# R&D Assignment

## 3D Dungeon Generator

Hogeschool van Amsterdam
Gameplay Engineering

Michael Spiegel
04.10.2022

# Contents

# 1 Introduction

For my research project I chose a 3D dungeon generator that should generate a new dungeon every time the game is started. Furthermore, I decided to make it possible to walk through the dungeon. For this I want to add a player that will be placed in a randomly chosen start room. To make the whole dungeon more interesting and not leave it as an empty place, I plan to add some scenery like torches. Additionally the dungeon should also contain enemies for which I try to implement a combat system. The focus lies on a performant generation of the dungeon and on the look. As a combat system can take much time to implement and needs a lot of testing it could be possible throughout the project that I will put more effort in the dungeon itself. For this I would then only implement a player to walk through the dungeon. Through this project I want to understand how the algorithms work that are used for dungeon generation. Furthermore, I want to get knowledge about combat systems in games.

# 2 Research

## 2.1 Existing algorithms for dungeon generation

There exist several algorithms which can be used to create dungeons. In this chapter three algorithms will be described and in the end compared.

### 2.1.1 Binary Space Partitioning

The binary space partitioning algorithm is very popular for creating dungeons because it is a fast algorithm and not that hard to implement. With this algorithm the dungeon is represented through a binary tree structure. The initial space will be split in each iteration of the algorithm. With this algorithm it is possible to create a variety of dungeons each time the code is run. For this it is necessary that specific parts of the split have to be randomised. The split can be horizontal or vertical. Figure 2.1 shows how a dungeon generated with Binary Space Partitioning looks like. (Williams, n.d.)

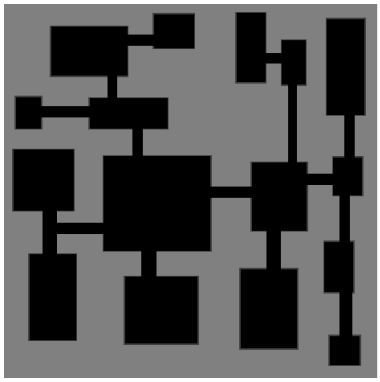

Figure 2.1: Example of a BSP generated dungeon
Source: Williams, n.d.

### 2.1.2 Cellular Automata

The Cellular automata algorithm uses a grid of cells to build a dungeon. Each cell has a reference to its neighbour cells and a defined state for the time $t = 0$. With the help of predefined rules the state for $t = t + 1$ can be calculated. Depending on the defined rules and the cell states there will be patterns that occur from time to time. A cell it self represents a space where the player can walk but also a space where the player can not walk for example rocks. As you can see in figure 2.2 Cellular Automata creates more cave like dungeons without real rooms. (Williams, n.d.)

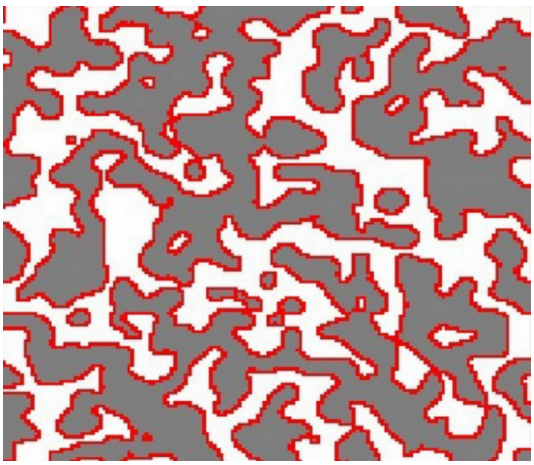

Figure 2.2: Example of a Cellular Automata generated dungeon
Source: Williams, n.d.

### 2.1.3 Delaunay Triangulation

The last algorithm is the delaunay triangulation algorithm for dungeon generation. This algorithm also uses cells but in this case a random rectangle is created inside of each cell. After this step the algorithm separates the room and prevents them from overlapping. All cells that are exceeding a threshold become rooms. To connect all rooms a graph for all the room centre points is constructed. Additionally to the delaunay triangulation algorithm it is necessary to generate a minimum spanning tree of the originally create graph to remove cycles. In figure 2.3 you can see how a dungeon generated with the delaunay triangulation looks like. Compared to the other two algorithms it is more similar to the result of the binary space partition generated dungeon because both use rooms. (Williams, n.d.)

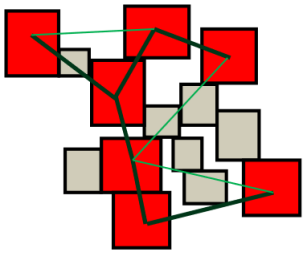

Figure 2.3: Example of a Delaunay Triangulation generated dungeon
Source: Williams, n.d.

## 2.2 Comparison of the algorithms

Now let's compare those algorithms to explain which one fits best for the purpose of this project. The cellular automata algorithm generates more cave like dungeons which is not applicable to the goal of this project. The dungeon should consist of rooms which only the binary space partition and the delaunay triangulation algorithm provides.

To compare those two algorithms let's have a look on the performance. Figure 2.4 shows the performance of the binary space partitioning algorithm. This

algorithm is slow when less rooms are used for the dungeon but has its strength in the generation of more complex dungeons that consists of a huge amount of rooms.

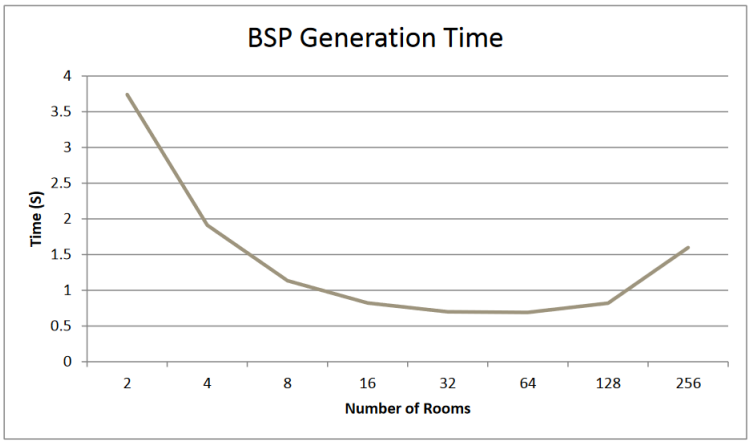


Figure 2.4: Performance of the Binary Space Partitioning algorithm
Source: Williams, n.d.

Figure 2.5 show that compared to the other algorithm the delaunay triangulation algorithm is faster when it comes to dungeons with less rooms. On the other side this algorithm gets slower with the increasing amount of rooms. This means, that this algorithm fits best if the dungeon will only consist of a few rooms.

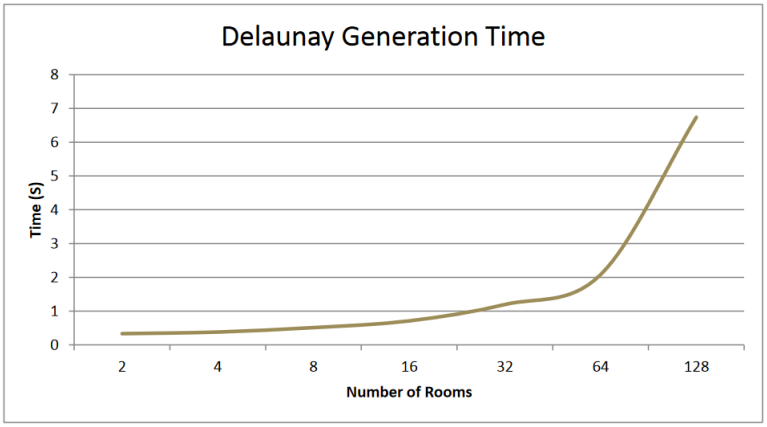


Figure 2.5: Performance of the Delaunay Triangulation algorithm
Source: Williams, n.d.

# 3 Implementation

## 3.1 Dungeon Generator

### 3.1.1 Creating rooms

### 3.1.2 Adding corridors

### 3.1.3 Adding walls

### 3.1.4 Determining Start and end room

### 3.1.5 Adding Scenery

## 3.2 Combat system

### 3.2.1 Player Movement

### 3.2.2 Enemy Movement

# 4 Conclusion and Reflection

# List of Figures

# Bibliography

Williams, N. (n.d.). An Investigation in Techniques used to Procedurally Generate Dungeon Structures. http://www.nathanmwilliams.com/files/AnInvestigationIntoDungeonGeneration.pdf