Comparing Procedural Content Generation Algorithms for Creating Levels in Video Games

Zina Monaghan
Technological University Dublin

Comparing Procedural Content Generation algorithms for creating levels in video games.

Zina Monaghan

A dissertation submitted in partial fulfilment of the requirements of Dublin Institute of Technology for the degree of M.Sc. in Computing (Advanced Software Development)

January 2018

DECLARATION

I certify that this dissertation which I now submit for examination for the award of MSc in Computing (Data Analytics), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the test of my work.

This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University.

The work reported on in this dissertation conforms to the principles and requirements of the Institute"s guidelines for ethics in research.

Signed: Zina Monaghan

Date: 04/01/2019

ABSTRACT

Procedural Content Generation (PCG) is used frequently in games to increase replayability by introducing variety to playthrough of a game and reduce development time by allowing complex game worlds to be developed by a smaller team over a more limited amount of time.

One common use of PCG in video games is to make randomly generated maps, this allows for the same game to be played multiple times and have a different map to play on each time. There are multiple algorithms that can be used to create both 3D and 2D maps, this essay focuses on five combination of algorithms that create 2D maps on a grid.

These algorithms will be compared for efficiency by measuring the execution time and Big O efficiency measurement. Time efficiency was chosen based on the literature review because in the scenario where it is being used to increase replayability of a game, it will need to be run every time the game is played. In the literature from which the five algorithms were identified, an equation for getting the Big O efficiency measurement was provided for each algorithm.

For this experiment, these two parameters were measured for a variety of map sizes with a fixed room size of 10px in order to test which algorithms were the most efficient at different map sizes. The experiment was then run again using a variety of room sizes to see the affect on the algorithms.

The comparison showed that BSP Rooms and BSP Corridors were the most efficient algorithm combination overall, it had the lowest execution time and its resource use is in the middle of the results for all the algorithms. RRP and DW was the least efficient algorithm combination, with the highest execution time and resource use.

Keywords: Procedural Content Generation, Procedural Generation, games, level design, map design, comparison, 2D maps, PCG algorithms, efficiency, algorithmic efficiency

ACKNOWLEDGEMENTS

I would like to thank Andrea Curley for all her help and guidance throughout this project, her feedback was invaluable and without it this project would not have been possible.

Thank you as well to all the lecturers in DIT for their help during the year and particularly to Luca Longo for his help on phrasing the research question and narrowing my topic.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. INTRODUCTION

This section will outline the area this project aims to explore and discuss what goals need to be achieved in the course of the experiment. This will include discussion of research methodologies and outlining what each chapter in the dissertation aims to cover.

1.1 Background

Procedural Content Generation (PCG) is a method of creating game content algorithmically, an example in games is using predefined seed values with pseudorandom number generators to create maps and decide what items appear on them (Brewer, 2017).

PCG can be used in a variety of ways, such as generating variations of a puzzle for a point and click adventure game so that each playthrough is slightly different (Fernández-Vara, 2014), using it to generate a unique history for the game world (Grinblat & Bucklew, 2017) or using it to generate levels of a game.

This project will focus on the use of PCG to make levels or maps for 2D games. PCG is often used in games to increase replayability. In the case of the algorithms to create levels, the replayability comes from the player having to figure out where they need to go next and having to learn the layout of the level every time they play (Smith, 2017).

For a procedurally generated level to contribute to replayability, each map generated must be different enough that the experience of playing it "feels" different every time. A high variance in the size of maps generated, the amount of rooms on the map and the size and shape of the rooms are factors that can contribute to replayability (Hilliard, ELAarag & Salis, 2017).

Another factor is the path the player takes from the start of the level to the end, this should be different enough each time that the player never feels like they are retreading their steps when replaying the game.

One approach to procedurally generating game levels, also referred to as dungeons, is to use an algorithm such as Random Room Placement (RRP) or Binary Space

Partitioning (BSP) to place rooms and then attempting to connect them via corridors using a separate algorithm such as Random Point Connect or Drunkard Walk (R. Baron, 2017).

Another approach is to use an algorithm such as Cellular Automata or Growing Tree to create corridors, then attempting to unify sections of it into rooms (Johnson, N. Yannakakis & Togelius, 2010). Both approaches use a grid system where the map is split into a series of tiles and the algorithm places rooms or corridors on those tiles.

1.2 Research Problem

This study aims to compare several PCG algorithms for generating 2D maps by measuring the impact map size and room size has on the execution time and algorithmic efficiency of each. It will then use the collected data to identify which algorithms are the most efficient and if there is any link between execution time and algorithmic efficiency.

The research question for this dissertation is:

Which Procedural Content Generation algorithm for generating 2D maps in video games compares best for efficiency?

In order to answer this question the research aims to identify several usable PCG algorithms for creating 2D maps and run comparison. The comparison will consist of creating maps for each algorithm using different room and maps sizes and evaluating the differences in execution time and resource use. It will hope to identify what factors contribute most to efficiency and which algorithm, if any, compares best for efficiency.

1.3 Research Objectives

- Review current literature about uses of PCG in video games and become familiar with common uses and terms.
- Investigate current research done on PCG algorithms used in 2D map generation.
- Identify algorithms for use in experiment and parameters to use in comparison.

- Create multiple maps for each algorithm and measure chosen parameters.
- Run any relevant statistical analysis on results
- Display results in appropriate charts and discuss results
- Identify factors that contribute to or detract from efficiency and compare results for each algorithm.
- Identify limitations of research and if it has any impact on application area
- Discuss further work that could be done in this field, discuss how project could be improved.

1.4 Research Methodologies

To reach the goals set by the research objectives, a literature review will be carried out on previous academic research in to the area of PCG in video games. A review will also be carried out to determine the best software and programming language to use for the experiment.

Each step of the project was run several times and the average was used in the final analysis. Trend lines were used to show the increase or decrease in efficiency across the range of maps. Sample increases at either end of the graph were then compared to see changes in the data over map size. The rate of change in efficiency was also compared to the rate of increase in the map sizes to test the significance of the change.

1.5 Scope and Limitations

The goal of this project is to research PCG algorithms used in making maps in video games, the implementation of these algorithms will focus on making 2D maps. While the algorithms discussed can be used for making 3D maps as well, that is outside the scope of this project.

In order to run the comparison, an application must be created that runs the algorithms and creates maps. This application must allow the user to vary the size of the map and the size of the rooms on the map. The map does not need to be part of a working game and spawning items or other game elements with PCG is not in the scope of this project.

1.6 Document Outline

Chapter 2: A review of previous research done in the area of PCG algorithms in video games, discussing the history of PCG in commercial games and its relevance to the industry. This chapter also looks at specific applications of PCG in video games and reviews the algorithms used to implement them. It discusses evaluation techniques used in the literature that could be applied to the current project and reviews some of the future work proposals from the literature.

Chapter 3: This chapter gives a detailed description of each of the algorithms that will be implemented in this project, it will also discuss the parameters that will be used in the experiment and what limitations will be set on them.

Chapter 4: A chart displaying the average of the results will be provided for each set of measurements taken in the experiment. Alongside this will also be brief description of the trends shown in the chart, including the rate of increase and a breakdown of the changes in efficiency across all map sizes used.

Chapter 5: This chapter will discuss the results of chapter 4 in detail including comparing the results from the different algorithms and discussing possible causes for results. It will also briefly discuss the visual differences in the maps created and attempt to find out which, if any, of the algorithms is most efficient.

Chapter 6: This section will give an overview of the work done in the previous chapters and discuss what the results mean for the research question. It will also attempt to outline further work that can be done in this area and identify improvements that could be made to the current project.

2. LITERATURE REVIEW

“Procedural content generation (PCG) in games refers to the algorithmical creation of game content with limited or indirect user input” (Togelius, Kastbjerg, Schedl, & Yannakakis, 2011).

PCG can be used for a wide range of tasks in video games from creating things the player interacts with directly, such as creating puzzles for the player to solve or enemies for them to fight, to creating a written history of the game world.

PCG algorithms can also be used to make games more efficient by cutting down on memory consumption. For example, in the game Elite, a large map made up of hundreds of start systems was compressed into few tens of kilobytes of memory, by storing the planets as a series of numbers that the algorithm used to create the map when the game was launched (Togelius, Yannakakis, Stanley, & Browne, 2011) .

They can also be used to cut down on development time by using it to do tasks such as placing grass on a map in a realistic pattern or creating variations in trees in an open world game. The rest of this chapter will discuss the history of PCG in video games and the different ways it can be applied. It will also review different ways of evaluating PCG algorithms and look at suggestions for further work in this area.

2.1 Taxonomy of PCG algorithm uses

This section will look at a framework, proposed in a 2014 paper by Smith, for analysing and discussing the use of PCG in video games. The aim of the framework was to address the ways in which PCG algorithms are used to create content and how it affects the experience of the player, with a focus on how the use of PCG increases replayability (Smith, 2014).

The framework found that the main ways PCG is used to add replayability to video game was “*reacting in a surprising environment*” where the game is designed to replayed multiple times to increase progress or beat scores, “*building generator strategies*” where the entire world the game is set in is procedurally generated and “*practising in different environments*” where PCG is used to add challenge to a game by introducing unexpected elements.

Games that do not include elements of "*reacting in a surprising environment*" often rely on the player learning a correct "path" through the game. Adding variations to maps or enemy types using PCG algorithms adds an element of surprise to each playthrough of the game. This affects both how a developer might create a game and how potential players are expected to play it.

Games that rely on "*building generator strategies*" expect the player to experiment with different parameters and difficulties when generating the world, affecting how they will play the game. The game designer must also develop the game with this in mind, making sure that the player can understand and explore the differences between the worlds generated.

Algorithms used to add the element "*practising in different environments*" to a game, often generate a lot of the content without any player input. The challenge of the game to the player is to practice strategies against a variety of generated challenges (environments). The variations added by the PCG algorithms allow the player to encounter similar challenges in different situations, to gain a different view of how to solve them.

2.2 History of PCG in video games

This section will review the history of the use of PCG algorithms in video games, giving examples of the uses of PCG algorithms discussed in section 2.1.

The tabletop board game *Dungeons and Dragons* has a large influence on gaming, and many PCG algorithms are used to emulate its mechanics. The first edition, released in 1974, was played by an estimated 20 million people (Brewer, 2017). All combat and movement in Dungeons and Dragons is determined by mathematical tables, using dice to simulate random number generation. Many of these mechanics were later incorporated into video games using procedural content generation.

Pedit5 is the earliest known computer based role playing game in 1975, it was made on the PLATO System which was the first generalized, computer-assisted, instruction system. The game used a PCG algorithm to generate a character with random Dungeons and Dragons-inspired statistics such as strength (how much damage the character could do in battle) and hit points (how much health the character has). It played from a top-down perspective and allowed the player to move the character through a dungeon encountering randomly generated monsters.

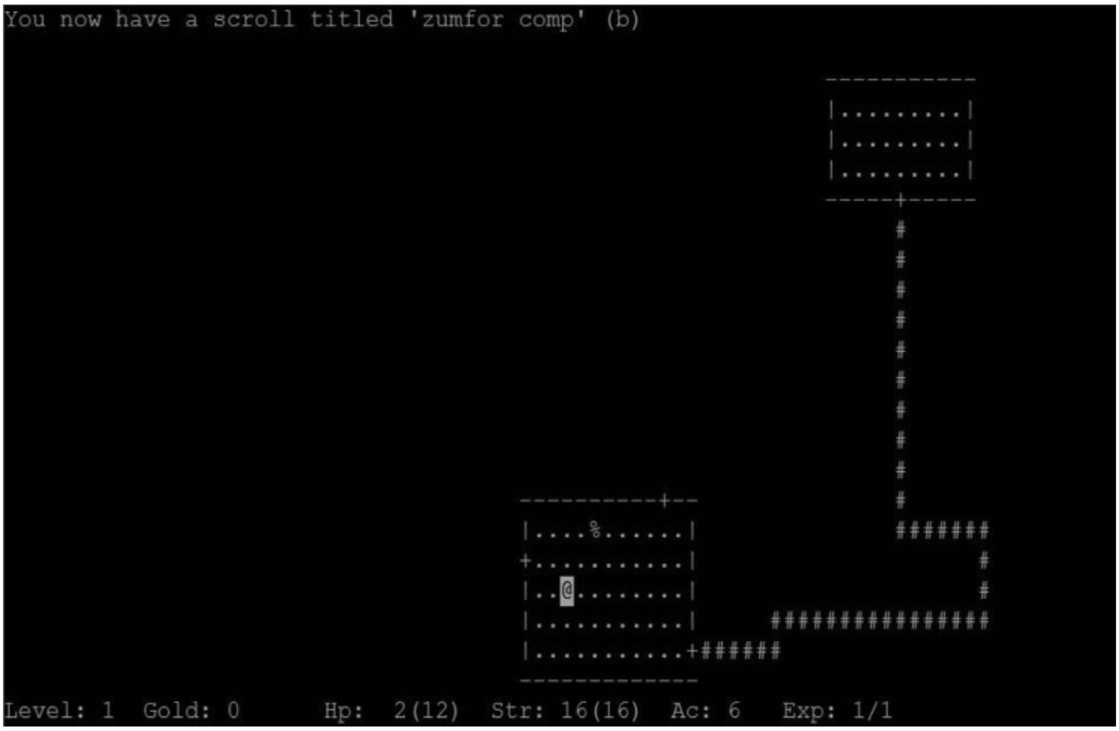

Fig 2.1: Screenshot of the game Rogue, the # symbol represents a corridor in dungeon, the _ and | symbols represent the walls of a room.

The game that popularised the use of PCG is *Rogue*, which was released in 1980 and was directly inspired by Dungeons and Dragons. In the game the player would explore a dungeon created using PCG, attempting to find an amulet. Rogue also used PCG to randomise the properties of objects that could be found in the dungeon and the type of enemies encountered.

Rogue's popularity spawned the genre *roguelike*, which refers to games that use PCG as a core element of gameplay. In 2008 the *International Roguelike Development Conference* defined a roguelike game as one that includes “randomized procedural generation of rooms and items, permadeath, turn-based movement, focus on combat rather than story or plot, resource management, high level of interactivity, and single-player gameplay”.

An example of an influential roguelike game is *Moria*, which came out in 1983. It used PCG to make elaborate cave systems which could span multiple screens, more complex than the dungeon in Rogue which was limited to 80 × 25 lines of text (the amount of text that could fit on a single screen).

The popularity of the internet in the 1990s led to many elements of roguelike games being used in other types of games. For example, Dungeon Crawl games like Diablo and Sandbox games like Minecraft, both of which have large amounts of generated content. Even though these games are not roguelike, they draw inspiration from the genre.

These games use PCG to create a variety of different kinds of content, such as maps, characters and in game dialogue. Some games used PCG to generate almost all aspects of the game, such as *Dwarf Fortress*, which has been in development since 2002. It used PCG to create randomly generated worlds, complete with generated history, lore and races. The use of PCG in games such as *Minecraft* helps increase the replayability of the game by giving the player a different map to explore on demand.

PCG is also be used to speed up development, for example in *Elder Scrolls IV: Oblivion*, an algorithm is used to generate vegetation, even though the map is not procedurally generated. This means the developers do not have to manually place vegetation across the entire map (Hendrikx, Meijer, Van Der Velden & Iosup, 2013).

2.3 Applications of PCG in video games

PCG in video games refers to the algorithmic creation of game content, but the term “game content” is very broad, this section will discuss in more detail what types of game content PCG is most commonly used for, and review some examples of algorithms.

2.3.1 Generating Maps/Levels

One common application of PCG is to use it to create a level or dungeon for a 2D video game. This works by creating a grid of cells, each cell contains the following properties: type (wall or floor), and location (x and y). At the start the grid is all wall cells, and the algorithm will change some cells to be floors based on a predefined set of rules in order to make a dungeon. This can be done in a number of ways.

One approach, used in a 2017 paper, uses two different algorithms to make dungeons. One algorithm is tasked with placing rooms on the map, and the second then attempts to connect the rooms (R. Baron, 2017).

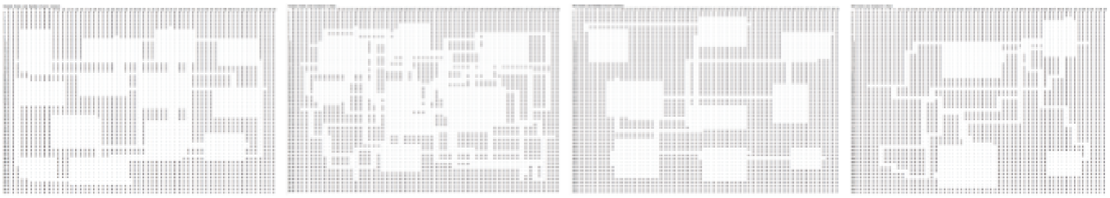

Fig 2.2: Maps created using combinations of rooms and floor algorithms. From left to right - Random Room Placement and Random Point Connect, Random Room Placement and Drunkard's Walk, BSP Room Placement and Random Point Connect, and BSP Room Placement and Drunkard's Walk.

The room generating algorithms used were *Random Room Placement (RRP)* and *Binary Space Partition (BSP)*. RRP is a brute force algorithm that works by generating a room of a random size, then generating random x and y coordinates to place it, it repeats the process until the desired number of rooms is reached. BSP partitions the map in to leaf nodes, stored in a tree data structure, the leaves are then further split until the desired number is reached, each leaf then has a room placed in it.

The corridor generating algorithms used were *Random Point Connect (RPC)*, *Drunkard's Walk*, and *Binary Space Partition (BSP) Corridors*. RPC works by selecting random points on the border of two rooms and attempting to draw a line to connect them between the two points. Drunkard's Walk is an application of cellular automata where the algorithm picks two points on the border of two rooms, similar to the RPC algorithm, then generates lines in random directions from the first point until it reaches the second. BSP corridors only works with the BSP generated rooms and uses the tree data structure to pair and connect rooms.

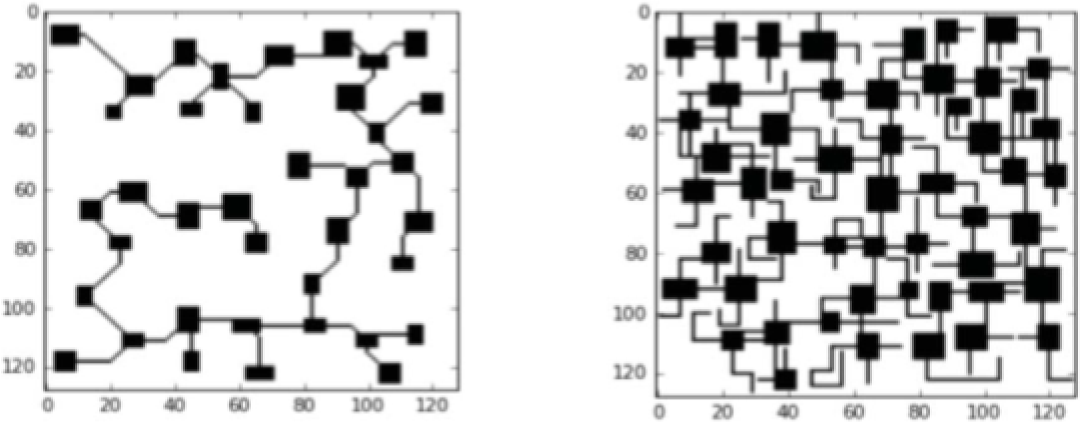

Fig 2.3: From left to right - map produced by Span algorithm, map produced by Growth algorithm.

Another approach to generating a dungeon is called *Span*, it works by first placing a predetermined number of rooms on the map, the rooms are placed randomly, but have a set minimum distance they can be from each other.

It then uses *Prim's algorithm* to determine which rooms are closest to each other and then connects them. Due to the processing needed to search through the list of rooms to find the closest, this can make the algorithm slower at larger map sizes (Hilliard, ELAarag & Salis, 2017).

The same experiment also discussed the algorithm *Growth*, which works by growing the dungeon from a point by adding features, in this case rooms and corridors. It grows each feature from a list of points, if the point contains a room then the feature is a corridor, if the point is a corridor it can add either a room or another corridor.

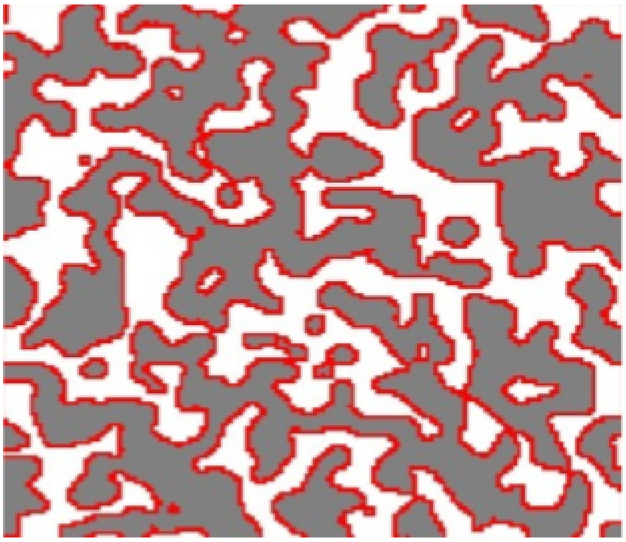

Fig 2.4: Map generated using cellular automata.

Another approach to level generation works by first randomly setting half the cells to be floors, then, using a variation of cellular automata, it goes through the grid and changes the type of each cell based on the number of neighbouring cells which are floors, this process is repeated until a cave like maze is created (Johnson, Yannakakis, & Togelius, 2010).

2.3.2 Generating Puzzles

Another application of PCG is to use it to make variations of puzzles. For the point and click puzzle game *Symon*, PCG was used to make the relationship between objects in the game and the puzzles they are used to solve be slightly different each time. The game aims to have a “dream like” atmosphere, which is complimented by the randomness of the puzzle generation (Fernández-Vara, 2014).

The game was designed so that when a puzzle was solved, it would provide information on how to solve a different puzzle. A puzzle map, shown in the Fig 1. below, was created that established the relationship between the puzzles.

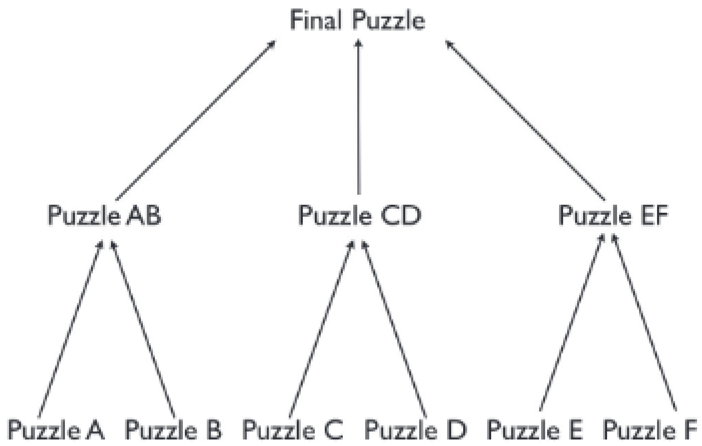

Fig 2.5: Sample Puzzle map of the adventure game Symon.

The PCG system placed puzzle patterns into the map so that the outcome of one puzzle would unlock the solution to a different puzzle. It did this by trying different combinations of puzzle patterns on the map until one fitted (Fernández-Vara & Thomson, 2012) .

The idea for this structure was inspired by the GRIOT system, which generated poetry procedurally by allowing the user to define the structure of stanzas and topics, then generating sentences using the structure. This increases replayability by giving the player new challenges every playthrough, preventing them from easily learning off the puzzle solutions (Smith, 2014).

2.3.3 Generating World History

PCG can also be used to generate text and other more abstract things about a game world, such as its history. The science fiction fantasy roguelike game *Caves of Qud* generates the history for the game world each time the player loads a new game (Grinblat & Bucklew, 2017).

2.3.3.1 Generating historical events

The history is generated in five periods, each ruled by a randomly generated sultan. Each period consisting of several generated historical events. For each event, a descriptive text snippet.

The players read this history from gospels and the descriptions of paintings and shrines in the game world. This means that the style of the writing in the text snippets must have a suitable tone for the way the player interacts with it, for example the description of event found in a gospel would be tonally different to the description of a painting of the event.

This system models the history as the interaction between historical entities such as places, items and important people, and historical events that modify the properties of those entities, as shown in Fig 2.6

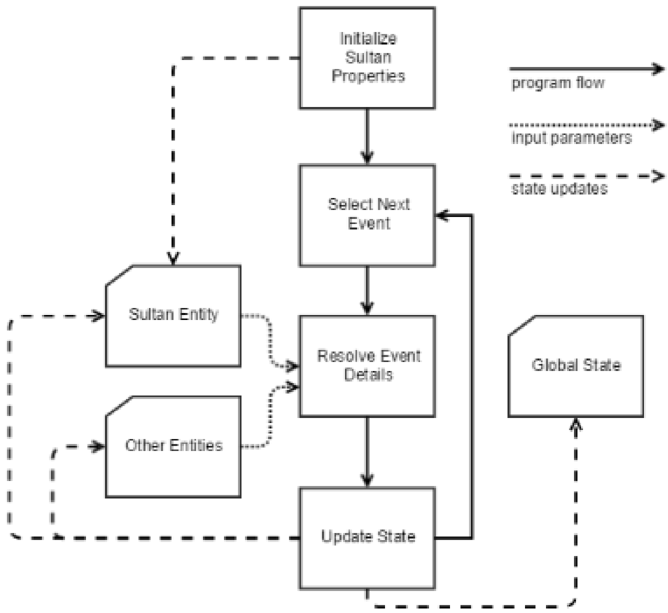

Fig 2.6: Flow diagram for the generation of a sultan's history in Caves of Qud.

The system first describes the current state of a sultan by generating some properties for them such as name, birth date and region of birth. It then chooses an event to happen to the sultan, for example the siege of a city.

The outcome of the event is based on the sultan's current state and random branching It them modifies the sultan's properties and the properties of any other entities connected to the event. The system them chooses more random events in the same fashion, each one updating the properties of the sultan and game world and generating a text snippet explaining the event.

2.3.3.2 Generating historical text

The text explanations for events in Caves of Qud are generated based on the sultan's state. For example, if the event was to besiege a city, the text snippet might give the explanation.

"*Acting against #injustice#, #sultanName# led an army to the gates of #location#"*

The injustice is then replaced with a reason based on the sultan's current state. For example, if the sultan was allied to a race of sentient frogs, the injustice might be "the persecution of frogs."

If no reason can be found, one is randomly generated and the state of the entities around the sultan will be updated to match it. So in the above example, the properties of the ally would be altered to contain frogs.

2.4 Evaluating PCG algorithms

This section will discuss the different possible metrics used for evaluating PCG algorithms. Since the focus of this research is on PCG map making algorithms, this section will focus on methods for comparing and evaluating generated maps, with a focus on efficiency.

In a 2017 paper, two sets of maps were created by generating rooms and corridors using two different corridor algorithms. The first algorithm was called *Span* and used Prim's algorithm to find the minimum distance between rooms, the second was called *Growth* and works by growing the corridor from a list of points until the list runs out.

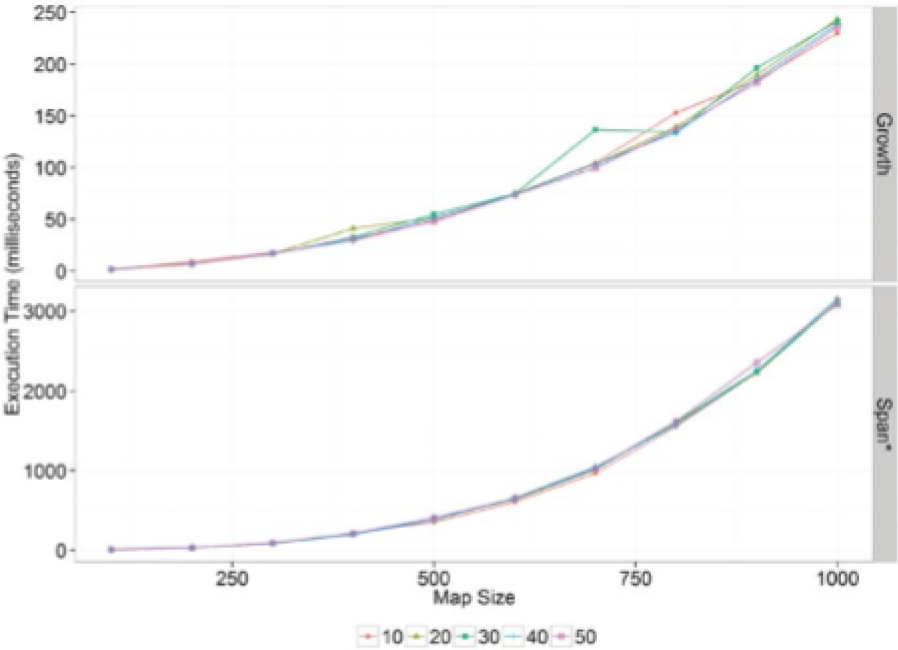

Fig 2.7: Graph comparing execution time of Span and Growth using different map sizes.

These algorithms were evaluated under a number of parameters. First, the execution time was compared to the map size, which found that the Span algorithm took much longer due to the higher computational costs associated with working out the shortest distance between rooms.

Next the experiment was run again using a variety of different room sizes, to see how room size affected the performance, it was found that the size of the room also adversely affected Span the most.

The number of rooms generated by each algorithm was also measured for a variety of map sizes, to ascertain which could get higher average room counts, and which was most affected by map size. It found Growth could achieve a higher room count, whereas Span often struggled to find suitable points on the map to place rooms. Span was also more affected by room size than Growth in the experiment.

In their experiment, R. Baron combined room and corridor placing algorithms and used them to generate both 2D and 3D maps, to see if the algorithm could be applied to both types of games, making it have a wider variety of uses. They used *Big O notation*, which analyses the resources used by an algorithm to determine how productively they are used.

For example, when applied to the room placement algorithm RRP, the researchers identified the maximum size of the rooms to be the main resource, and came up with the formula *O (max_room_width * max_room_height)*. This can similarly be applied to other PCG algorithms to measure their efficiency.

2.5 Research Summary / Conclusion

The main use of PCG in games is to increase their playability, this can be done in many different ways. Minecraft uses it to generate the map, allowing the player to constantly discover new areas, Symon uses it to generate the puzzles, meaning that the user has to solve new ones every time they play.

However, the developers of Symon make that point that PCG is not a solution to making a game replayable or interesting, but a tool to help it come about. Games require good ideas and interesting mechanics outside of the PCG engine to be replayable.

2.5.1 Future Directions

Improvements could be made on games similar to Caves of Qud by increasing the variety of events that can happen and allowing the events to directly affect entities other than the sultan. For example, if there were events that affected the properties of locations and items, such as natural disasters or wars between minor factions in the society, this would broaden the scope to make the history of the world about more than simply the life of the ruler (Grinblat & Bucklew, 2017).

The limitation to using PCG for storytelling systems is that the randomness of the content often makes the characters and scenarios created seem unrealistic or uncompelling. This is why this type of content is not used in large commercially successful games currently, more research could be done in this area (Hendrikx, Meijer, Van Der Velden & Iosup, 2013).

Another area of research, highlighted by the developers of the game Symon, was the need for better tools to help incorporate PCG mechanics into point and click games.

This means creating software to help developers create procedurally generated puzzles (Fernández-Vara, 2014).

R. Baron suggests running the experiment of combining room and corridor algorithms again with a greater variety of algorithms. They also suggests adding other features generated by PCG to the maps, such as items or enemies.

3. DESIGN AND METHODOLOGY

For this experiment the maps will be represented by a square grid of tiles. Each tile contains a status property that can either be "on" or "off", the maps generated are the type used in 2D top down games. On tiles represent floors on the map, and off tiles represent walls.

A room is a section on the grid that consists of on tiles and will be a square or rectangular shape, the width and height of the room is controlled by minimum and maximum values. A corridor is section of the map consisting of on tiles whose width or height must be 1, which will be generated after the rooms have been placed on the grid.

A room cannot be directly touching another room and must connect to at least one corridor, and each corridor must connect to at least one room.

This section will discuss the algorithms used to place the rooms and corridors on the grid and what parameters should be used to constrain them, for example the size of the map and the minimum and maximum sizes of the rooms.

3.1 Algorithms

3.1.1 Room generation

Rooms on the map are represented by rectangles of varying sizes, these algorithms for generating the rooms need to decide on the width and height of the room as well as its location on the map. These algorithms need to be paired with corridor generating algorithms to create a completed map.

3.1.1.1 Random Room Placement (RRP)

Random Room Placement (RRP) is a brute force algorithm that starts by randomly generating a room of a random width and length, the randomness of the size of the room is controlled by maximum and minimum allowed lengths.

Next the algorithm picks a random point on the grid to be the bottom-left corner of the room, the algorithm will only choose a point that is far enough away from the top and right edges of the map, to guarantee that the room will fit. The algorithm will then check that the room is not intersecting any floor tiles on the map, if a floor tile is found, the algorithm will attempt to generate another point to place the room on the map, this process is shown in Fig 3.1.

This processes is repeated until either the desired number of rooms have be placed or the maximum number of attempts (tolerance value) is reached. The successfully placed rooms will be stored in a list in the order they were placed on the map, to be used by a corridor algorithm to connect the rooms.

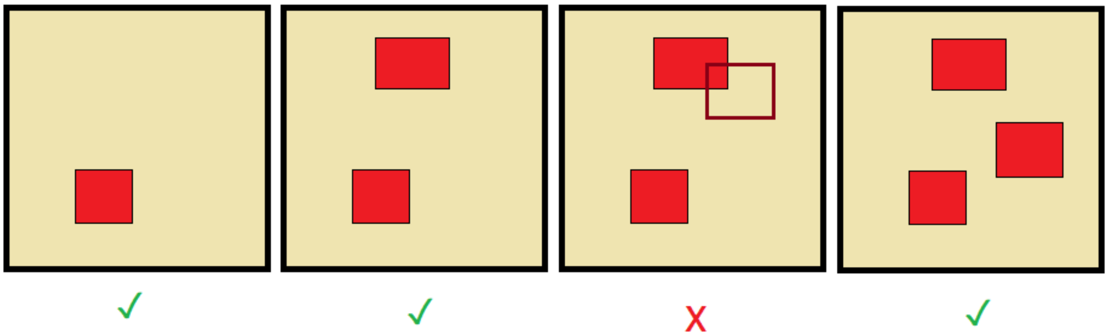

Fig 3.1: RRP places rooms at random on the map. If new room intersects with an existing room, a new location is chosen

3.1.1.2 Binary Space Partitioning Room Placement (BSP Rooms)

As shown in Fig 3.2, Binary Space Partitioning (BSP) Room Placement works by dividing the map into a series of rectangles, and then placing a room in each one on the map. The rectangle can be split either vertically or horizontally, and the minimum size of the split is determined based on the minimum size of the rooms.

Fig 3.2: Visual representation of how BSP Rooms splits map and places rooms

The initial rectangle is called root, and it and all sub rectangles are stored in a tree structure, shown in Fig 3.3. Each rectangle can only be split once, and the two rectangles created by the split are stored as children, when the algorithm is finished dividing the rectangles, it places a room in every non-split rectangle.

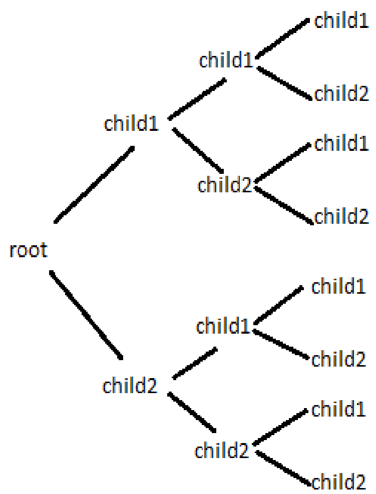

Fig 3.3: BSP Rooms stored rectangles in a tree structure, each sub-leaf can only be divided once

3.1.2 Corridor generation

The algorithms for generating corridors are responsible for drawing the lines on the map to connect the rooms that were generated by the previous sections algorithms. The corridors are a single tile wide and need to connect to every room on the map to create a useable layout.

3.1.2.1 Random Point Connect (RPC)

Random Point Connect (RPC) is a brute force algorithm that works by first iterating through the list of rooms created by the room generating algorithm. It connects each item in the list to the one next to it, so the first room, at index 0, is connected to the room at index 1 and the last room (index is the number_of_rooms - 1) is connected the first room.

For each corridor it draws, it selects a random point at the edge of each of the rooms and connects. If the points are parallel, the algorithm will draw a straight corridor in the appropriate vertical or horizontal direction. If the points are not parallel, the algorithm will draw a forked corridor made up of three straight corridors in order to connect the points.

It draws the same number of corridors as rooms on the map, and the algorithm can intersect other corridors or rooms that are already on the map.

3.1.2.2 Drunkard's Walk (DW)

Drunkard's Walk (DW) is an application of Cellular Automata that, similarly to RPC, also selects vertex between the rooms and starts drawing a line at the first vertex incrementing a randomly chosen amount of vertices in the direction of the second.

The when the corridor is being made by DW, the algorithm do it in "steps". Each step adds a corridor of a random length that goes in the direction of the end point. The algorithm will continue to take steps until it either runs out of maximum allowed steps, or reaches the end point.

3.1.2.3 Binary Space Partitioning Corridors (BSP Corridors)

BSP Corridors works using the tree structure used by BSP Rooms placing algorithm. It sorts through the sub leaves, connecting each bottom leaf to its partner that shares a parent. When all sub-leafs are paired it connects one out of each pair to another pair, until all sub-leafs containing rooms are connected. So in Fig 3.3, it would connect each child1 to its sibling child2, and to the child1 of the pair next to it.

3.2 Experiment Design

3.2.1 Parameters for generating maps

For each algorithm or combination of algorithms, several maps will be generated. This section will discuss what parameters need to be considered when generating the maps, such as the size of the maps generated,the number of rooms allowed on the map and the allowed size of those rooms.

Many of these parameters are based on the experiment by Hilliard et al, which compared the algorithms Span and Growth for execution time, as discussed in the literature review.

3.2.1.1 Tolerance value

When an algorithm attempts to place a room or corridor, it must check that it meets the requirements for placing, for example when placing a room if the planned location touches or overlaps with a different room it cannot be placed. The DW corridor also requires a maximum umber of attempts at drawing a corridor before the algorithm ends.

For the purposes of this experiment a tolerance value of 20 was chosen, it represents the maximum number of times the algorithm will attempt to place a room or corridor. For the purposes of this experiment a tolerance value of 20 was chosen.

3.2.1.2 Map parameters

Maps will be generated in sizes ranging from 100x100 to 1000x1000, increasing in increments of 100 to test the impact of different map sizes on the algorithm. This means that for each algorithm and room size, 10 maps will be generated, one at each map size, and the execution time and resource use will be measured for each one.

3.2.1.3 Room parameters

The amount of rooms required on each map will be determined using the following equation

$$R = (M/S) * 2$$

R is the required number of rooms, M is the the size of the map and S is the maximum size for a room. For measuring space efficiency, the algorithm will not limit the number of rooms generated and instead default to creating the maximum number of rooms allowed by the tolerance value.

The default room size will be between 10 tiles. This will be used for comparing the execution time across different map sizes, so that the room size is standard between the maps produced. Next the experiment will vary the room size from 4px to 20px, in increments of 4px. This will be used to test if execution time or resource use changes significantly at different room sizes.

3.2.2 Efficiency Measurements

3.2.2.1 Time efficiency (execution time)

In games that use PCG maps, every time a person plays the game a new level needs to be generated, this makes time an important factor when choosing which algorithm to use in a game. If the algorithm takes too long to create a map, it could turn people away from playing the game regularly.

First the experiment will measure the average execution time of each algorithm or combination of algorithms. It will then vary the size of the map generated and measure the execution time again, which will be compared against the initial execution time. If an algorithm takes significantly longer to generate a map using a larger map size, then it would mean that map size negatively impacts the time efficiency of the algorithm.

Next the experiment will vary the allowed room sizes and measurer the execution time again, and compare it against the initial execution time. If an algorithm takes significantly longer to generate a map using a larger rooms or using a larger variance in room size, then it would mean that room size negatively impacts the time efficiency of the algorithm.

3.2.2.2 Algorithmic efficiency (resource use / Big O)

Algorithmic efficiency is the measure of computational resources used by the algorithm. It analyses the resources used by an algorithm to determine how productively they are used. The most commonly used notation to describe resource consumption is called *Big O notation*, developed by Donald Knuth.

Big O notation represents the complexity of an algorithm as a function of the size of the input, *f(n)*. It measures the efficiency as *f (n) = O (g(n))*, where *f (n)* and *g (n)* are functions defined for positive integers.

The RRP room placement algorithm works by generating a room based on dimensions provided to it and then attempting to place the room at a random point on the map. The main resources identified from this process that will be used in the big O measure is the room dimensions, resulting in the measure *O (max_room_width * max_room_height)*.

The BSP room placement algorithm works by splitting the map into sections (children) until each one is a minimum size, a room is then placed into each section. The resources identified from this process are the number of splits performed, the total number of children created and the dimensions of the rooms. This resulted in the measure *O (number_of_splits + number_of_children + (max_room_width * max_room_height))*.

The RPC corridor placing algorithm works by selecting a random point on the border of two rooms and drawing a line between them. The resources identified in this process are the number of times the process is repeated in order to connect all the rooms, and the width and height of the map, as this controls the maximum lengths of the corridor. This resulted in the measure *O (number_of_iterations * (map_width + map_height))*.

The BSP corridor placing algorithm works by first pairing the bottom leaf nodes who share a parent to each other, it then pairs one of each pair to a different pair of siblings. The resources identified in this algorithm are the number of calculations (the work needed to select a pair of rooms and draw a corridor between them), and the dimensions of the map. This resulted in the measure *O(3 calculations * (map_width + map_height))* where calculations is multiplied by 3 for the number of calculations per child, per corridor part and per pairing.

The Drunkard's Walk corridor placing algorithm works by selecting a random point on the border of a room and a second point a certain distance away from the room. The algorithm continues to draw the corridor in a random direction until it either reaches a room, goes out of bounds or goes beyond a maximum allowed length. The resources identified in this algorithm are the number of times the main function places a new corridor for a room, the number of times the placing function draws a corridor before it either reaches a room or gives up, and the dimensions of the map. This resulted in the measure *O (number_of_main_loop_iterations * number_of_while_loop_iterations (map_width + map_height))*.

3.3 Application Design

The maps generated are the type used in 2D top down games, and will be created in the Unity game engine using C#.

The class diagram in Fig 3.4 shows how the application was initially designed. The class *MenuControl* ontains the functionality for the user to select the algorithm, map size and room size for the map they want to make. When the user presses the "Create Map" button, the class *BoardCreator* will be called and passed the details of the map to create.

BoardCreator first calls the desired map algorithm class and starts tracking the execution time. The map class creates a new *CellMap* and adds the desired number of rooms to it. The *CellMap* and efficiency score are then returned to the *BoardCreator*, which ends its first timer.

Then the corridor algorithm is called and a second timer is started. The algorithm is given the CellMap, which it adds corridors to and returns along with its efficiency score. Finally the map is rendered and the total execution time and efficiency is displayed to the user.

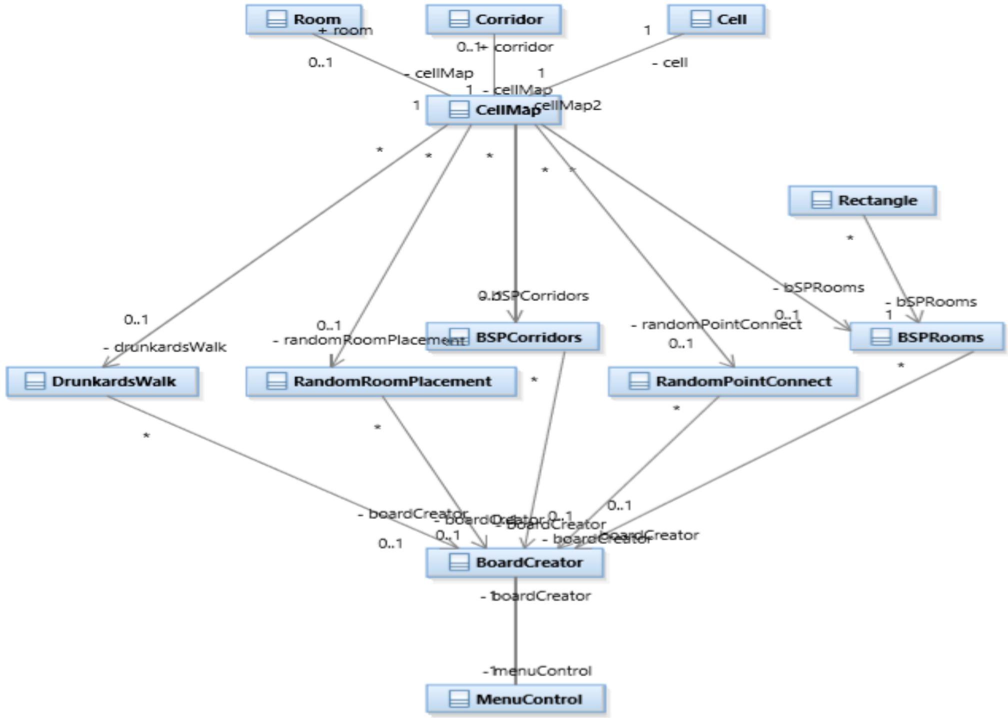

Fig 3.4: Proposed Class Diagram for map making application

3.3.1 Application requirements

Application should be able to implement a variety of PCG algorithms for creating a 2D map. User should be able to apply each algorithm, or combination of algorithms to the grid and have the application produce a number of sample dungeons.

The samples will be generated using different parameters, including different grid sizes and different values for the minimum and maximum amount of "on" tiles allowed on the grid.

During generation the time taken will be tracked along with the CPU usage for the machine. The dungeons generated will be sorted according to algorithms and parameters and then compared.

4. IMPLEMENTATION AND RESULTS

This section will review the results of implementing the combination of algorithms given the parameters reviewed in Chapter 3. Each combination of algorithm will be run 5 times for each part of the experiment, and the average of all runs will be used for analysis. All maps and rooms will be measured in (px), map sizes will range from 100px to 1000px, increasing in increments of 100px.

Execution time is measured in milliseconds (ms), a line chart will be used to display the results of the measurements, this was chosen because it displays the change in execution time over a series of map sizes, allowing trends to be more easily seen. A trend line representing the behaviour of the data will also be included on some charts.

Resource use, which is measured using Big O notation, will be displayed on a smoothed line chart. This chart is similar to a normal line chart, except the line used is smoothed, it was selected because measurements are taken from the equations discussed in Chapter 3, and are expected to follow a curve.

First, the measurements for the lowest map size of 100px and highest map size of 1,000px will be compared to will identify if the difference is positive or negative, and the percentage of the increase/decrease. Next the difference in increase or decrease between the two highest set of points (900px - 1,000px) and the two lowest set of points (100px - 200px) will be compared to see if the data behaves differently at the different size ranges.

Finally the series of map sizes will be split in half (from 100px - 500px and 600px - 1,000px) and the increase/decrease in the two ranges will be compared to see if the behaviour identified by the previous comparison is repeated across the larger range.

4.1 Using fixed room size

In this section, each combination of algorithm was run using a room size of 10px, measuring execution time and resource use. For each combination, the results will be displayed and key points on the chart will be identified.

4.1.1 BSP Rooms and BSP Corridors

4.1.1.1 Execution Time

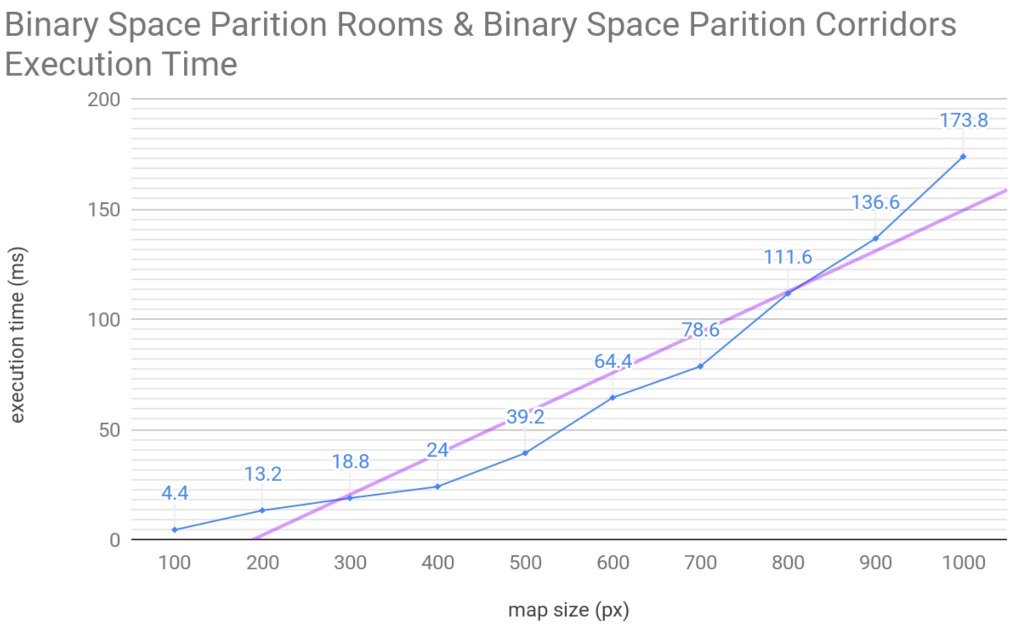

Fig 4.1: Line Chart showing execution time of Binary Space Partition Rooms and Binary Space Partition Corridors when room size is 10px at different map sizes (blue) and trend line (purple).

In Fig 4.1, execution time is shown to increase as map size increases, this is also reflected in the positive direction of the trend line. Table 4.1 shows that at all sample map ranges, the execution time increases at a faster rate then the map size, indicating that map size negatively affects execution time efficiency.

map size range	map size px increase	map size % increase	execution time ms increase	execution time % increase
100px - 1,000px	900px	900%	169.4ms	3,850%
100px - 500px	400px	400%	34.8ms	791%
100px - 200px	100px	100%	8.8ms	200%
600px - 1,000px	400px	67%	109.4ms	170%
900px - 1,000px	100px	11%	37.2ms	27%

Table 4.1: Execution time increase of Binary Space Partition Rooms and Binary Space Partition Corridors when room size is 10px at different map size ranges.

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time goes from 4.4ms to 39.2ms (791% increase). When increasing map size from 600px to 1,000px (67% increase), the execution time goes from 64.4ms to 173.8ms (170% increase).

When comparing an increase of 100px at the highest and lowest end of the graph, when the map size is increased from 100px to 200px (100% increase), the execution time goes from 4.4ms to 13.2ms (200% increase). When increasing map size from 900px to 1,000px (11% increase), the execution time goes from 136.6ms to 173.8ms (27% increase).

This indicates that the percentage increase of the execution time gets lower at higher map sizes, however the execution time still increases at a faster rate then the map size.

4.1.1.2 Resource use (Big O)

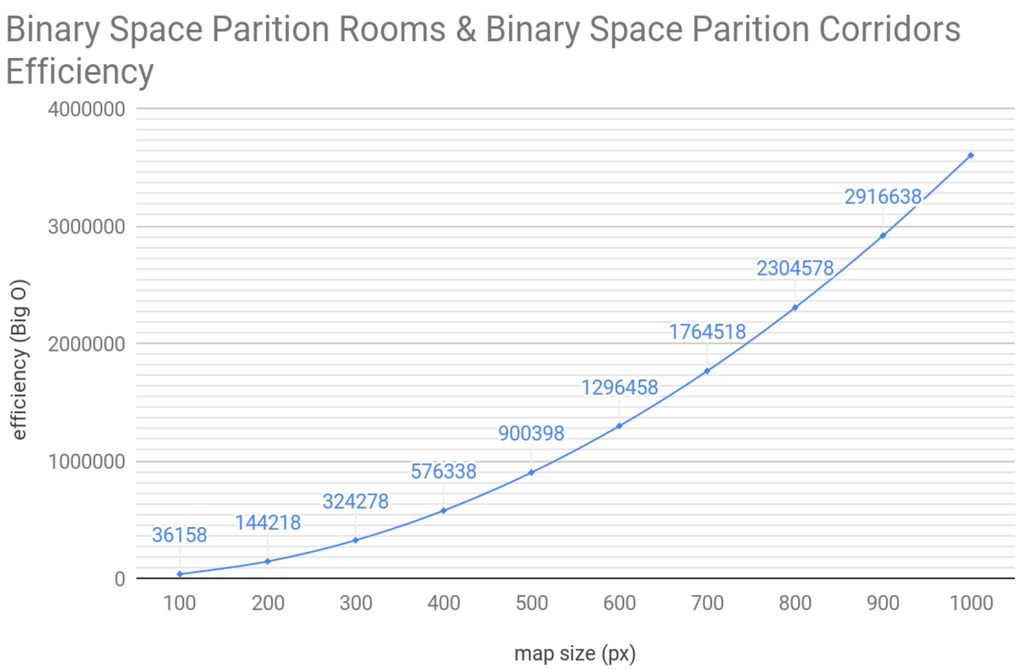

Fig 4.2: Smooth Line Chart showing resource use of Binary Space Partition Rooms and Binary Space Partition Corridors when room size is 10px at different map sizes.

In Fig 4.2, resource use is shown to increase as map size increases. Table 4.2 shows that at all sample map ranges, the resource use increases at a faster rate then the map size, indicating that map size negatively affects resource efficiency.

map size range	map size px increase	map size % increase	resource use resource increase	resource use % increase
100px - 1,000px	900px	900%	3,564,540	9,858%
100px - 500px	400px	400%	864,240	2,390%
100px - 200px	100px	100%	108,060	299%
600px - 1,000px	400px	67%	2,304,240	178%
900px - 1,000px	100px	11%	684,060	23%

Table 4.2: Resource use increase of Binary Space Partition Rooms and Binary Space Partition Corridors when room size is 10px at different map size ranges.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use goes from 36,158 to 900,398 (2,390% increase). When increasing map size from 600px to 1,000px (67% increase), the resource use goes from 1,296,458 to 3,600,698 (178% increase).

When comparing an increase of 100px at the highest and lowest end of the graph, when the map size is increased from 100px to 200px (100% increase), the resource use goes from 36,158 to 144,218 (299% increase). When increasing map size from 900px to 1,000px (11% increase), the resource use goes from 2,916,638 to 3,600,698 (23% increase).

This indicates that the percentage increase of the resource use gets lower at higher map sizes, however the resource use still increases at a faster rate then the map size.

4.1.2 BSP Rooms and RPC

4.1.2.1 Execution Time

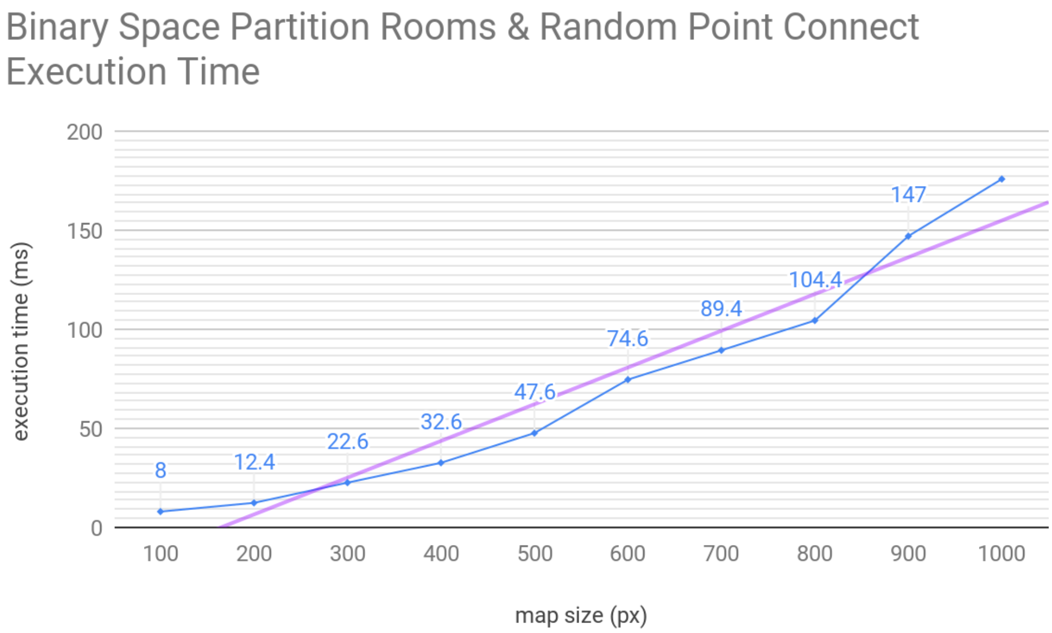

Fig 4.3: Line Chart showing execution time of Binary Space Partition Rooms and Random Point Connect when room size is 10px at different map sizes (blue) and trend line (purple).

In Fig 4.3, execution time is shown to increase as map size increases, this is also reflected in the positive direction of the trend line. Table 4.3 shows that at the lowest map size range, 100px to 200px, the map size increases at a faster rate then the execution time.

However at all other sample map size ranges, execution time increases at a faster rate then map size. This indicates that at lower map ranges, execution time increases at a slower rate then map size, but at higher map sizes it increases at a faster rate.

map size range	map size px increase	map size % increase	execution time ms increase	execution time % increase
100px - 1,000px	900px	900%	167.8ms	2,098%
100px - 500px	400px	400%	39.6ms	495%
100px - 200px	100px	100%	4.4ms	55%
600px - 1,000px	400px	67%	101.2ms	136%
900px - 1,000px	100px	11%	28.8ms	20%

Table 4.3: Execution time increase of Binary Space Partition Rooms and Random Point Connect when room size is 10px at different map size ranges.

When comparing the total increase in map size to the increase in execution time, when the map size is increased from 100px to 1,000px (900% increase), the execution time goes from 8ms to 175.8ms (2,098% increase).

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time goes from 8ms to 47.6ms (495% increase). When increasing map size from 600px to 1,000px (67% increase), the execution time goes from 74.6ms to 175.8ms (136% increase).

This indicates that the execution time efficiency is still negatively impacted by map size, however the rate of increase in execution time is at its fastest in the middle section of the graph, and is slower at the highest and lowest sections.

4.1.2.2 Resource use (Big O)

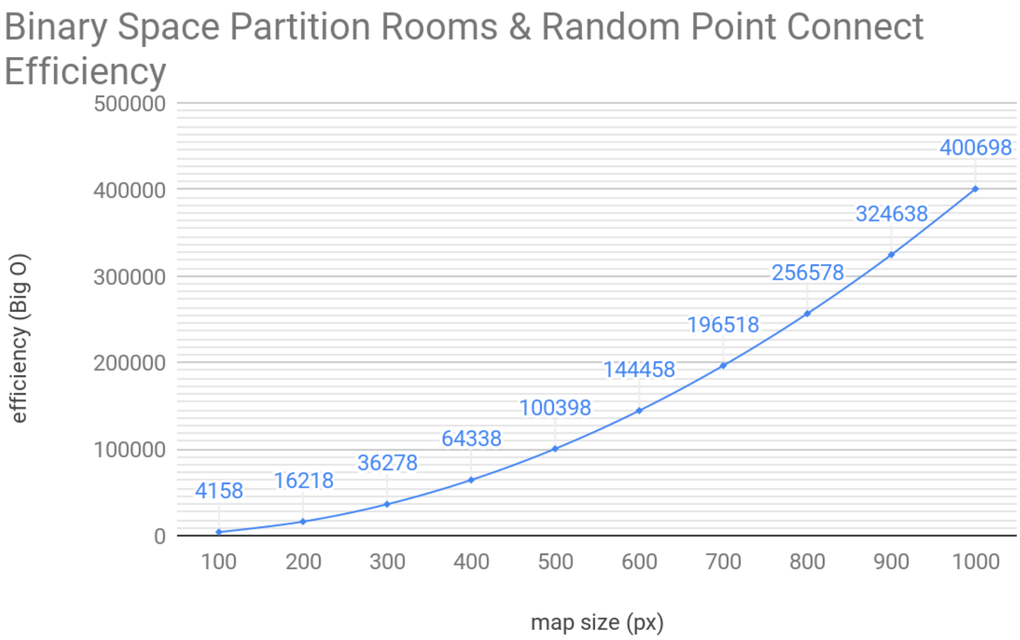

Fig 4.4: Smooth Line Chart showing resource use of Binary Space Partition Rooms and Random Point Connect when room size is 10px at different map sizes.

In Fig 4.4, resource use is shown to increase as map size increases. Table 4.4 shows that at all sample map ranges, the resource use increases at a faster rate then the map size, indicating that map size negatively affects resource efficiency.

map size range	map size px increase	map size % increase	resource use increase	resource use % increase
100px - 1,000px	900px	900%	396,540	9,537%
100px - 500px	400px	400%	96,240	2,315%
100px - 200px	100px	100%	12,060	290%
600px - 1,000px	400px	67%	25,6240	177%
900px - 1,000px	100px	11%	76,060	23%

Table 4.4: Resource use increase of Binary Space Partition Rooms and Random Point Connect when room size is 10px at different map size ranges.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use goes from 4,158 to 100,398 (2,315% increase). When increasing map size from 600px to 1,000px (67% increase), the resource use goes from 144,458 to 400,698 (177% increase).

When comparing an increase of 100px at the highest and lowest end of the graph, when the map size is increased from 100px to 200px (100% increase), the resource use goes from 4,158 to 16,218 (290% increase). When increasing map size from 900px to 1,000px (11% increase), the resource use goes from 324,638 to 400,698 (23% increase).

This indicates that the percentage increase of the resource use gets lower at higher map sizes, however the resource use still increases at a faster rate then the map size.

4.1.3 BSP Rooms and DW

4.1.3.1 Execution Time

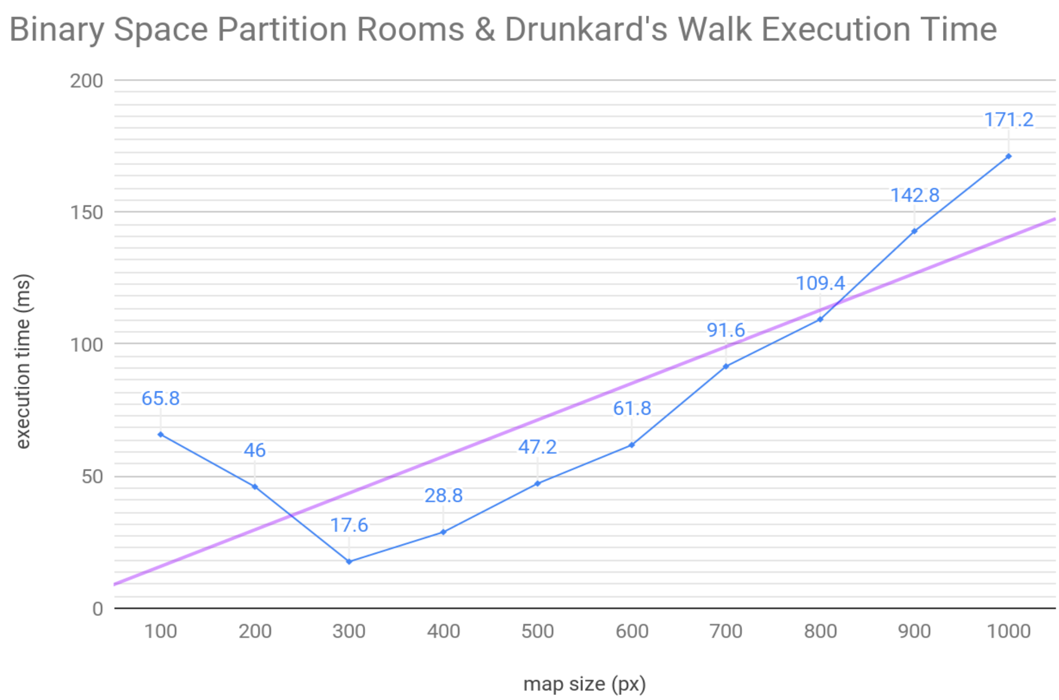

Fig 4.5: Line Chart showing execution time of Binary Space Partition Rooms and Drunkard's Walk when room size is 10px at different map sizes (blue) and trend line (purple).

In Fig 4.5, execution time starts off by decreasing as map size increases, however after map size is increased past 300px, execution time starts to increase. The positive direction of the trend line indicates that overall, the execution time increases as map size increases.

Table 4.5 shows that at the lowest map size ranges of 100px to 200px, and 100px to 500px, the execution rate decreases as map size increases. It also shows that at the higher map size ranges of 600px to 1000px and 900px to 1000px, the execution time increases as map size increases.

map size range	map size px increase	map size % increase	execution time ms increase / decrease	execution time % increase / decrease
100px - 1,000px	900px	900%	105.4ms	160%
100px - 500px	400px	400%	-18.6ms	-28%
100px - 200px	100px	100%	-19.8ms	-30%
600px - 1,000px	400px	67%	109.4ms	177%
900px - 1,000px	100px	11%	28.4ms	20%

Table 4.5: Execution time increase / decrease (shown as -) of Binary Space Partition Rooms and Drunkard's Walk when room size is 10px at different map size ranges.

When comparing the total increase in map size to the increase in execution time, when the map size is increased from 100px to 1,000px (900% increase), the execution time goes from 65.8ms to 171.2ms (160% increase).

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time goes from 65.8ms to 47.2ms (28% increase). When increasing map size from 600px to 1,000px (67% increase), the execution time goes from 61.8ms to 171.2ms (177% increase).

This indicates that the execution time is negatively affected by map size and that the rate of increase of the execution time increases at higher map sizes.

4.1.3.2 Resource use (Big O)

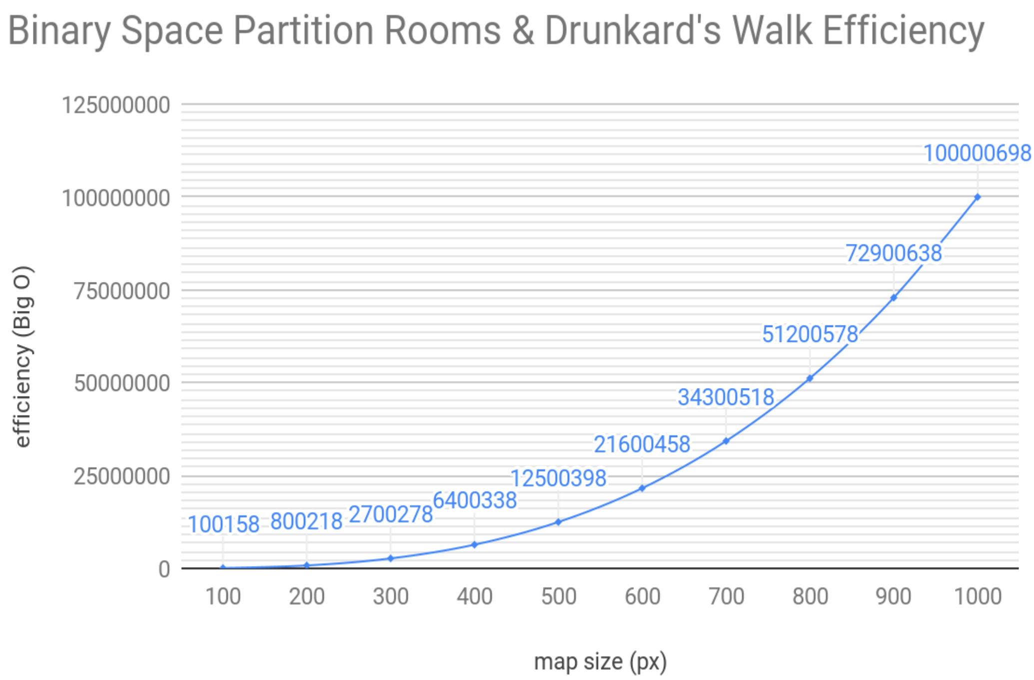

Fig 4.6: Smooth Line Chart showing resource use of Binary Space Partition Rooms and Drunkard's Walk when room size is 10px at different map sizes.

In Fig 4.6, resource use is shown to increase as map size increases. Table 4.6 shows that at all sample map ranges, the resource use increases at a faster rate then the map size, indicating that map size negatively affects resource efficiency.

map size range	map size px increase	map size % increase	resource use increase	resource use % increase
100px - 1,000px	900px	900%	99,900,540	99,743%
100px - 500px	400px	400%	12,400,240	12,381%
100px - 200px	100px	100%	700,060	699%
600px - 1,000px	400px	67%	78,400,240	363%
900px - 1,000px	100px	11%	27,100,060	37%

Table 4.6: Resource use increase of Binary Space Partition Rooms and Drunkard's Walk when room size is 10px at different map size ranges.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use goes from 100,158 to 12,500,398 (12,381% increase). When increasing map size from 600px to 1,000px (67% increase), the resource use goes from 21,600,458 to 100,000,698 (363% increase).

When comparing an increase of 100px at the highest and lowest end of the graph, when the map size is increased from 100px to 200px (100% increase), the resource use goes from 100,158 to 800,218 (699% increase). When increasing map size from 900px to 1,000px (11% increase), the resource use goes from 72,900,638 to 100,000,698 (37% increase).

This indicates that the percentage increase of the resource use gets lower at higher map sizes, however the resource use still increases at a faster rate then the map size.

4.1.4 RRP and RPC

4.1.4.1 Execution Time

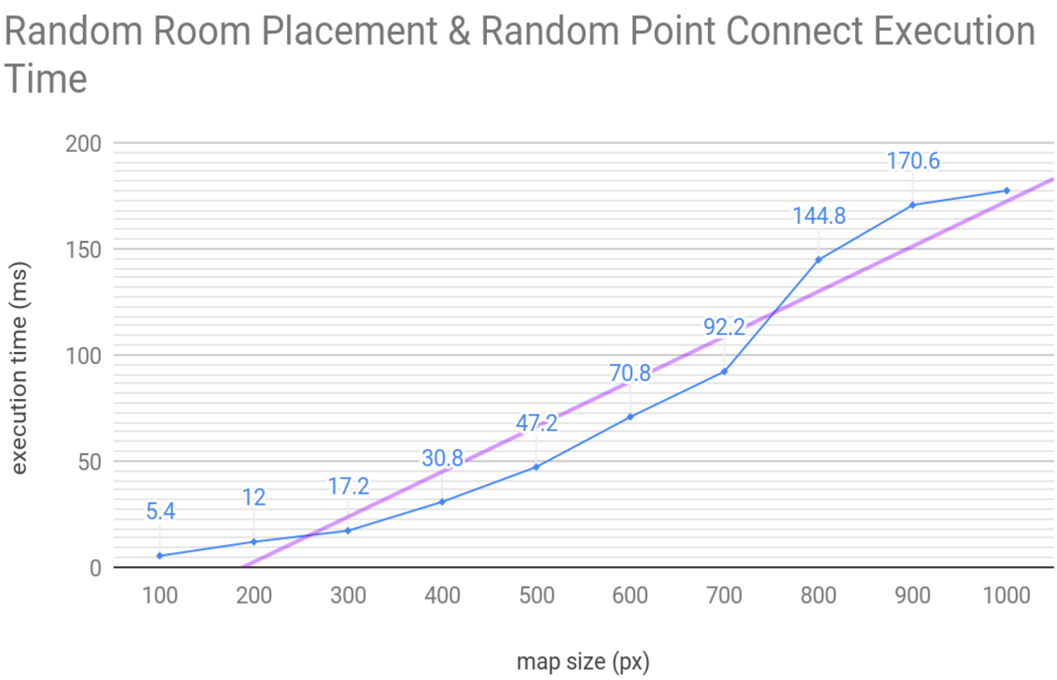

Fig 4.7: Line Chart showing execution time of Random Room Placement and Random Point Connect when room size is 10px at different map sizes (blue) and trend line (purple).

In Fig 4.7, execution time is shown to increase as map size increases, this is also reflected in the positive direction of the trend line. Table 4.7 shows that at lower map size ranges the execution time increases at a faster rate then higher map sizes.

map size range	map size px increase	map size % increase	execution time ms increase	execution time % increase
100px - 1,000px	900px	900%	172	3,185%
100px - 500px	400px	400%	41.8	774%
100px - 200px	100px	100%	6.6	122%
600px - 1,000px	400px	67%	106.6	151%
900px - 1,000px	100px	11%	6.8	4%

Table 4.7: Execution time increase of Random Room Placement and Random Point Connect when room size is 10px at different map size ranges.

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time goes from 5.4ms to 47.2ms (774% increase). When increasing map size from 600px to 1,000px (67% increase), the execution time goes from 70.8ms to 177.4ms (151% increase).

When comparing an increase of 100px at the highest and lowest end of the graph, when the map size is increased from 100px to 200px (100% increase), the execution time goes from 5.4ms to 12ms (122% increase). When increasing map size from 900px to 1,000px (11% increase), the execution time goes from 170.6ms to 177.4ms (4% increase).

This indicates that at the rate of increase of the execution time gets slower as map size increases and that at higher map ranges, the map size increases at a faster rate then the execution time.

4.1.4.2 Resource use (Big O)

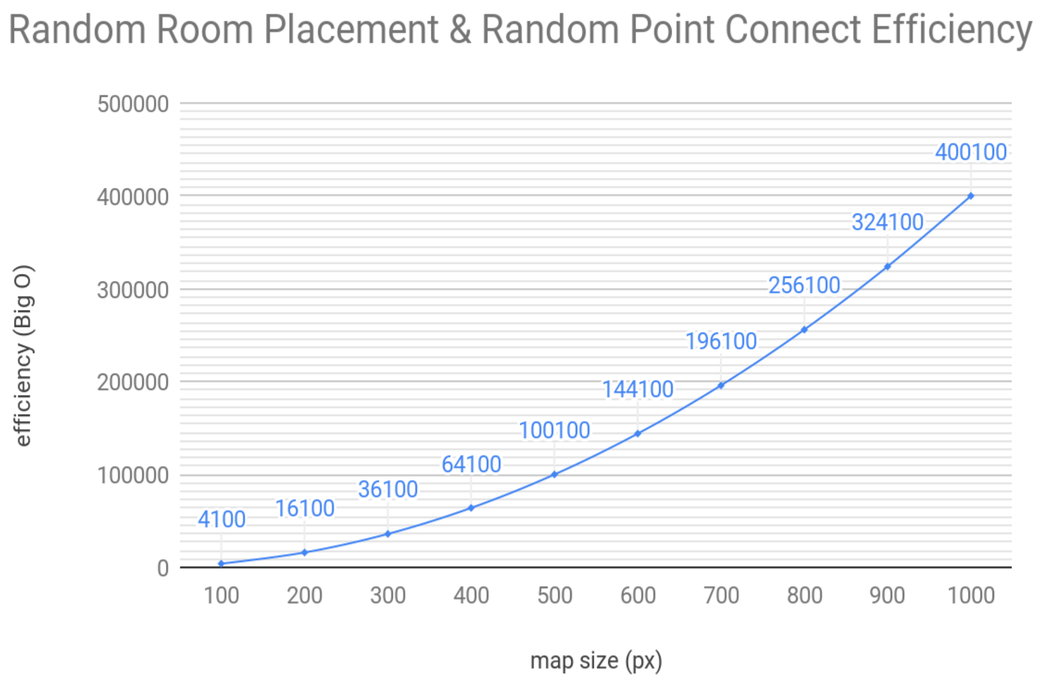

Fig 4.8: Smooth Line Chart showing resource use of Random Room Placement Rooms and Random Point Connect when room size is 10px at different map sizes.

In Fig 4.8 resource use is shown to increase as map size increases. Table 4.8 shows that at all sample map ranges, the resource use increases at a faster rate then the map size, indicating that map size negatively affects resource efficiency.

map size range	map size px increase	map size % increase	resource use increase	resource use % increase
100px - 1,000px	900px	900%	396,000	9,659%
100px - 500px	400px	400%	96,000	2,341%
100px - 200px	100px	100%	12,000	293%
600px - 1,000px	400px	67%	256,000	178%
900px - 1,000px	100px	11%	76,000	23%

Table 4.8: Resource use increase of Random Room Placement Rooms and Random Point Connect when room size is 10px at different map size ranges.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use goes from 4,100 to 100,100 (2,341% increase). When increasing map size from 600px to 1,000px (67% increase), the resource use goes from 144,100 to 400,100 (178% increase).

When comparing an increase of 100px at the highest and lowest end of the graph, when the map size is increased from 100px to 200px (100% increase), the resource use goes from 4,100 to 16,100 (293% increase). When increasing map size from 900px to 1,000px (11% increase), the resource use goes from 324,100 to 400,100 (23% increase).

This indicates that the percentage increase of the resource use gets lower at higher map sizes, however the resource use still increases at a faster rate then the map size.

4.1.5 RRP and DW

4.1.5.1 Execution Time

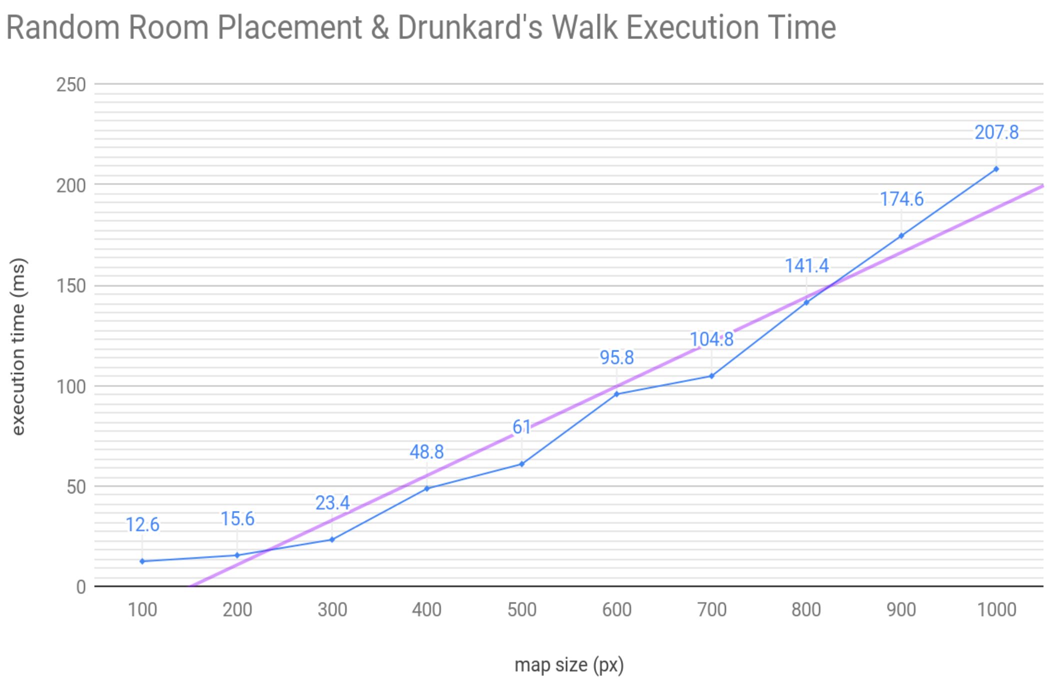

Fig 4.9: Line Chart showing execution time of Random Room Placement and Drunkard's Walk when room size is 10px at different map sizes (blue) and trend line (purple).

In Fig 4.9, execution time is shown to increase as map size increases, this is also reflected in the positive direction of the trend line. Table 4.9 shows that at lower map size ranges, the rate of increase of the execution time is lower when compared to the rate of increase of the map size. It also shows that at higher map size ranges, the rate of increase in the execution time becomes faster then the rate of increase of the map size.

map size range	**map size px increase**	**map size % increase**	**execution time ms increase**	**execution time % increase**
100px - 1,000px	900px	900%	195.2	1,549%
100px - 500px	400px	400%	48.4	384%
100px - 200px	100px	100%	3	24%
600px - 1,000px	400px	67%	112	117%
900px - 1,000px	100px	11%	33.2	19%

Table 4.9: Execution time increase of Random Room Placement and Drunkard's Walk when room size is 10px at different map size ranges.

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time goes from 12.6ms to 61ms (384% increase). When increasing map size from

600px to 1,000px (67% increase), the execution time goes from 95.8ms to 207.8ms (117% increase).

When comparing an increase of 100px at the highest and lowest end of the graph, when the map size is increased from 100px to 200px (100% increase), the execution time goes from 12.6ms to 15.6ms (24% increase). When increasing map size from 900px to 1,000px (11% increase), the execution time goes from 1174.6ms to 207.8ms (19% increase).

This indicates that at the rate of increase of the execution time gets higher as map size increases and that at higher map ranges, the execution time increases at a faster rate then the map size

4.1.5.2 Resource use (Big O)

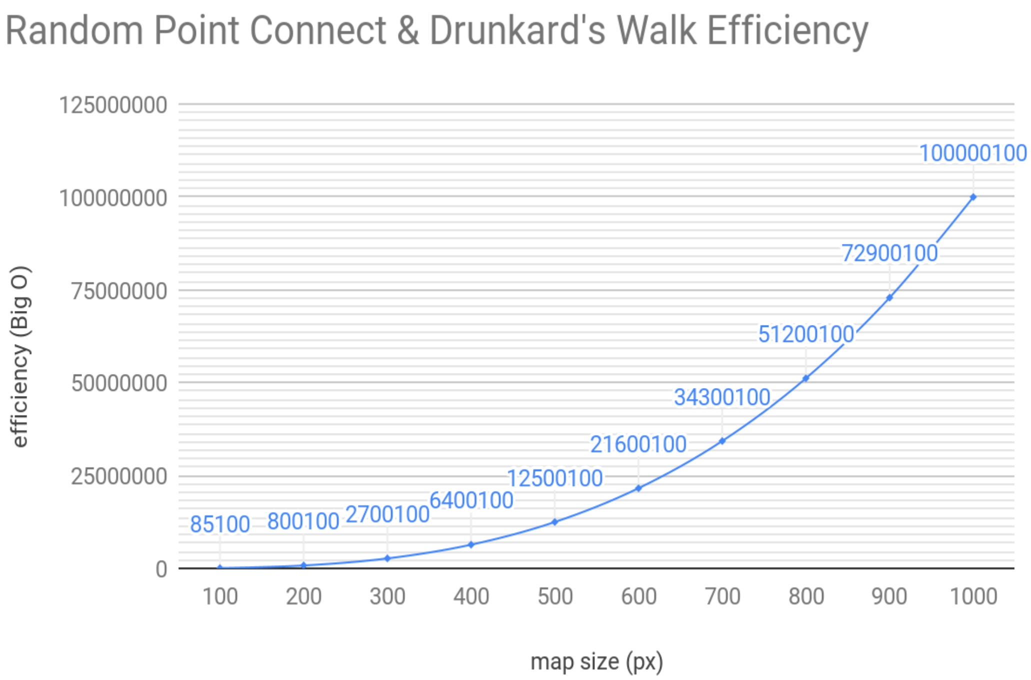

Fig 4.10: Smooth Line Chart showing resource use of Random Room Placement Rooms and Drunkard's Walk when room size is 10px at different map sizes.

In Fig 4.10 resource use is shown to increase as map size increases. Table 4.10 shows that at all sample map ranges, the resource use increases at a faster rate then the map size, indicating that map size negatively affects resource efficiency.

map size range	map size px increase	map size % increase	resource use increase	resource use % increase
100px - 1,000px	900px	900%	99,915,000	117,409%
100px - 500px	400px	400%	12,415,000	14,589%
100px - 200px	100px	100%	715,000	840%
600px - 1,000px	400px	67%	78,400,000	363%
900px - 1,000px	100px	11%	27,100,000	37%

Table 4.10: Resource use increase of Random Room Placement Rooms and Drunkard's Walk when room size is 10px at different map size ranges.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use goes from 85,100 to 12,500,100 (14,589% increase). When increasing map size from 600px to 1,000px (67% increase), the resource use goes from 21,600,100 to 100,000,100 (363% increase).

When comparing an increase of 100px at the highest and lowest end of the graph, when the map size is increased from 100px to 200px (100% increase), the resource use goes from 85,100 to 800,100 (840% increase). When increasing map size from 900px to 1,000px (11% increase), the resource use goes from 72,900,100 to 100,000,100 (37% increase).

This indicates that the percentage increase of the resource use gets lower at higher map sizes, however the resource use still increases at a faster rate then the map size.

4.2 Using varied room sizes

For this portion of the test, the execution time was measured for different map sizes using a range of different room sizes. The ranges chosen range from 4px to 20px, increasing in increments of 4px, with the same maximum and minimum room sizes being used for each measurement.

Each graph will show the execution rate or resource use of an algorithm combination for each room size in order to best show the differences in results.

4.2.1 BSP Rooms and BSP Corridors

4.2.1.1 Execution Time

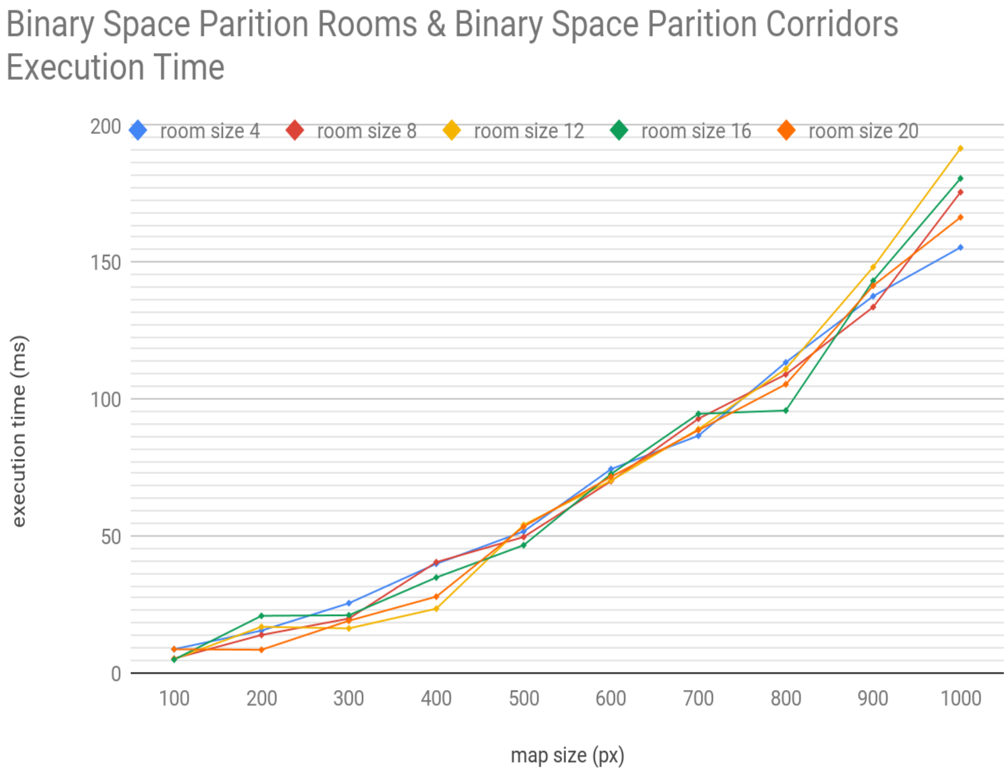

Fig 4.11: Line Chart showing execution time of Binary Space Partition Rooms and Binary Space Partition Corridors at different room sizes and map sizes.

As can be seen in Fig 4.11, the execution time increases as map size increases across all room sizes.

When comparing the values at the lowest and highest map sizes used, at map size of 100px, room size of 12px has the lowest execution time (5ms) and room sizes of 4px and 20px have the highest (8.6ms). At map size of 1,000px, room size of 4px has the lowest execution time (155.4ms) and room size of 12px has the highest (191.6ms). This shows that none of the room sizes have consistently higher execution time then the others across the map sizes.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	1,707%	3,277%	3,732%	3,663%	1,835%
100px - 500px	400%	500%	854%	980%	871%	521%
100px - 200px	100%	79%	165%	236%	333%	-2%
600px - 1,000px	67%	109%	151%	173%	149%	132%
900px - 1,000px	11%	13%	31%	29%	26%	18%

Table 4.11: Execution time increase of Binary Space Partition Rooms and Binary Space Partition Corridors at different map size ranges for each room size.

Table 4.11 shows that at most map size ranges, the execution time increases at a faster rate than the map size. The exception is that at the lowest map size range of 100px to 200px, at room size 4px the execution time increases at a slower rate then the map size and at room size 20px, the execution time decreases as map size increases.

At all map size ranges, room sizes 4px and 20px have the lowest increases in execution time. This means that the rate of increase of the execution time is higher at room sizes 8px - 16px.

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time increases as follows at different room sizes; 4px : 500%, 8px : 854%, 12px : 980%, 16px : 871%, 20px : 521%. When increasing map size from 600px to 1,000px (67% increase), the execution time increases as follows at different room sizes; 4px : 109%, 8px : 151%, 12px : 173%, 16px : 149%, 20px : 132%.

This indicates that at all room sizes, the rate of increase of the execution time is faster at lower map ranges and slower at higher map ranges.

4.2.1.2 Resource use (Big O)

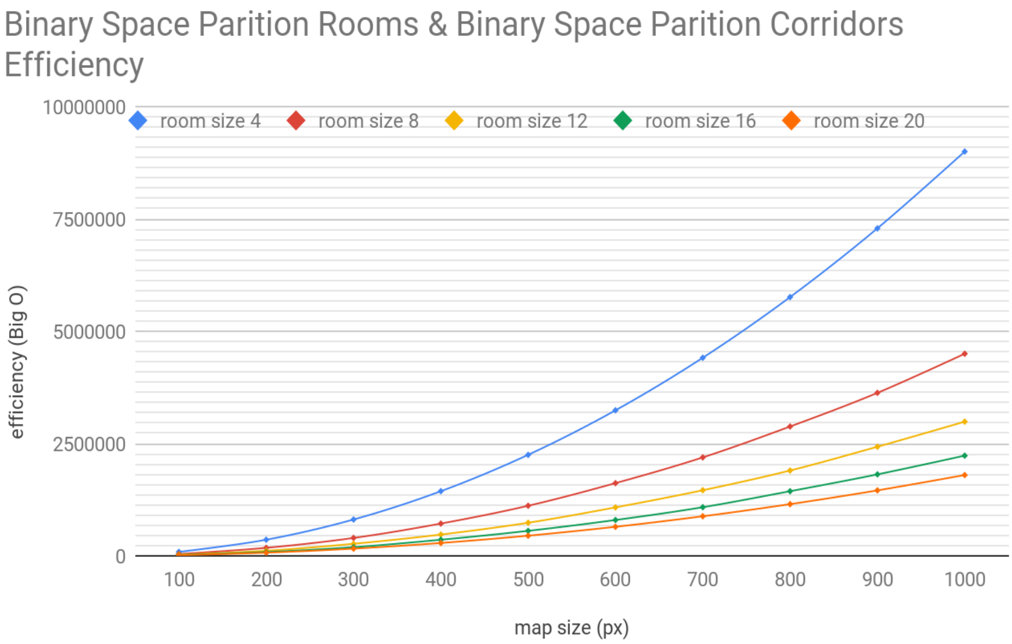

Fig 4.12: Smooth Line Chart showing resource use of Binary Space Partition Rooms and Binary Space Partition Corridors at different room sizes and map sizes.

As shown in Fig 4.12, at all map sizes, higher room sizes use less resources, this indicates that resource use increases as room size decreases.

This trend can be seen when comparing resource use at each end of the chart. At map size of 100px, the resource use for each room size is as follows; 4px : 90,164, 8px : 43,334, 12px : 28,990, 16px : 21,890, 20px : 18,428. At map size of 1,000px, the resource use for each room size is as follows; 4px : 9,001,514, 8px : 4,500,812, 12px : 2,988,640, 16px : 2,232,626, 20px : 1,800,698.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	9,883%	10,286%	10,209%	10,099%	9,672%
100px - 500px	400%	2,396%	2,476%	2,447%	2,451%	2,345%
100px - 200px	100%	300%	316%	298%	296%	293%
600px - 1,000px	67%	178%	178%	177%	179%	178%
900px - 1,000px	11%	23%	24%	23%	23%	23%

Table 4.12: Resource use increase of Binary Space Partition Rooms and Binary Space Partition Corridors at different map size ranges for each room size.

Table 4.12 that shows that while the amount of resources used increases as room size decreases, the rate of increase does not vary as much between room sizes.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use increases as follows at different room sizes; 4px : 2,396%, 8px : 2,476%, 12px : 2,447%, 16px : 2,451%, 20px : 2,345%. When increasing map size from 600px to 1,000px (67% increase), the resource use increases as follows at different room sizes; 4px : 178%, 8px : 178%, 12px : 177%, 16px : 179%, 20px : 178%.

This indicates that rate of increase of the resource use gets lower at higher map sizes. The difference in increases gets smaller at higher map sizes, as is also shown when looking at the highest map size range of 900px to 1,000px, where all room size increases are between 23% and 24%.

4.2.2 BSP Rooms and RPC

4.2.2.1 Execution Time

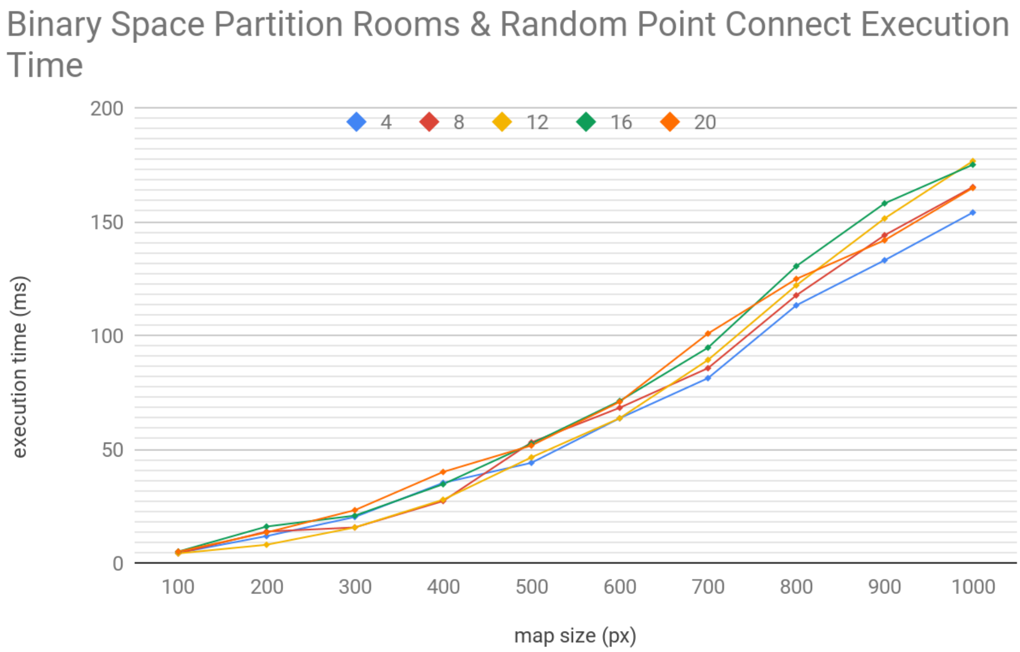

Fig 4.13: Line Chart showing execution time of Binary Space Partition Rooms and Random Point Connect at different room sizes and map sizes.

As can be seen in Fig 4.13, the execution time increases as map size increases across all room sizes. When comparing the values at the lowest and highest map sizes used, at map size of 100px, room size of 8px has the lowest execution time (4.2ms) and room sizes of 16px and 20px have the highest (5ms). At map size of 1,000px, room size of 4px has the lowest execution time (154ms) and room size of 12px has the highest (176ms). This shows that none of the room sizes have consistently higher execution time then the others across the map sizes.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	3,400%	3,833%	4,105%	3,400%	3,196%
100px - 500px	400%	900%	1,162%	1,005%	948%	932%
100px - 200px	100%	168%	229%	90%	220%	168%
600px - 1,000px	67%	142%	142%	178%	146%	133%
900px - 1,000px	11%	16%	15%	17%	11%	16%

Table 4.13: Execution time increase of Binary Space Partition Rooms and Random Point Connect at different map size ranges for each room size.

Table 4.13 shows that at most map size ranges, the execution time increases at a faster rate than the map size. The exception is at the map size range of 900px to 1,000px, at room size 16px, where both execution time and map size increase at the same rate and at the map size range of 100px to 200px, at room size 12px, where execution time increases at a lower rate then map size.

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time increases as follows at different room sizes; 4px : 900%, 8px : 1,162%, 12px : 1,005%, 16px : 948%, 20px : 932%. When increasing map size from 600px to 1,000px (67% increase), the execution time increases as follows at different room sizes; 4px : 142%, 8px : 142%, 12px : 178%, 16px : 146%, 20px : 133%.

The rate of increase of the execution time is highest at room sizes of 8px and 12px for most map size ranges. However, this varies which could indicate that room size does not have a strong impact on the rate of increase of the execution time.

4.2.2.2 Efficiency (resource use)

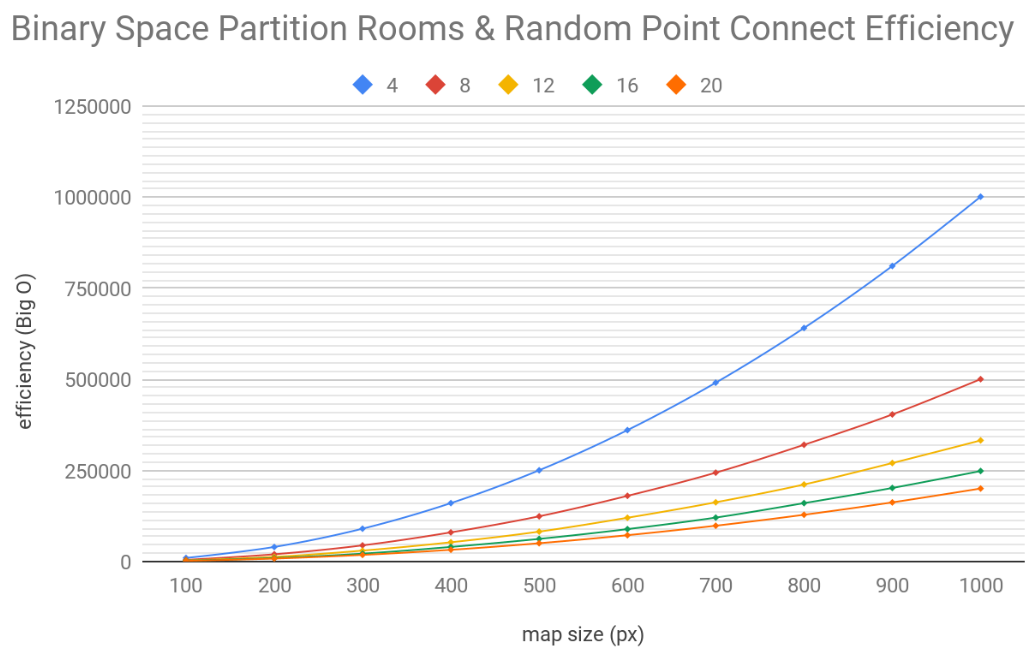

Fig 4.14: Smooth Line Chart showing resource use of Binary Space Partition Rooms and Random Point Connect at different room sizes and map sizes.

As shown in Fig 4.14, at all map sizes, higher room sizes use less resources, this indicates that resource use increases as room size decreases.

This trend can be seen when comparing resource use at each end of the chart. At map size of 100px, the resource use for each room size is as follows; 4px : 10,164, 8px : 4,934, 12px : 3390, 16px : 2,690, 20px : 2,225. At map size of 1,000px, the resource use for each room size is as follows; 4px : 1,001,514, 8px : 500,812, 12px : 332,640, 16px : 248,626, 20px : 200,698.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	9,754%	10,050%	9,712%	9,143%	8,920%
100px - 500px	400%	2,367%	2,422%	2,330%	2,221%	2,172%
100px - 200px	100%	297%	310%	285%	269%	280%
600px - 1,000px	67%	177%	177%	176%	178%	177%
900px - 1,000px	11%	23%	24%	23%	23%	23%

Table 4.14: Resource use increase of Binary Space Partition Rooms and Random Point Connect at different map size ranges for each room size.

Table 4.14 that shows that while the amount of resources used increases as room size decreases, the rate of increase does not vary as much between room sizes.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use increases as follows at different room sizes; 4px : 2,367%, 8px : 2,422%, 12px : 2,330%, 16px : 2,221%, 20px : 2,172%. When increasing map size from 600px to 1,000px (67% increase), the resource use increases as follows at different room sizes; 4px : 177%, 8px : 177%, 12px : 176%, 16px : 178%, 20px : 177%.

This indicates that rate of increase of the resource use gets lower at higher map sizes. The difference in increases gets smaller at higher map sizes, as is also shown when looking at the highest map size range of 900px to 1,000px, where all room size increases are between 23% and 24%.

4.2.3 BSP Rooms and DW

4.2.3.1 Execution Time

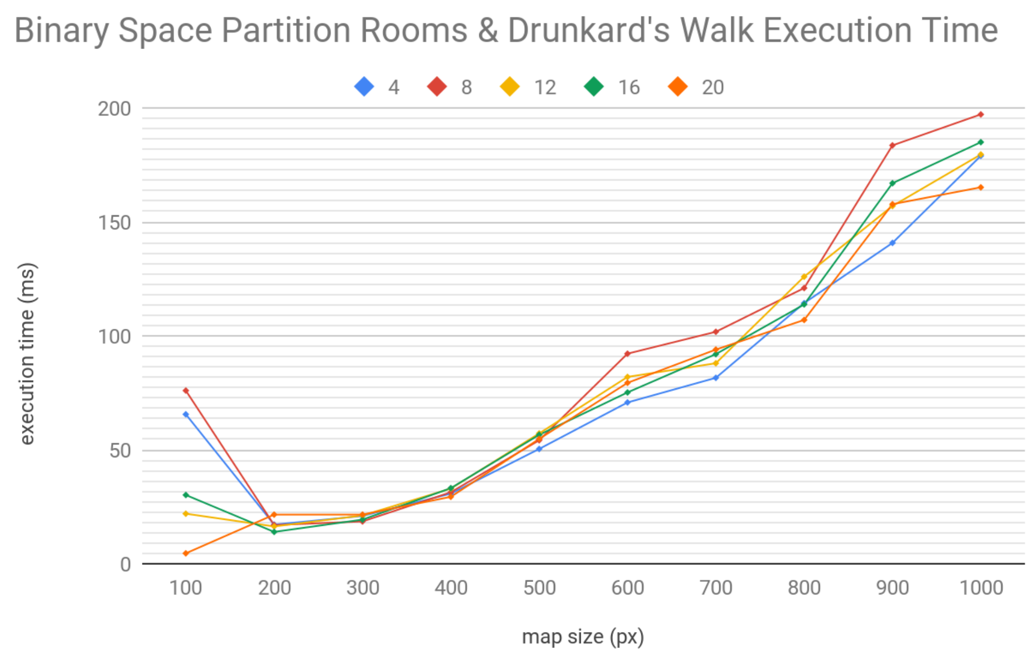

Fig 4.15: Line Chart showing execution time of Binary Space Partition Rooms and Drunkard's Walk at different room sizes and map sizes.

As can be seen in Fig 4.15, for room sizes 4px, 8px, 12px and 16px, at the lowest map sizes, the execution time decreases as map size increases. However, after map size 200px, the execution time then begins increasing as room size increases. At room size

20px, execution always increases as map size increases. Map size of 8px can be seen from the graph to have a execution time then other room sizes at most ranges.

Table 4.15 also shows that the execution time decreases for room sizes 4px, 8px, 12px and 16px at map size range 100px to 200px. Execution time also decreases for room sizes 4px and 8px at map size range 100px to 500px.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	173%	159%	716%	513%	3,491%
100px - 500px	400%	-23%	-29%	160%	87%	1,091%
100px - 200px	100%	-74%	-78%	-25%	-54%	370%
600px - 1,000px	67%	153%	114%	119%	146%	108%
900px - 1,000px	11%	27%	7%	14%	11%	5%

Table 4.15: Execution time increase of Binary Space Partition Rooms and Drunkard's Walk at different map size ranges for each room size.

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time increases as follows at different room sizes; 4px : -23%, 8px : -29%, 12px : 160%, 16px : 87%, 20px : 1,091%. When increasing map size from 600px to 1,000px (67% increase), the execution time increases as follows at different room sizes; 4px : 153%, 8px : 114%, 12px : 119%, 16px : 146%, 20px : 108%.

This shows that for all room sizes, at the map size range 600px to 1,000px, execution time increases at a faster rate then map size, which could indicate that the rate increases as map size increases.

4.2.3.2 Resource use (Big O)

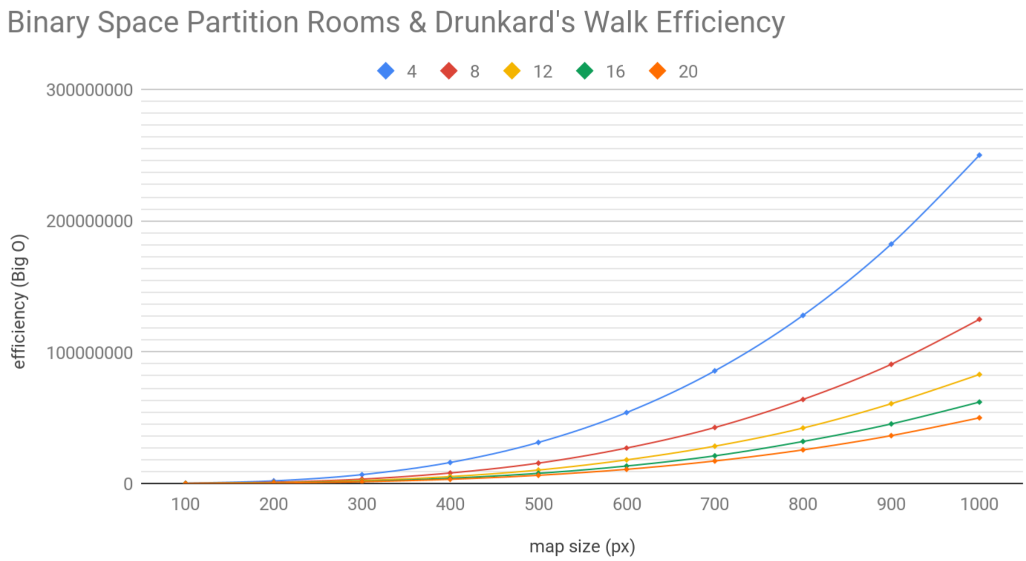

Fig 4.16: Smooth Line Chart showing resource use of Binary Space Partition Rooms and Drunkard's Walk at different room sizes and map sizes.

As shown in Fig 4.16, at all map sizes, higher room sizes use less resources, this indicates that resource use increases as room size decreases.

This trend can be seen when comparing resource use at each end of the chart. At map size of 100px, the resource use for each room size is as follows; 4px : 250,164, 8px : 120,134, 12px : 80,190, 16px : 60,290, 20px : 50,428. At map size of 1,000px, the resource use for each room size is as follows; 4px : 250,001,514, 8px : 125,000,812, 12px : 83,000,640, 16px : 62,000,626, 20px : 50,000,698.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	99,835%	103,951%	103,405%	102,737%	99,053%
100px - 500px	400%	12,392%	12,803%	12,683%	12,755%	12,295%
100px - 200px	100%	700%	733%	698%	697%	694%
600px - 1,000px	67%	363%	363%	361%	365%	363%
900px - 1,000px	11%	37%	38%	37%	37%	37%

Table 4.16: Resource use increase of Binary Space Partition Rooms and Drunkard's Walk at different map size ranges for each room size.

Table 4.16 that shows that while the amount of resources used increases as room size decreases, the rate of increase does not vary as much between room sizes.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use increases as follows at different room sizes; 4px : 12,392%, 8px : 12,803%, 12px : 12,683%, 16px : 12,755%, 20px : 12,295%. When increasing map size from 600px to 1,000px (67% increase), the resource use increases as follows at different room sizes; 4px : 363%, 8px : 363%, 12px : 361%, 16px : 365%, 20px : 363%.

This indicates that rate of increase of the resource use gets lower at higher map sizes. The difference in increases gets smaller at higher map sizes, as is also shown when looking at the highest map size range of 900px to 1,000px, where all room size increases are between 37% and 38%.

4.2.4 RRP and RPC

4.2.4.1 Execution Time

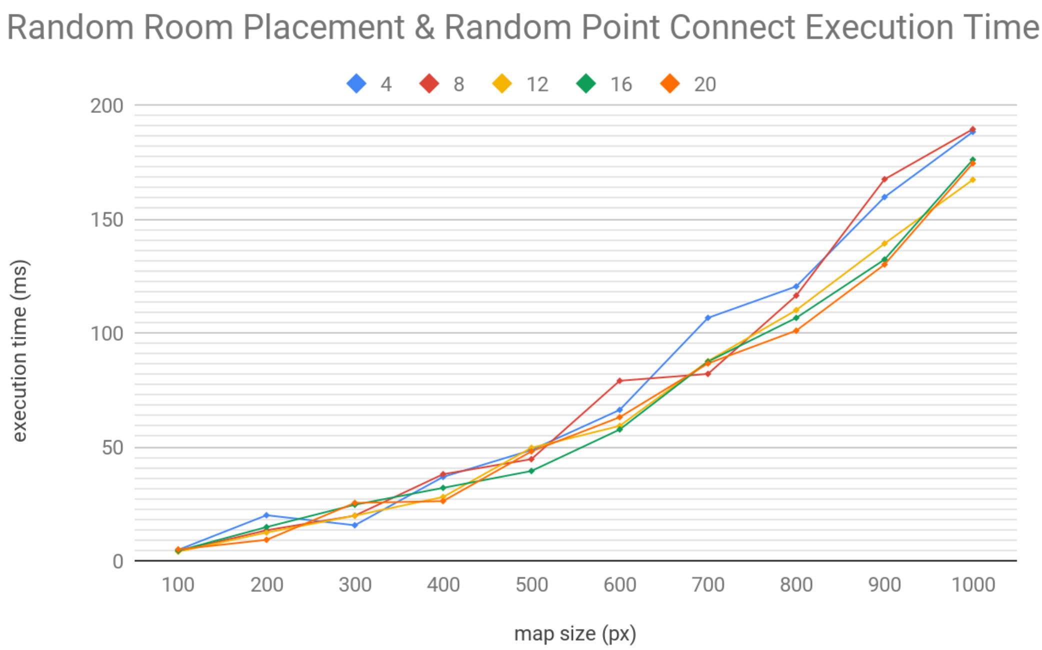

Fig 4.17: Line Chart showing execution time of Random Room Placement and Random Point Connect at different room sizes and map sizes.

As can be seen in Fig 4.17, the execution time increases as map size increases across all room sizes. At higher map sizes, room sizes of 4px and 8px appear to have higher execution times then the other room sizes, which could indicate that execution time is higher at lower room sizes.

When comparing the values at the lowest and highest map sizes used, at map size of

100px, room size of 12px has the lowest execution time (4ms) and room size of 20px has the highest (5ms). At map size of 1,000px, room size of 12px has the lowest execution time (167.2ms) and room size of 8px has the highest (189.4ms). This could indicate that room size of 8px has a higher execution time then the other room sizes.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	3,821%	4,410%	4,080%	3,900%	3,388%
100px - 500px	400%	913%	962%	1,140%	795%	860%
100px - 200px	100%	317%	219%	210%	236%	84%
600px - 1,000px	67%	184%	140%	182%	206%	177%
900px - 1,000px	11%	18%	13%	20%	33%	34%

Table 4.17: Execution time increase of Random Room Placement and Random Point Connect at different map size ranges for each room size.

Table 4.17 shows that at most map size ranges, the execution time increases at a faster rate than the map size for all room sizes, the exception is at room size 20px, at map size range 100px to 200px.

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time increases as follows at different room sizes; 4px : 913%, 8px : 962%, 12px : 1,140%, 16px : 795%, 20px : 860%. When increasing map size from 600px to 1,000px (67% increase), the execution time increases as follows at different room sizes; 4px : 184%, 8px : 140%, 12px : 182%, 16px : 206%, 20px : 177%.

This also indicates that the execution rate for all room sizes increases faster then the map size. The rate of increase of the execution time is highest at room sizes of 8px and 12px for most map size ranges. However, this varies which could indicate that room size does not have a strong impact on the rate of increase of the execution time.

4.2.4.2 Resource use (Big O)

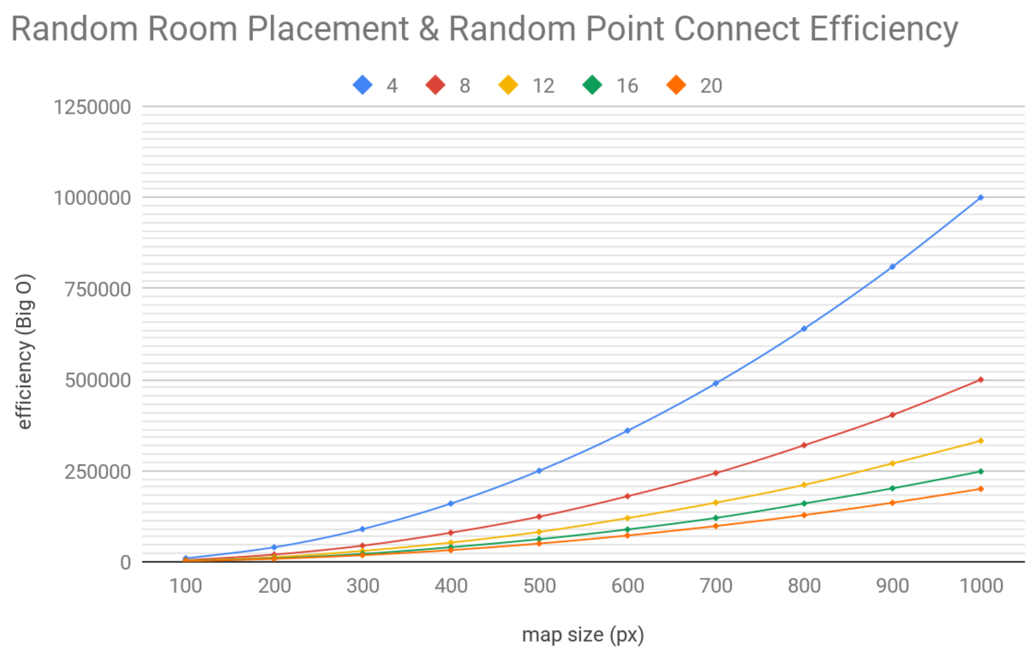

Fig 4.18: Smooth Line Chart showing resource use of Random Room Placement and Random Point Connect at different room sizes and map sizes.

As shown in Fig 4.18, at all map sizes, higher room sizes use less resources, this indicates that resource use increases as room size decreases.

This trend can be seen when comparing resource use at each end of the chart. At map size of 100px, the resource use for each room size is as follows; 4px : 10,016, 8px : 4,864, 12px : 3,344, 16px : 2,656, 20px : 2,400. At map size of 1,000px, the resource use for each room size is as follows; 4px : 1,000,016, 8px : 500,064, 12px : 332,144, 16px : 248,256, 20px : 200,400.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	9,884%	10,181%	9,833%	9,247%	8,250%
100px - 500px	400%	2,396%	2,451%	2,356%	2,244%	2,000%
100px - 200px	100%	300%	313%	287%	271%	250%
600px - 1,000px	67%	178%	178%	176%	179%	177%
900px - 1,000px	11%	23%	24%	23%	23%	23%

Table 4.18: Resource use increase of Random Room Placement and Random Point Connect at different map size ranges for each room size.

Table 4.18 that shows that while the amount of resources used increases as room size decreases, the rate of increase does not vary as much between room sizes.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use increases as follows at different room sizes; 4px : 2,396%, 8px : 2,451%, 12px : 2,356%, 16px : 2,244%, 20px : 2,000%. When increasing map size from 600px to 1,000px (67% increase), the resource use increases as follows at different room sizes; 4px : 178%, 8px : 178%, 12px : 176%, 16px : 179%, 20px : 177%.

This indicates that rate of increase of the resource use gets lower at higher map sizes. The difference in increases gets smaller at higher map sizes, as is also shown when looking at the highest map size range of 900px to 1,000px, where all room size increases are between 23% and 24%.

4.2.5 RRP and DW

4.2.5.1 Execution Time

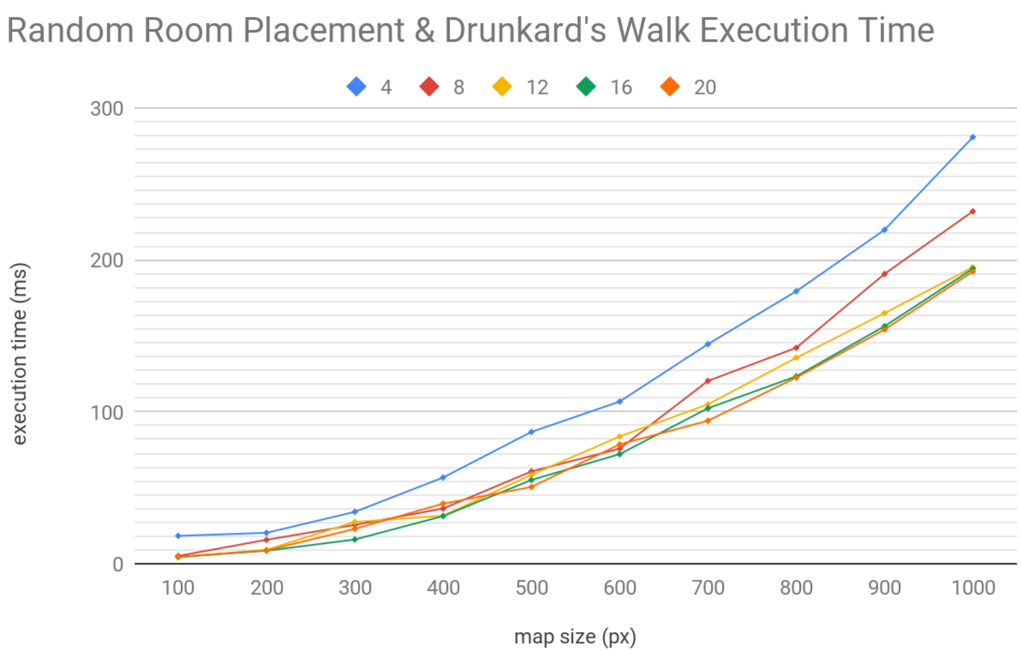

Fig 4.19: Line Chart showing execution time of Random Room Placement and Drunkard's Walk at different room sizes and map sizes.

As can be seen in Fig 4.19, the execution time increases as map size increases across all room sizes. At all map sizes, room sizes of 4px and 8px appear to have higher execution times then the other room sizes, which could indicate that execution time is higher at lower room sizes.

When comparing the values at the lowest and highest map sizes used, at map size of 100px, room size of 12px has the lowest execution time (4.2ms) and room size of 4px has the highest (18.4ms). At map size of 1,000px, room size of 20px has the lowest execution time (192.4ms) and room size of 4px has the highest (280.8ms). This shows that room size of 4px has a higher execution time then the other room sizes at both ends of the chart.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	1,426%	4,362%	4,548%	4,122%	4,083%
100px - 500px	400%	372%	1,069%	1,295%	1,100%	1,000%
100px - 200px	100%	11%	204%	119%	87%	91%
600px - 1,000px	67%	163%	206%	133%	169%	145%
900px - 1,000px	11%	28%	22%	18%	24%	25%

Table 4.19: Execution time increase of Random Room Placement and Drunkard's Walk at different map size ranges for each room size.

Table 4.19 shows that room size of 4px has the lowest rate of increase in execution time at map size ranges 100px to 200px and 100px to 500px, and the highest at map size range 900px to 1,000px. This indicates that the rate of increase of the execution time gets faster at higher map ranges.

When comparing the execution time increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the execution time increases as follows at different room sizes; 4px : 372%, 8px : 1,069%, 12px : 1,295%, 16px : 1,100%, 20px : 1,000%. When increasing map size from 600px to 1,000px (67% increase), the execution time increases as follows at different room sizes; 4px : 163%, 8px : 206%, 12px : 133%, 16px : 169%, 20px : 145%.

This shows that the rate of increase of the execution time gets slower at higher map size ranges for all room sizes except room size of 4px.

4.2.5.2 Resource use (Big O)

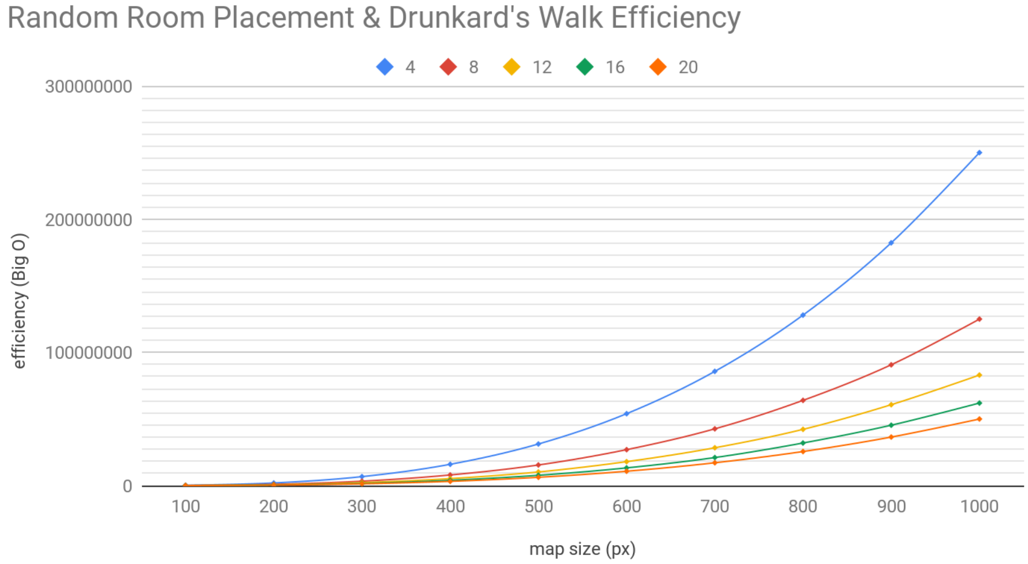

Fig 4.20: Smooth Line Chart showing resource use of Random Room Placement and Drunkard's Walk at different room sizes and map sizes.

As shown in Fig 4.20, at all map sizes, higher room sizes use less resources, this indicates that resource use increases as room size decreases.

This trend can be seen when comparing resource use at each end of the chart. At map size of 100px, the resource use for each room size is as follows; 4px : 250,016, 8px : 120,064, 12px : 80,144, 16px : 60,256, 20px : 45,400. At map size of 1,000px, the resource use for each room size is as follows; 4px : 250,000,016, 8px : 125,000,064, 12px : 83,000,144, 16px : 62,000,256, 20px : 50,000,400.

map size increase	map size % increase	% change at room 4px	% change at room 8px	% change at room 12px	% change at room 16px	% change at room 20px
100px - 1,000px	900%	99,894%	104,011%	103,464%	102,795%	110,033%
100px - 500px	400%	12,399%	12,810%	12,690%	12,762%	13,667%
100px - 200px	100%	700%	733%	699%	697%	782%
600px - 1,000px	67%	363%	363%	361%	365%	363%
900px - 1,000px	11%	37%	38%	37%	37%	37%

Table 4.20: Resource use increase of Random Room Placement and Drunkard's Walk at different map size ranges for each room size.

Table 4.20 that shows that while the amount of resources used increases as room size decreases, the rate of increase does not vary as much between room sizes.

When comparing the resource use increase on the left and right sides of the graph, when the map size is increased from 100px to 500px (400% increase), the resource use increases as follows at different room sizes; 4px : 12,399%, 8px : 12,810%, 12px : 12,690%, 16px : 12,762%, 20px : 13,667%. When increasing map size from 600px to 1,000px (67% increase), the resource use increases as follows at different room sizes; 4px : 363%, 8px : 363%, 12px : 361%, 16px : 365%, 20px : 363%.

This indicates that rate of increase of the resource use gets lower at higher map sizes. The difference in increases gets smaller at higher map sizes, as is also shown when looking at the highest map size range of 900px to 1,000px, where all room size increases are between 37% and 38%.

4.3 Summary of Results

In this section, the results of the key points of the charts discussed in previous sections will be summarized in tables and some notable differences between algorithm combinations will be identified.

Map size increase	Map Size % increase	BSP + BSP % increase	BSP + RPC % increase	BSP + DW % increase / decrease	RRP + RPC % increase	RRP + DW % increase
100px - 1,000px	900%	3,850%	2,098%	160%	3,185%	1,549%
100px - 500px	400%	791%	495%	-28%	774%	384%
100px - 200px	100%	200%	55%	-30%	122%	24%
600px - 1,000px	67%	170%	136%	177%	151%	117%
900px - 1,000px	11%	27%	20%	20%	4%	19%

Table 4.21: Percentage increase/decrease in execution time at different map size ranges.

Table 4.21, which summarises rate of change in the execution time for the selected map size ranges, shows that when looking at the range 100px - 1000px, BSP Rooms and DW has the lowest rate of increase in its execution time, increasing at a slower rate then the map size. The execution time of the other algorithms at this map size range all increase at more than twice the rate as the map size, indicating that it has a negative effect on execution time efficiency.

BSP Rooms and DW is the only algorithm combination to experience a decrease in execution time, at map size ranges 100px to 200px and 100px to 500px. At the highest map size range of 900px to 1,000px, RRP and DW is the only algorithm combination to increase at a slower rate then the map size.

Map size increase	Map Size % increase	BSP + BSP % increase	BSP + RPC % increase	BSP + DW % increase	RRP + RPC % increase	RRP + DW % increase
100px - 1,000px	900%	9,858%	9,537%	99,743%	9,659%	117,409%
100px - 500px	400%	2,390%	2,315%	12,381%	2,341%	14,589%
100px - 200px	100%	299%	290%	699%	293%	840%
600px - 1,000px	67%	178%	177%	363%	178%	363%
900px - 1,000px	11%	23%	23%	37%	23%	37%

Table 4.22: Percentage increase in resource use at different map size ranges.

As can be seen on Table 4.12, RPC and DW and BSP and DW have the same rate of increase in resource use across map size ranges 600px to 1,000px and 900px and 1,000px, indicating that DW might have a strong enough effect on the efficiency at those map ranges that the room selecting algorithm does not have an impact.

This is also supported by the fact that RPC and DW has the highest increase in resource use across all other map size ranges and BSP and DW has the second highest rate of increase.

5. ANALYSIS, EVALUATION AND DISCUSSION

In this section the results of the measurements for each algorithm will be discussed in further detail, focusing on possible causes for decreases or increases in efficiency and resource time.

5.1 Analysis of Results

In this section the results for the execution time and resource use measurements will be compared. The comparison will focus on which algorithms preformed best or worst for each measurement and attempt to identify trends and significant results. It will also attempt to analyse the room and corridor algorithms separately by comparing the performance of combinations that feature the same room or corridor algorithms.

5.1.1 Execution Time

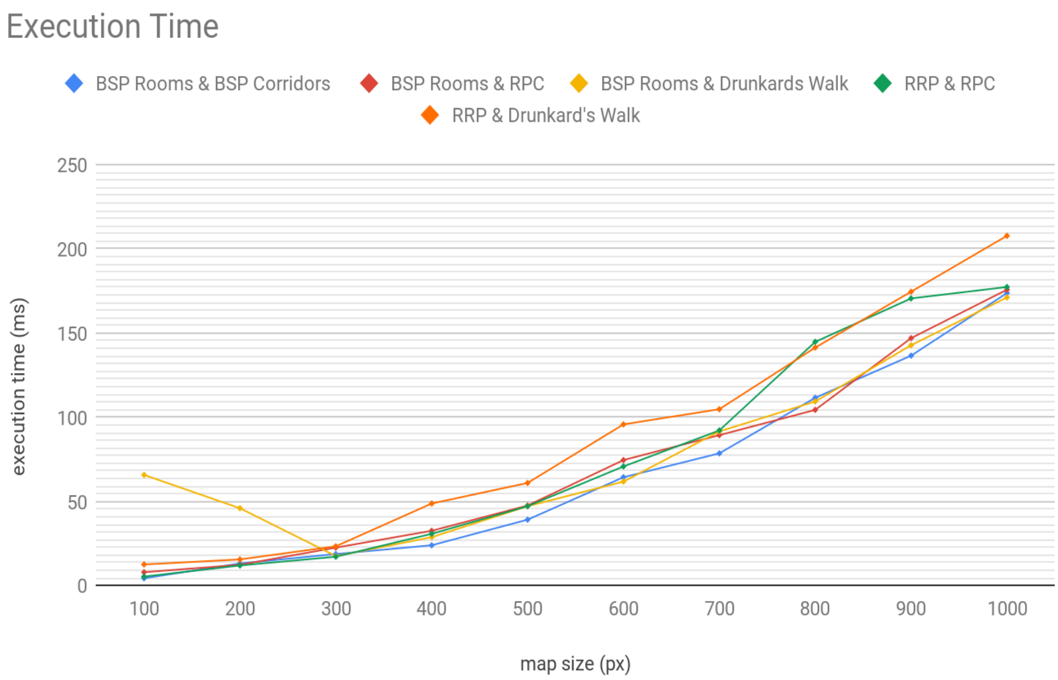

Fig 5.1: Line Chart showing execution time for all algorithm combinations when room size is 10px at different map sizes.

As can be seen in the Fig 5.1, the combination of Random Room Placement (RRP) and Drunkard's Walk (DW) has a noticeably higher execution time for most map sizes and at higher map sizes (600px to 1000px) the difference in execution rate get higher. At these higher map sizes, the RRP and Random Point Connect (RPC) has the second highest execution time across most map sizes.

The combination with the lowest execution time is BSP Rooms and BSP Corridors. At higher map sizes, the differences in execution time between the three combinations using BSP Rooms becomes lower.

5.1.1.1 Analysis of room algorithms

Since RRP is the common factor between the two combinations with the highest execution time, it can be assumed that it is the reason for the higher time. One possible reason for this is the way RRP is implemented.

When it creates a room, first it generates a random width and height and then attempts to place it on the map, if the placement fails, it will try to place the room again repeatedly until it is either successful or the number of tries reaches the tolerance value. If all attempts to place the room fail, it will attempt to generate a new width and height and then place the room again, repeating the previous process until the required number of rooms is reached. This means that the amount of possible times the algorithm attempts to place a room is *(tolerance_value x tolerance_value x number_of_rooms)*.

In contrast, the other room placement algorithm, Binary Space Partition (BSP) Rooms, divides the map into rectangles of a minimum width that is larger than the maximum room width. This means that the algorithm only needs to place each rooms once, meaning most of its execution time is used splitting the root rectangle into the leaf nodes, the amount of times a leaf node will attempt to be split is equal to the number of rooms placed on the map.

Another possible reason for the higher execution time of the algorithms combined with RRP is because of the way corridor algorithms connect rooms. Both DW and RPC connect rooms in the order they were placed on the map, which in the case of RRP is random, meaning that the algorithm could be trying to connect room on opposite sides of the map.

In contrast BSP Rooms divides the map and then places the rooms by loopholing through the tree and placing rooms in the bottom leaves, meaning that the rooms are often placed close to the same order they appear on the map, making the corridors shorter, lowering the execution time.

5.1.1.2 Analysis of corridor algorithms

The corridor algorithm with the lowest execution time is BSP Corridor algorithm. One possible reason for this is the way BSP Corridors connects rooms, by iterating through all the leaf nodes in the BSP Rooms tree and connecting pairs of children to each other, it then connects on of each children to the children of the parents sibling. This means that it draws the amount of corridors equal to number_of_rooms – 1.

In contrast, RPC and DW algorithm works by iterating through the array of rooms and connecting each one to the one proceeding it, so that each room is connected to 2 other rooms, meaning the amount of corridors is the equal to the number of rooms.

One possible reason the combination of RRP and DW has the highest execution time is also because of the way DW connects the rooms. For each pair of rooms it is connecting, it selects a random point on the border of each room being connected, and takes a “step”.

Each step increments the corridor in the appropriate and y directions towards the second point at a random length. This process is repeated until either the corridor reaches the second point or the maximum number of steps are reached.

Since RRP positions the rooms randomly, there is a high chance, that the rooms selected to be connected are further away from each other, meaning that DW has to take more steps and has a higher chance of having to start again, before reaching its destination.

5.1.2 Resource Use

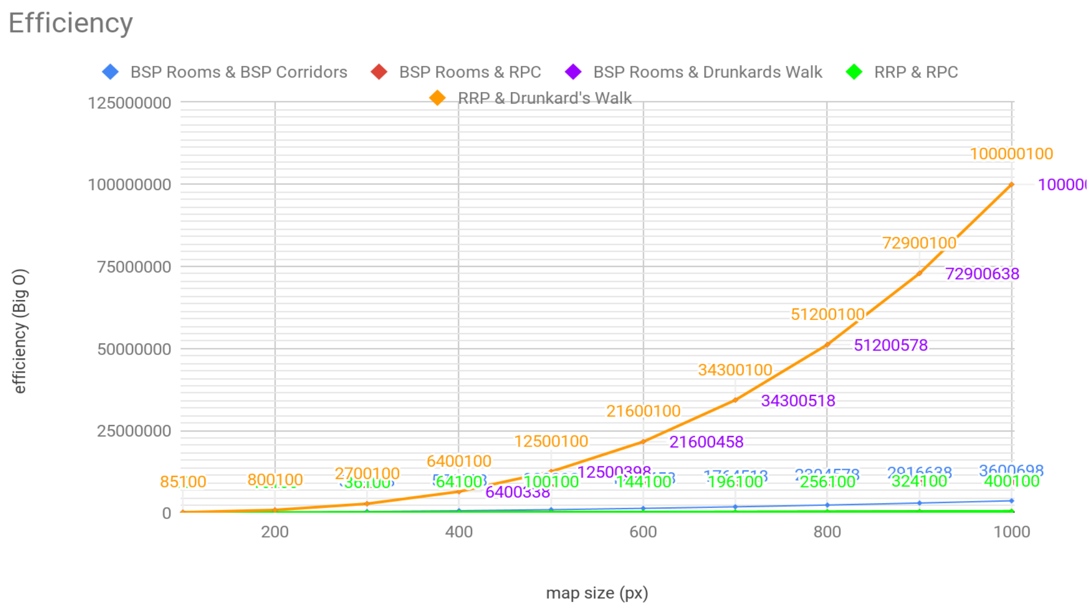

Fig 5.2: Smooth Line Chart showing resource use for all algorithm combinations when room size is 10px at different map sizes.

As shown in Fig 5.2, BSP Rooms and DW and RRP and DW have very high, nearly identical resource uses, this indicates that DW uses the most resources of any of the algorithms.

This is due to the fact that the efficiency of DW is affected by the maximum number of steps that it can take before finishing, which increases based on map size and is much higher then the limiters used in the other corridor placing algorithms.

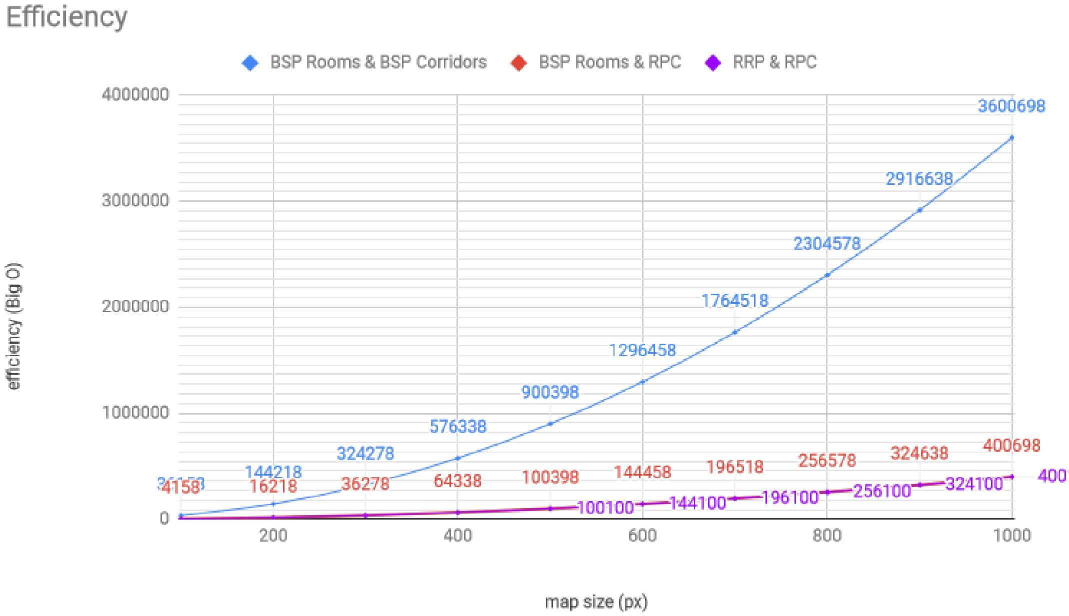

Fig 5.3: Smooth Line Chart showing resource use for algorithm combinations that do not include Drunkard's Walk when room size is 10px at different map sizes.

Fig 5.3 shows the resource use of the algorithm combinations that do not include DW since in the previous chart they were not visible enough to be able to meaningfully see the differences.

The highest algorithm combination is BSP Rooms and BSP Corridors, this is possibly due to the fact that efficiency of BSP Corridors is affected by both the number of children in the tree for BSP Rooms and the number of pairings it has to make and the size of the map. The efficiency of RPC is only affected by the map size and number of rooms.

The lowest algorithm combination is RRP and RPC, this is due to the fact that the random nature of the algorithms require less resources to be held. RRP only uses the maximum dimensions of the rooms as its resource and RPC only uses the map size and number of rooms.

5.2 Evaluation of Results

In this section the results of the analysis of the algorithms will be compared and evaluated. The evaluation will focus on the performance of the combinations and possible limitations in the way the results were collected.

5.2.1 Performance of algorithm combinations

This section will discuss the performance of each combination of algorithms, and discuss what conditions contribute to or detract from efficiency.

All algorithm combinations had significantly lower resource use when using higher rooms sizes, meaning resource efficiency is positively effected by room size. Since not all algorithms showed change when room size was varied, that could indicate a larger range of room sizes might have been needed.

RRP and DW has both the highest execution time and the highest resource use out of all the algorithm combinations. The execution time is high due to the fact that both algorithms rely on having multiple attempts to place rooms and draw corridors, whereas the other algorithms only require one attempt per successfully placed room/corridor. This indicates that RRP and DW are the least efficient of the algorithm combinations, based on the efficiency parameters used in this experiment.

The execution time for RRP and DW is also higher when using smaller room sizes, so the recommend use for this algorithm combination would be when using small maps and large rooms.

BSP Rooms and BSP Corridors has the lowest execution time and its efficiency lies in the middle of the algorithms, making it the most efficient overall. While execution time is negatively effected by increases in map size, the rate of increase of the execution time gets slower as map size increases.

RRP and RPC has the lowest resource use however it has the second highest execution time, indicating that execution time and resource use are not indicative of each other.

5.2.2 Limitations of results

The proposed formulas for resource use using Big O efficiency have draw backs for this particular experiment. The formulas are based on an analysis of the algorithms that identified the key factors contributing to resource use, discussed in Chapter 3.

These formulas result in a constant number for each map size and room size used when the algorithms are used with a fixes number of rooms. The resulting formula for each combination of algorithm is the result of the two constants produced by the room and corridor placing algorithms added together.

However, this does not take into account the effect the algorithms haver on each other. For example, random room placement causes the corridor algorithms to be longer, which should use more resources, however corridor length is not taken in to account in any of the corridor placement resource sue formulas. How this could be calculated is not in the scope of this research project, however, possibly maximum distance between rooms could be taken in to account for resource use to mitigate this issue.

For many of the algorithms, room size did not impact execution time, this may be because this measurement was effected by the limit in the amount of rooms allowed to be placed on the map.

The limit on the number of rooms was based on the map size divided by the room size multiplied by two. This means that as map size increases the number of rooms allowed increases at a set rate, and as room size, the number of allowed rooms decreases.

This might mean that less rooms are placed when using larger room sizes, which could be one of the reasons for the higher efficiency. This could also mean that the execution time for larger rooms is faster because the algorithm is placing less rooms.

5.3 Discussion of Results

This section will discuss if any of the finding of the experiment are significant / relevant, and weather or not the research goals were achieved. It will also discuss other aspects of the PCG algorithms, such as differences in map layout and the limitations of the research.

5.3.1 Research Goals

The goal of this project was to identify PCG algorithms for creating 2D maps to use in a comparison and to identify parameters to compare them under for efficiency. Then to run a comparison and identify significant differences in efficiency and possible causes. This section will discuss if these goals have been achieved and any possible relevance of the results.

The algorithms identified were 5 combinations of PCG algorithms, made up of 2 room placing algorithms and 3 corridor placing algorithms. These were chosen so that the study could run a comparison looking at the 5 combinations as a whole, and also compare how the individual algorithms effect each other.

The efficiency measurement chosen were execution time and resource use and a comparison was successfully run, however no signifiant connection between resource use and execution time was found.

The comparison did discover significant differences between the algorithms, which could impact their usability in a video game. For example, BSP Rooms and BSP corridors has the lowest execution time, meaning is speed is an important factor in the application or if the algorithm needs to be run multiple times in a single playthrough, this algorithm would be appropriate.

Resource use is always lower when using larger room sizes, which means that if resource use is negativity impacting a game when using these algorithms, larger room sizes would be recommended.

5.3.2 Map Layouts

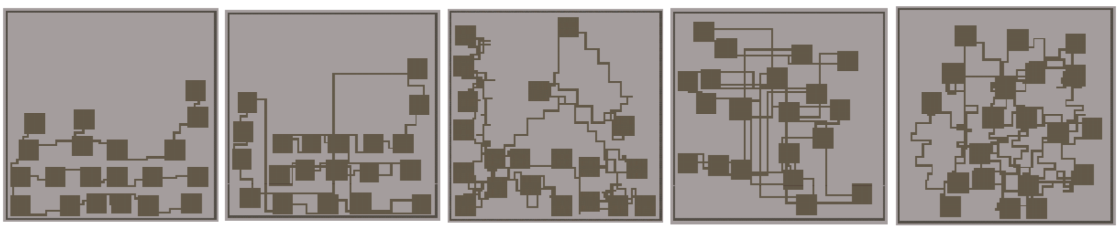

Fig 5.4: From left to right: BSP Rooms & BSP Corridors, BSP Rooms & RPC, BSP Rooms & DW, RRP & RPC and RRP & DW.

Fig 5.4 shows each of the maps produced by the algorithms at map size 100px and room size 10px, as can be seen the manner in which the rooms and corridors are placed has a large effect on the final layout of the maps.

BSP Rooms and BSP Corridors creates a more orderly map, in most maps produced, the corridors will connect the rooms adjacent to each other, and will rarely intersect with other corridors. This is because BSP Corridors, which can only be used in conjunction with BSP Rooms, works by connecting the first child in each leaf node to the first child in the neighbouring leaf node, creating ordered paths connecting the closest rooms to each other.

For maps produced using BSP Rooms, the rooms are usually grouped together, with a large amount of map space unused. This is because when the desired number of children is reached to fit all the rooms, the algorithm will stop splitting the map, meaning that sometime one half of the map will be split into as many sections as can fit and the other side will only be split once or twice.

Rooms placed by RRP are more unevenly spread across the map then BSP Rooms. The corridors generated when used with RRP are also usually longer, this is because the corridor placing algorithms both connect rooms in the order they are placed on the map. In the case of RRP is random, rooms placed after each other in chronological order can be at opposite sides of the map, causing longer corridors.

Maps using RPC have mostly straight corridors, due the fact that the algorithm connects the rooms in order they are placed in as direct a rout as possible. Maps using DW have many more jagged meandering pathways, due to the fact that drunkard's walk breaks each corridor into individual steps, that navigate in random vertex towards the end point.

6. CONCLUSION

This chapter will review the experiment ad a whole, focusing on the outcomes of the implementation and weather or not it successfully reached its goals and answered the research question.

6.1 Research Overview

The research first discussed the current use of PCG in video games and looked at different applications and algorithms used in previous research. It then took five algorithms for creating 2D maps and ran a comparison on the results based on parameters used in a previous comparison of 2D map algorithms.

Based on the findings from the comparison BSP Rooms and BSP Corridors were the most efficient algorithm combination overall, it had the lowest execution time and its resource use is in the middle of the results for all the algorithms. RRP and DW was the least efficient algorithm combination, with the highest execution time and resource use.

6.2 Problem Definition

The research question asked was:

Which Procedural Content Generation algorithm for generating 2D maps in video games compares best for efficiency?

The algorithms chosen for the experiment were a combination of two room generating algorithms, Binary Space Partitioning (BSP) Rooms and Random Room Placement (RRP) and three corridor generating algorithms, Binary Space Partitioning (BSP) Corridors, Random Point Connect (RPC) and Drunkard's Walk (DW).

The measurements chosen for efficiency were execution time and Big O measurement.

6.3 Contributions and Impact

The experiment takes the algorithms discussed in the 2017 paper by R. Baron and applied the execution time measurement used by Hilliard et al. in their 2017 paper in a comparison of efficiency.

It also used the Big O measurement detailed in the R. Baron paper to measure resource use alongside execution time. It showed that resource use and execution time do not affect each other when the algorithms are run at the chosen map and room sizes.

It demonstrated a possible flaw in the way the formulas for the room and corridors are combined. Since the analysis ran to identify the resource use was only done on each algorithm individually, it did not take into account how the algorithms might affect each others resource use.

It showed that the combination of RRP and DW performed poorly when tested for efficiency using the parameters in this project, which could negativity impact how it is used in game development, as this means it could cause games using it to run slower and less efficiently.

6.4 Future Work & Recommendations

In the original experiment by Hilliard et al, the number of rooms was also measured alongside execution time. As show in the tables in APPNDIX I – V, the number of rooms placed for each algorithm combination was recorded during this experiment as well.

However due to the fact that the number of rooms was controlled by a formula, the number was determined based on the size of rooms and maps used and was the same between all algorithms.

If the algorithms had been allowed to place as many rooms as they could, the number of rooms could have been used as a parameter in the comparison, to see how number of rooms effected execution time and efficiency at different map and room sizes.

However, this would have involved running all parts of the experiment twice. The first run through if the experiment, which is the one preformed in chapter 4, involved using a cap on number for rooms placed on the maps in order to compare the affect of map size and room size on execution time and efficiency in isolation.

The second run of the experiment would retake the measurements when not using a limit on number of rooms, these results would then be compared with the previous results to identify the impact of number of rooms.

Execution time could also be measured alongside the number of rooms to test how effected it is by the number and which algorithms can place the most rooms at the shortest execution time.

Another possible future area of study could focus on measuring the variations in the shapes of the maps produced, this could be done in several ways. A start and end point could be added to random rooms on the map and the path between them could be measured and compared between the algorithms to test for variety in the room layouts. The shape of the map could analysed by working out the number of possible paths available between the start and end points.

In the experiment ran by R. Baron, the maps were generated in both 2D and 3D, in order to compare test if the algorithms are applicable to both types of games. This comparison could be ran alongside the one done in this project to compare the differences in efficiency between 2D and 3D implementations of the algorithms.

More work is also needed to look at how these algorithms work when used in an actual game, adding other game elements to the map, such as enemies or items, and then testing for efficiency might change the results.

BIBLIOGRAPHY

Baron, J. R. (2017). Procedural Dungeon Generation Analysis and Adaptation. In *Proceedings of the SouthEast Conference* (pp. 168–171). New York, NY, USA: ACM. https://doi.org/10.1145/3077286.3077566

Brewer, N. (2017). Computerized Dungeons and Randomly Generated Worlds: FromRogue to Minecraft[Scanning Our Past]. *Proceedings of the IEEE*, *105*(5), 970–977. https://doi.org/10.1109/JPROC.2017.2684358

Fernández-Vara, C., & Thomson, A. (2012). Procedural Generation of Narrative Puzzles in Adventure Games: The Puzzle-Dice System (p. 12). Presented at the Proceedings of the The third workshop on Procedural Content Generation in Games, ACM. https://doi.org/10.1145/2538528.2538538

Fernández-Vara, C. (2014). Creating Dreamlike Game Worlds Through Procedural Content Generation. In *Seventh Intelligent Narrative Technologies Workshop*. Retrieved from https://www.aaai.org/ocs/index.php/INT/INT7/paper/view/9250

Grinblat, J., & Bucklew, C. B. (2017). Subverting Historical Cause & Effect: Generation of Mythic Biographies in Caves of Qud. In *Proceedings of the 12th International Conference on the Foundations of Digital Games* (pp. 76:1–76:7). New York, NY, USA: ACM. https://doi.org/10.1145/3102071.3110574

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural Content Generation for Games: A Survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, *9*(1), 1:1–1:22. https://doi.org/10.1145/2422956.2422957

Hilliard, N., Salis, J., & ELAarag, H. (2017). Algorithms for Procedural Dungeon Generation. *J. Comput. Sci. Coll.*, *33*(1), 166–174.

Johnson, L., Yannakakis, G. N., & Togelius, J. (2010). Cellular Automata for Real-time Generation of Infinite Cave Levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (pp. 10:1–10:4). New York, NY, USA: ACM. https://doi.org/10.1145/1814256.1814266

Smith, G. (2014). Understanding Procedural Content Generation: A Design-centric

Analysis of the Role of PCG in Games. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems* (pp. 917–926). New York, NY, USA: ACM. https://doi.org/10.1145/2556288.2557341

Smith, G., Othenin-Girard, A., Whitehead, J., & Wardrip-Fruin, N. (2012). PCG-based Game Design: Creating Endless Web. In *Proceedings of the International Conference on the Foundations of Digital Games* (pp. 188–195). New York, NY, USA: ACM. https://doi.org/10.1145/2282338.2282375

Smith, G. (2017). What Do We Value in Procedural Content Generation? In *Proceedings of the 12th International Conference on the Foundations of Digital Games* (pp. 69:1–69:2). New York, NY, USA: ACM. https://doi.org/10.1145/3102071.3110567

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 172–186. https://doi.org/10.1109/TCIAIG.2011.2148116

Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011). What is Procedural Content Generation?: Mario on the Borderline. In *Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games* (pp. 3:1–3:6). New York, NY, USA: ACM. https://doi.org/10.1145/2000919.2000922

APPENDIX I - BSP Rooms & BSP Corridors Results

map size	room size	execution time 1	execution time 2	execution time 3	execution time 4	execution time 5	execution time avg	num rooms	efficiency
100	*10*	4	5	4	5	4	**4.4**	20	36158
200	*10*	8	29	9	11	9	**13.2**	40	144218
300	*10*	14	14	36	15	15	**18.8**	60	324278
400	*10*	23	24	23	25	25	**24**	80	576338
500	*10*	35	32	52	38	39	**39.2**	100	900398
600	*10*	47	54	82	73	66	**64.4**	120	1296458
700	*10*	61	85	78	79	90	**78.6**	140	1764518
800	*10*	116	116	83	119	124	**111.6**	160	2304578
900	*10*	146	163	131	95	148	**136.6**	180	2916638
1000	*10*	181	188	154	175	171	**173.8**	200	3600698
100	*4*	15	4	4	16	4	**8.6**	50	90164
200	*4*	19	8	20	21	9	**15.4**	100	360314
300	*4*	27	29	26	29	16	**25.4**	150	810464
400	*4*	38	45	40	41	35	**39.8**	200	1440614
500	*4*	48	55	52	49	54	**51.6**	250	2250764
600	*4*	87	77	63	66	79	**74.4**	300	3240914
700	*4*	82	79	107	85	80	**86.6**	350	4411064
800	*4*	117	109	118	104	119	**113.4**	400	5761214
900	*4*	149	156	134	124	125	**137.6**	450	7291364
1000	*4*	154	145	184	145	149	**155.4**	500	9001514
100	*8*	7	6	5	4	4	**5.2**	24	43334
200	*8*	11	8	11	12	27	**13.8**	50	180212
300	*8*	14	15	15	19	36	**19.8**	74	399884
400	*8*	53	27	47	27	48	**40.4**	100	720362
500	*8*	41	70	44	42	51	**49.6**	124	1116434
600	*8*	71	47	57	96	79	**70**	150	1620512
700	*8*	110	92	106	93	63	**92.8**	174	2192984
800	*8*	112	115	116	97	105	**109**	200	2880662
900	*8*	135	134	138	135	126	**133.6**	224	3629534
1000	*8*	162	161	191	203	161	**175.6**	250	4500812
100	*12*	4	7	4	5	5	**5**	16	28990
200	*12*	8	51	9	8	8	**16.8**	32	115438
300	*12*	16	18	17	15	15	**16.2**	50	270292
400	*12*	24	24	24	22	23	**23.4**	66	475540
500	*12*	70	82	32	53	33	**54**	82	738388
600	*12*	72	70	77	84	48	**70.2**	100	1080442
700	*12*	81	105	82	87	90	**89**	116	1462090

800	*12*	125	82	124	110	114	**111**	132	1901338
900	*12*	141	131	149	169	151	**148.2**	150	2430592
1000	*12*	160	192	176	192	238	**191.6**	166	2988640
100	*16*	4	5	5	5	5	**4.8**	12	21890
200	*16*	9	10	10	65	10	**20.8**	24	86726
300	*16*	15	17	15	14	44	**21**	36	194762
400	*16*	22	61	24	41	26	**34.8**	50	360404
500	*16*	53	32	62	35	51	**46.6**	62	558440
600	*16*	70	73	70	80	70	**72.6**	74	799676
700	*16*	94	98	145	61	75	**94.6**	86	1084112
800	*16*	105	86	91	93	104	**95.8**	100	1440554
900	*16*	138	155	148	143	132	**143.2**	112	1814990
1000	*16*	185	154	234	154	176	**180.6**	124	2232626
100	*20*	4	25	5	5	4	**8.6**	10	18428
200	*20*	8	10	8	8	8	**8.4**	20	72458
300	*20*	16	15	16	34	14	**19**	30	162488
400	*20*	23	23	23	48	22	**27.8**	40	288518
500	*20*	57	54	40	75	41	**53.4**	50	450548
600	*20*	71	72	60	66	89	**71.6**	60	648578
700	*20*	85	95	90	71	102	**88.6**	70	882608
800	*20*	101	103	104	110	109	**105.4**	80	1152638
900	*20*	126	140	137	125	179	**141.4**	90	1458668
1000	*20*	165	155	185	150	177	**166.4**	100	1800698

APPENDIX II - BSP Rooms & Random Point Connect Results

map size	room size	execution time 1	execution time 2	execution time 3	execution time 4	execution time 5	execution time avg	num rooms	efficiency
100	*10*	21	6	5	4	4	**8**	20	4158
200	*10*	8	28	10	9	7	**12.4**	40	16218
300	*10*	25	13	13	18	44	**22.6**	60	36278
400	*10*	22	46	22	49	24	**32.6**	80	64338
500	*10*	51	57	63	35	32	**47.6**	100	100398
600	*10*	68	90	54	94	67	**74.6**	120	144458
700	*10*	92	103	103	62	87	**89.4**	140	196518
800	*10*	103	105	94	114	106	**104.4**	160	256578
900	*10*	130	141	158	175	131	**147**	180	324638
1000	*10*	164	164	190	186	175	**175.8**	200	400698
100	*4*	4	4	5	5	4	**4.4**	50	10164
200	*4*	27	8	8	8	8	**11.8**	100	40314
300	*4*	14	40	15	18	14	**20.2**	150	90464
400	*4*	41	23	54	27	31	**35.2**	200	160614
500	*4*	33	65	40	50	32	**44**	250	250764
600	*4*	46	53	102	67	50	**63.6**	300	360914
700	*4*	87	61	84	94	80	**81.2**	350	491064
800	*4*	117	105	110	140	94	**113.2**	400	641214
900	*4*	123	129	148	152	113	**133**	450	811364
1000	*4*	164	161	149	151	145	**154**	500	1001514
100	*8*	4	4	4	4	5	**4.2**	24	4934
200	*8*	8	8	8	37	8	**13.8**	50	20212
300	*8*	14	15	16	16	17	**15.6**	74	44684
400	*8*	22	23	22	48	21	**27.2**	100	80362
500	*8*	54	32	70	50	59	**53**	124	124434
600	*8*	66	70	55	85	65	**68.2**	150	180512
700	*8*	60	73	96	106	93	**85.6**	174	244184
800	*8*	114	119	119	110	126	**117.6**	200	320662
900	*8*	123	159	176	98	164	**144**	224	403934
1000	*8*	156	165	178	158	169	**165.2**	250	500812
100	*12*	4	4	4	5	4	**4.2**	16	3390
200	*12*	8	8	7	9	8	**8**	32	13038
300	*12*	14	17	14	15	18	**15.6**	50	30292
400	*12*	21	21	28	43	26	**27.8**	66	53140
500	*12*	34	68	32	66	32	**46.4**	82	82388
600	*12*	66	66	50	56	80	**63.6**	100	120442
700	*12*	84	86	93	79	104	**89.2**	116	162890

800	*12*	103	143	130	122	112	**122**	132	211738
900	*12*	131	177	193	132	124	**151.4**	150	270592
1000	*12*	151	209	195	154	174	**176.6**	166	332640
100	*16*	4	6	5	5	5	**5**	12	2690
200	*16*	8	10	8	47	7	**16**	24	9926
300	*16*	15	32	19	19	19	**20.8**	36	21962
400	*16*	29	29	25	29	61	**34.6**	50	40404
500	*16*	47	70	37	42	66	**52.4**	62	62440
600	*16*	72	48	87	85	64	**71.2**	74	89276
700	*16*	88	71	116	92	106	**94.6**	86	120912
800	*16*	136	117	140	120	139	**130.4**	100	160554
900	*16*	141	145	178	159	167	**158**	112	202190
1000	*16*	174	184	171	175	171	**175**	124	248626
100	*20*	5	5	6	4	5	**5**	10	2225
200	*20*	30	10	9	8	10	**13.4**	20	8458
300	*20*	14	19	49	15	19	**23.2**	30	18488
400	*20*	50	23	51	32	44	**40**	40	32518
500	*20*	41	77	40	32	68	**51.6**	50	50548
600	*20*	53	85	68	89	59	**70.8**	60	72578
700	*20*	113	89	99	101	102	**100.8**	70	98608
800	*20*	146	116	165	116	81	**124.8**	80	128638
900	*20*	135	146	148	160	120	**141.8**	90	162668
1000	*20*	163	160	156	177	168	**164.8**	100	200698

APPENDIX III - BSP Rooms & Drunkard's Walk Results

map size	room size	execution time 1	execution time 2	execution time 3	execution time 4	execution time 5	execution time avg	num rooms	efficiency
100	*10*	16	128	4	135	46	**65.8**	20	100158
200	*10*	8	8	196	10	8	**46**	40	800218
300	*10*	14	15	15	28	16	**17.6**	60	2700278
400	*10*	45	29	23	24	23	**28.8**	80	6400338
500	*10*	54	37	58	52	35	**47.2**	100	12500398
600	*10*	73	55	48	66	67	**61.8**	120	21600458
700	*10*	90	87	93	92	96	**91.6**	140	34300518
800	*10*	107	110	114	107	109	**109.4**	160	51200578
900	*10*	137	135	147	138	157	**142.8**	180	72900638
1000	*10*	182	153	188	170	163	**171.2**	200	100000698
100	*4*	6	5	287	22	8	**65.6**	50	250164
200	*4*	9	10	29	10	28	**17.2**	100	2000314
300	*4*	19	17	18	17	34	**21**	150	6750464
400	*4*	28	28	46	26	26	**30.8**	200	16000614
500	*4*	39	39	67	46	61	**50.4**	250	31250764
600	*4*	80	74	54	74	72	**70.8**	300	54000914
700	*4*	93	96	69	75	75	**81.6**	350	85751064
800	*4*	115	132	117	122	86	**114.4**	400	128001214
900	*4*	149	141	144	136	134	**140.8**	450	182251364
1000	*4*	192	192	169	168	174	**179**	500	250001514
100	*8*	5	148	105	116	6	**76**	24	120134
200	*8*	29	8	30	9	9	**17**	50	1000212
300	*8*	15	16	15	32	15	**18.6**	74	3330284
400	*8*	44	26	23	40	24	**31.4**	100	8000362
500	*8*	44	62	33	64	68	**54.2**	124	15500434
600	*8*	72	150	101	68	70	**92.2**	150	27000512
700	*8*	133	86	98	96	96	**101.8**	174	42630584
800	*8*	129	118	121	110	127	**121**	200	64000662
900	*8*	235	182	146	194	161	**183.6**	224	90720734
1000	*8*	232	188	189	174	203	**197.2**	250	125000812
100	*12*	7	5	76	17	5	**22**	16	80190
200	*12*	12	9	27	9	25	**16.4**	32	640238
300	*12*	17	35	18	19	17	**21.2**	50	2250292
400	*12*	36	29	33	42	25	**33**	66	5280340
500	*12*	80	55	47	69	35	**57.2**	82	10250388
600	*12*	97	78	82	80	73	**82**	100	18000442
700	*12*	127	70	82	74	87	**88**	116	28420490

800	*12*	206	93	113	123	95	**126**	132	42240538
900	*12*	208	142	158	122	155	**157**	150	60750592
1000	*12*	225	156	152	178	187	**179.6**	166	83000640
100	*16*	7	5	125	8	6	**30.2**	12	60290
200	*16*	11	8	9	8	34	**14**	24	480326
300	*16*	20	16	30	15	16	**19.4**	36	1620362
400	*16*	32	44	24	23	43	**33.2**	50	4000404
500	*16*	76	35	62	66	44	**56.6**	62	7750440
600	*16*	79	94	92	55	56	**75.2**	74	13320476
700	*16*	94	96	87	88	95	**92**	86	21070512
800	*16*	112	114	118	109	116	**113.8**	100	32000554
900	*16*	135	134	203	167	196	**167**	112	45360590
1000	*16*	165	188	185	157	230	**185**	124	62000626
100	*20*	5	4	5	4	5	**4.6**	10	50428
200	*20*	10	10	69	11	8	**21.6**	20	400458
300	*20*	14	19	15	34	26	**21.6**	30	1350488
400	*20*	24	26	25	42	30	**29.4**	40	3200518
500	*20*	53	60	41	82	38	**54.8**	50	6250548
600	*20*	73	88	106	53	77	**79.4**	60	10800578
700	*20*	83	64	115	101	107	**94**	70	17150608
800	*20*	114	82	115	91	133	**107**	80	25600638
900	*20*	138	164	174	156	157	**157.8**	90	36450668
1000	*20*	166	170	167	158	165	**165.2**	100	50000698

APPENDIX IV - Random Room Placement & Random Point Connect Results

map size	room size	execution time 1	execution time 2	execution time 3	execution time 4	execution time 5	execution time avg	num rooms	efficiency
100	*10*	4	6	4	7	6	**5.4**	20	4100
200	*10*	7	8	9	28	8	**12**	40	16100
300	*10*	14	17	18	20	17	**17.2**	60	36100
400	*10*	23	24	26	48	33	**30.8**	80	64100
500	*10*	56	41	39	62	38	**47.2**	100	100100
600	*10*	76	74	86	59	59	**70.8**	120	144100
700	*10*	97	74	95	125	70	**92.2**	140	196100
800	*10*	143	153	182	120	126	**144.8**	160	256100
900	*10*	160	169	171	164	189	**170.6**	180	324100
1000	*10*	206	173	169	163	176	**177.4**	200	400100
100	*4*	5	5	5	5	4	**4.8**	50	10016
200	*4*	44	8	8	32	8	**20**	100	40016
300	*4*	15	16	16	16	15	**15.6**	150	90016
400	*4*	25	25	49	37	48	**36.8**	200	160016
500	*4*	38	40	62	42	61	**48.6**	250	250016
600	*4*	57	56	79	78	61	**66.2**	300	360016
700	*4*	123	108	106	81	115	**106.6**	350	490016
800	*4*	123	124	106	132	117	**120.4**	400	640016
900	*4*	156	162	161	157	162	**159.6**	450	810016
1000	*4*	183	195	175	224	164	**188.2**	500	1000016
100	*8*	4	4	5	4	4	**4.2**	24	4864
200	*8*	9	33	8	8	9	**13.4**	50	20064
300	*8*	14	19	14	37	15	**19.8**	74	44464
400	*8*	47	24	44	51	24	**38**	100	80064
500	*8*	36	31	61	38	57	**44.6**	124	124064
600	*8*	72	141	52	76	54	**79**	150	180064
700	*8*	94	70	71	87	88	**82**	174	243664
800	*8*	117	117	109	123	116	**116.4**	200	320064
900	*8*	223	173	122	162	157	**167.4**	224	403264
1000	*8*	192	168	194	194	199	**189.4**	250	500064
100	*12*	4	4	5	3	4	**4**	16	3344
200	*12*	9	8	8	8	29	**12.4**	32	12944
300	*12*	15	25	30	14	15	**19.8**	50	30144
400	*12*	24	27	24	43	22	**28**	66	52944
500	*12*	39	74	44	55	36	**49.6**	82	82144
600	*12*	54	49	70	72	51	**59.2**	100	120144

700	*12*	65	91	79	94	109	**87.6**	116	162544
800	*12*	124	114	110	116	86	**110**	132	211344
900	*12*	136	152	146	138	124	**139.2**	150	270144
1000	*12*	179	165	167	157	168	**167.2**	166	332144
100	*16*	5	4	4	4	5	**4.4**	12	2656
200	*16*	8	8	42	8	8	**14.8**	24	9856
300	*16*	39	15	15	37	17	**24.6**	36	21856
400	*16*	48	22	45	23	22	**32**	50	40256
500	*16*	37	59	34	33	34	**39.4**	62	62256
600	*16*	49	48	69	71	51	**57.6**	74	89056
700	*16*	88	73	98	88	90	**87.4**	86	120656
800	*16*	110	113	117	84	109	**106.6**	100	160256
900	*16*	132	142	136	137	114	**132.2**	112	201856
1000	*16*	163	204	198	154	161	**176**	124	248256
100	*20*	4	4	5	5	7	**5**	10	2400
200	*20*	10	8	10	9	9	**9.2**	20	8400
300	*20*	44	15	38	16	14	**25.4**	30	18400
400	*20*	22	23	23	41	22	**26.2**	40	32400
500	*20*	64	35	51	35	55	**48**	50	50400
600	*20*	74	71	50	48	72	**63**	60	72400
700	*20*	98	96	73	67	99	**86.6**	70	98400
800	*20*	109	111	92	109	84	**101**	80	128400
900	*20*	159	140	115	129	107	**130**	90	162400
1000	*20*	165	181	158	220	148	**174.4**	100	200400

APPENDIX V - Random Room Placement & Drunkard's Walk

map size	room size	execution time 1	execution time 2	execution time 3	execution time 4	execution time 5	execution time avg	num rooms	efficiency
100	*10*	11	5	6	5	36	**12.6**	20	85100
200	*10*	34	11	10	11	12	**15.6**	40	800100
300	*10*	20	17	18	44	18	**23.4**	60	2700100
400	*10*	72	28	55	29	60	**48.8**	80	6400100
500	*10*	72	52	76	40	65	**61**	100	12500100
600	*10*	83	119	101	93	83	**95.8**	120	21600100
700	*10*	135	115	88	83	103	**104.8**	140	34300100
800	*10*	156	139	135	166	111	**141.4**	160	51200100
900	*10*	168	183	165	188	169	**174.6**	180	72900100
1000	*10*	229	202	200	196	212	**207.8**	200	100000100
100	*4*	67	6	5	7	7	**18.4**	50	250016
200	*4*	12	51	12	14	13	**20.4**	100	2000016
300	*4*	48	43	25	31	24	**34.2**	150	6750016
400	*4*	61	41	66	42	74	**56.8**	200	16000016
500	*4*	83	121	62	91	77	**86.8**	250	31250016
600	*4*	111	113	118	107	85	**106.8**	300	54000016
700	*4*	136	143	155	142	147	**144.6**	350	85750016
800	*4*	170	187	184	186	170	**179.4**	400	128000016
900	*4*	222	216	207	210	244	**219.8**	450	182250016
1000	*4*	254	258	283	301	308	**280.8**	500	250000016
100	*8*	5	6	5	4	6	**5.2**	24	120064
200	*8*	10	35	11	11	12	**15.8**	50	1000064
300	*8*	18	19	42	12	37	**25.6**	74	3330064
400	*8*	29	30	31	53	39	**36.4**	100	8000064
500	*8*	67	77	44	48	68	**60.8**	124	15500064
600	*8*	69	64	90	91	65	**75.8**	150	27000064
700	*8*	117	140	126	130	89	**120.4**	174	42630064
800	*8*	148	148	142	138	135	**142.2**	200	64000064
900	*8*	208	173	193	180	200	**190.8**	224	90720064
1000	*8*	212	199	241	215	293	**232**	250	125000064
100	*12*	4	4	4	5	4	**4.2**	16	80144
200	*12*	9	10	9	9	9	**9.2**	32	640144
300	*12*	17	54	16	18	33	**27.6**	50	2250144
400	*12*	27	29	28	47	27	**31.6**	66	5280144
500	*12*	74	72	40	68	39	**58.6**	82	10250144
600	*12*	84	89	96	59	91	**83.8**	100	18000144
700	*12*	107	105	104	119	90	**105**	116	28420144

800	*12*	132	165	131	119	131	**135.6**	132	42240144
900	*12*	169	163	172	155	166	**165**	150	60750144
1000	*12*	199	191	192	203	191	**195.2**	166	83000144
100	*16*	5	5	5	4	4	**4.6**	12	60256
200	*16*	8	8	9	9	9	**8.6**	24	480256
300	*16*	16	16	17	15	16	**16**	36	1620256
400	*16*	25	35	25	47	25	**31.4**	50	4000256
500	*16*	74	64	38	62	38	**55.2**	62	7750256
600	*16*	78	79	71	53	80	**72.2**	74	13320256
700	*16*	98	107	102	98	107	**102.4**	86	21070256
800	*16*	132	140	127	120	98	**123.4**	100	32000256
900	*16*	154	151	162	163	152	**156.4**	112	45360256
1000	*16*	199	194	205	195	178	**194.2**	124	62000256
100	*20*	5	4	4	5	5	**4.6**	10	45400
200	*20*	9	9	8	9	9	**8.8**	20	400400
300	*20*	16	37	16	28	18	**23**	30	1350400
400	*20*	27	68	27	28	48	**39.6**	40	3200400
500	*20*	69	38	66	41	39	**50.6**	50	6250400
600	*20*	76	102	79	85	51	**78.6**	60	10800400
700	*20*	101	97	78	99	96	**94.2**	70	17150400
800	*20*	129	122	118	119	125	**122.6**	80	25600400
900	*20*	157	147	163	157	147	**154.2**	90	36450400
1000	*20*	169	221	214	166	192	**192.4**	100	50000400