

Objectives:

Unfortunately not much to discuss this week due to other coursework commitments.

Progress:

Other coursework commitments meant no notable work could be done this week.

Supervisor's Comments:

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 06/04//2021 **Last diary date: 29/03/2021**

Objectives:

Discuss not using qualitative analysis, potentially not enough time and project may suffer as a result.
Inquire about Viva and the details about it.
Discuss the layout for methodology section and each individual method, if current layout could be improved.
Other formatting improvements that could be made?

Progress:

Methodology section fleshed out and additional information included in literature review.

Supervisor's Comments:

Losing qualitative analysis could be okay, as long as quantitative analysis makes up for it.
Viva details will be dealt with closer to the time, supervisor will organise this.
Layout for methodology could be improved, gave advice on how best to lay this out.
Additionally gave advice on the evaluation as layout is still unsure.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 13/04/2021 **Last diary date: 06/04/2021**

Objectives:

Discuss feedback from draft paper sent to supervisor.

Progress:

Added title page and needed information such as authorship declaration and GDPR declaration.
Added information that was missing from sections such as different tree types for space partitioning. Most results and figures added, and pseudo code included for some methods.
Minimum viable product nearly achieved at this point.

Supervisor's Comments:

Good progress on the methodology, but still missing evaluation and conclusion sections which should be a priority now. Ensure formatting for figures is correct.
Gave feedback from full review of paper, lots of small changes to be fixed and implemented.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Alfie Thomasson **Supervisor: Babis Koniaris**

Date: 19/04/2021 **Last diary date: 13/04/2021**

Objectives:

Final meeting before submission, full draft sent to supervisor and will be discussed.
Mainly would like to go over evaluation contents in preparation for submission.
Also go over requirements for submission, appendices etc to ensure everything that is required to be submitted is correct.

Progress:

Final additions and changes made to paper, evaluation finished and notes on evolutionary cellular automata added. Conclusion is finished and paper is very nearly ready for submission.

Supervisor's Comments:

Nothing major to note, some small changes and feedback but overall paper is looking fine! Would be good to include more evaluation but on the whole it's nearly ready to submit.