Daniel Szczypior

T00171641

Supervisor: Dr. Pat Doody

Procedural Building Generation

Final Year Project - COMPUTING WITH GAMES DEVELOPMENT

Table of Contents

[Chapter 1: Introduction 3](#_Toc500909187)

[Chapter 2: Procedural generation 4](#_Toc500909188)

[2.1 Introduction 4](#_Toc500909189)

[2.2 History of Procedural Generation 4](#_Toc500909190)

[2.3 Advantages of procedural generated elements 5](#_Toc500909191)

[2.4 Disadvantages of procedural generated elements 5](#_Toc500909192)

[2.5 Games that use procedural Generation 6](#_Toc500909193)

[2.5.1 Diablo Series 6](#_Toc500909194)

[2.5.2 Left 4 Dead Series 6](#_Toc500909195)

[2.5.3 Spore 6](#_Toc500909196)

[2.5.4 No Man’s Sky 6](#_Toc500909197)

[2.5.5 Minecraft 6](#_Toc500909198)

[2.5.6 Astroneer 7](#_Toc500909199)

[2.5.7 Borderlands 7](#_Toc500909200)

[2.5.8 The Binding of Isaac 7](#_Toc500909201)

[2.5.9 XCOM series 7](#_Toc500909202)

[2.5.10 Daggerfall 7](#_Toc500909203)

[2.5.11 Elite 7](#_Toc500909204)

[2.5.12 Rouge 8](#_Toc500909205)

[2.5.13 Dwarf Fortress 8](#_Toc500909206)

[2.6 Software that use procedural Generation 8](#_Toc500909207)

[2.6.1 Terragen 8](#_Toc500909208)

[2.6.2 SpeedTree 8](#_Toc500909209)

[Chapter 3: Procedural Generation Algorithms 9](#_Toc500909210)

[3.1 Introduction 9](#_Toc500909211)

[3.2 Perlin Noise 9](#_Toc500909212)

[3.2.1 Example of Perlin Noise with Fractal Brownian Motion 10](#_Toc500909213)

[3.3 Fractals 11](#_Toc500909214)

[3.4 Voronoi Diagram 12](#_Toc500909215)

[3.4.1 Lloyd's relaxation algorithm 13](#_Toc500909216)

[3.5 Diamond Square 13](#_Toc500909217)

[3.6 Biomes 14](#_Toc500909218)

[3.7 Seed 14](#_Toc500909219)

[3.8 Comparison of algorithms <TO BE DONE LATER> 15](#_Toc500909220)

[Chapter 4: Methodology & Design 15](#_Toc500909221)

[4.1 Research 15](#_Toc500909222)

[4.2 Research Question 15](#_Toc500909223)

[4.3 Proposed Solution 15](#_Toc500909224)

[4.4 Vision Document 16](#_Toc500909225)

[4.4.1 Purpose 16](#_Toc500909226)

[4.4.2 Scope 16](#_Toc500909227)

[4.4.3 User Environment 16](#_Toc500909228)

[Chapter 5: Implementation 17](#_Toc500909229)

[5.1 Prototype 17](#_Toc500909230)

[5.1.1 Progression 18](#_Toc500909231)

[References 19](#_Toc500909232)

# Chapter 1: Introduction

# Chapter 2: Procedural generation

## 2.1 Introduction

Procedurally generated element is created automatically by computer algorithms. For example, take a function that draws a rectangle in the upper left corner of the screen. The rectangle not generated procedurally will usually be an image loaded from a file, but the rectangle procedurally generated will be a drawing function that will draw four linked together straight lines. The procedural approach allows for easy modification of procedure parameters. Using random numbers generators is very intuitive. Procedurally generated components are a useful way to make games last longer through randomly generated levels, items and other elements diversifying a gameplay.

## 2.2 History of Procedural Generation

The most famous video game in history was Tetris. For the first time, logic games did not require us to find the right solution. Instead, it offered an endless stream of blocks in several different shapes, which had to be arranged accordingly. To date, most popular logic games like Bejeweled, Candy Crush, and 2048 use procedural generation.

In 1984, the first attempt was made to create a digital universe. It were David Braben and Ian Bell, who gave the world the first true Space Sim, Elite. The title gave the players eight galaxies, each of which consisted of 256 planets. All these attractions were available on 8-bit machines in a simple vector pseudo 3D.



Over time, disk space has ceased to be a problem, and the gaming world has slowly become fascinated with film narrative, while the developers have boldly taken advantage of the possibility of directing the player's experiences and starting to control every aspect of their production.

Diablo series is a typical example of this. The cult Hack 'n' Slash of 1996 not only offered immense undergrounds full of creatures for extermination, but also guaranteed that each trip into the depths of the earth would be a different experience. The game itself builds levels and randomly places opponents and treasures.

The Japanese RPG has offered the player so many years. "Random encounters", that is random battles with random opponents, while players are moving around the map. Most RPG games, like the Skyrim, randomly select the majority of the items they find in closed boxes. The computer also has the power of decision in the most famous series of strategy games Civilization. It is precisely the mechanisms of procedural generation that correspond to the shape of the map and the distribution of raw materials and competitive nations. The general parameters of this process are set before start of the game.

Procedural generation is today a suitable solution for independent developers. Big studios can afford large teams working on the development of the expanded elements of the game world, such as Skyrim and the Witcher 3. However, when one person is working on the game, it is easier for this person to do some computer work. It was Marcus "Notch" Persson, who made Minecraft. In this case, at the beginning of the game, the game itself builds the whole world, after which player will move. No one bothered that the title did not offer carefully designed levels, especially since it was up to players to do what they would do with the land which was offered.

## 2.3 Advantages of procedural generated elements

There are many advantages to procedural generated elements, no external data, no additional importers and parsers. Several times smaller size of the element - instead of data, only input parameters for algorithm are specified. Also possibility of random selection of parameters and generic generalization algorithms - the same set of instructions can be used to generate character names and city names.

## 2.4 Disadvantages of procedural generated elements

There are of course disadvantages of procedural generated elements, such as high CPU time consumption when generating components - loading them as external data generally takes much less time. No direct control over game content - many resources can be created easier and faster in specialized programs, producing them yourself gives direct control over the quality of the resource produced. Necessity of meticulous selection of parameters - improperly selected parameters and the lack of proper rules may lead to little reliable generator results.

## 2.5 Games that use procedural Generation

### 2.5.1 Diablo Series

Uses a partially random two-dimensional terrain in conjunction with the random placement of enemies and objects.

### 2.5.2 Left 4 Dead Series

Random character placement, random enemies placement (spawned out of character sight), controls the rate zombies spawn and checks what player is experiencing to ensure to not overwhelm players. Powerups such as pistols, ammunition, medkits etc. Music is also partially random. Zombies and cars are also created by instancing of in game entities which allows thousands of possible results.

### 2.5.3 Spore

Music is procedurally generated, same goes for worlds. All entities are generated by other players and then shared, they are generated in in-game editor which uses procedural generation and procedural animation.

### 2.5.4 No Man’s Sky

In its assumptions, a total sandbox with a huge, procedurally generated world, one of the most important elements of production is exploration on an almost gigantic scale. Each planet has distinctive and unique species of flora and fauna and offer other minerals to collect. How ambitious the project is No Man's Sky best shows the number of planets to visit, because they are supposed to be 18 trillion. On the other hand, No Man's Sky is to occupy 6 GB, most of which are audio files.

### 2.5.5 Minecraft

Generates entire worlds from a seed and uses Perlin noise algorithm. It starts out by painting basic topographical map, later adds noise to create details like lakes, shrubbery and animals. At later stage, it’s procedural generation was modified by adding the uses of biomes and fractals to further advance it’s possibilities.

### 2.5.6 Astroneer

Entire planets are generated procedurally based on their type e.g. Terrain, barren, radiated, arid, exotic etc. (biomes) and players received tools for transforming land, as if they were made of plasticine.

### 2.5.7 Borderlands

Weapon models, behaviour and their stats are created procedurally. You can get revolver that shots missiles, sniper rifle that shots everything from homing darts to rockets.

### 2.5.8 The Binding of Isaac

Rooms are procedurally generated every time level start but the content of the room is generated when player enter said room.

### 2.5.9 XCOM series

While the XCOM: Enemy Unknown and its expansion XCOM: Enemy Within where almost every map, buildings and obstacles were created procedurally the XCOM 2 uses parcel system that places pre build buildings and objects onto it, which allows for better control over the achieved result.

### 2.5.10 Daggerfall

While very few things are generated procedurally during runtime, many of the towns, location and dungeons were pre-generated procedurally by workstations and then tweaked by hand. Many of the side quest were generated procedurally during run time and its quest engine is capable of linking dungeons dynamically to quest. NPC can be created with random face and class and placed anywhere where there is a vacant house or a dungeon. As for the items they can also be generated as quest items or as one that belongs to NPC. Dialog lines can also be created procedurally.

### 2.5.11 Elite

Star systems procedurally generated from a fixed seed, which generated the description of the star system, including its economy, government, tech level, population and description of every planet. There were 256 starts per galaxy with the total number of 8 galaxies.

### 2.5.12 Rouge

A progenitor of all roguelikes and one of the earliest examples of procedural generation, released in 1980. Every time player descends a set of stairs new lever is generated procedurally. Every time players dies entire adventure is restarted and created afresh.

### 2.5.13 Dwarf Fortress

Created by two brothers, disgusted with the ASCII graphics, the title has earned not only the status of the most complex game ever created, but also deserves to be part of the permanent exhibition of the prestigious MOMA in New York. It creates not only the whole world, but also the whole terrain, the location of cities, forms, caves, natural deposits, etc., as well as all its history for several thousand years. All these events can be read, although these are rather simple messages. One of the stories of the game invoked by the creators. The brave dwarf decided to sacrifice his life in defence of the fort. After some time, his mother became depressed and stopped contacting the rest of the group. After a few weeks she decided to finish with her and drown herself in a nearby river.

## 2.6 Software that use procedural Generation

### 2.6.1 Terragen

Scenery generator which can be used to create renderings and animations of landscapes. On the cover of American Newsweek on April 16, 2001, an image created in Terragen appeared. In addition, it was also used in the making of the Brandy Norwood video clip "What About Us?". The program is used by many photographers and artists. The area in the program is generated from a two-dimensional heightmap. It is possible to import and export these maps as images for use in other programs.



### 2.6.2 SpeedTree

Group of vegetation programming and modelling software, it creates foliage, it's animations and architecture are real time created for video games and demanding real time simulations. It was used in Iron Man 3, Avatar movie, Pirates of the Caribbean: Dead Men Tell No Tales, Captain America: Civil War and also games like Assassin's Creed Unity, Far Cry 4, Horizon Zero Dawn, The Witcher 3: Wild Hunt and many more. The technique is designed to realistically reproduce the behaviour of vegetation. Every tree, bush and grass behave like in the real world. Under the influence of the wind, the trees bend, the bushes rustle and the grass diverges from the animals or the player. It is very useful for programmers, because they do not have to create an environment from scratch, they only use a ready-made technique. This system was based on C ++ libraries. (Lorditch, 2015)

# Chapter 3: Procedural Generation Algorithms

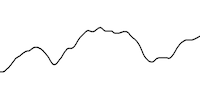
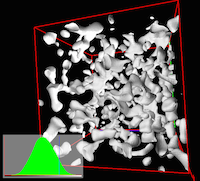
## 3.1 Introduction

There are many procedural generation algorithm, and they all have their own strength and flaws. Some are better for certain types of jobs while some are good for everything but do not excel in any direction.

## 3.2 Perlin Noise

Developed by Ken Perlin in 1983. It is algorithm used in procedural content generation. It can be used for creation of any sort of wave-like material or texture. For creation Minecraft-like terrain, water, clouds, fire effects. Mostly used in 2 and 3 dimensions but can be extended to 4 dimension rather trivially (So you can animate it). Can also be used for 1 dimension.

1 Dimension



3 Dimension

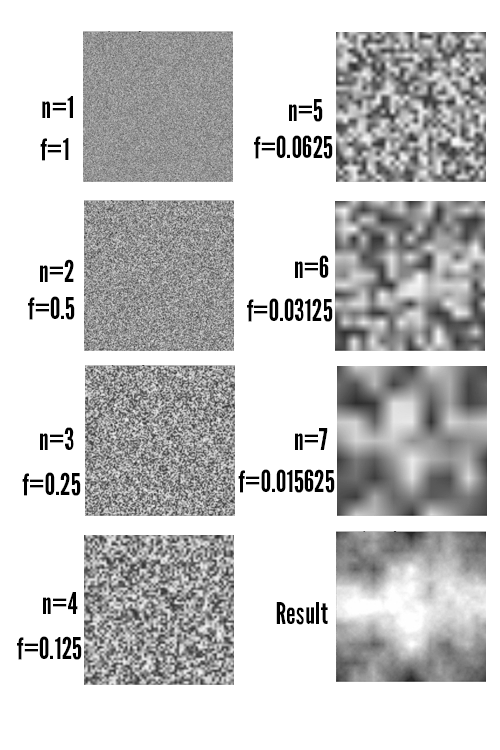
2 Dimension

The Perlin noise generator algorithm relies on interpolative arrays of pseudorandom noise to be combined with variable accuracy. Such a solution, unlike a cellular automaton, allows you to give fields values from the selected range, not just zero-one. In practice, the Perlin noise generation function returns a two-dimensional image in shades of grey. When making a simple interpretation, one can assume that the brightness of individual fields is proportional to their virtual height as elements of the world. If to choose the value corresponding to the height of the water level, it is easy to determine which fields are under the surface and which above. (Flafla2, 2014)

While Perlin noise looks more natural than other algorithm it doesn’t fully express the irregularities that one might expect in nature, like mountains and hills or even rocks. To allow that octaves to be used. Fractal Brownian Motion, work really well with Perlin Noise, it is a combination of multiple steps (octaves) of Perlin Noise, or another noise function, each witch different frequency and amplitude. The variation in frequency from step to step is called lacunarity and the variation in amplitude from a step to step is called gain. (Lowe, 2015)

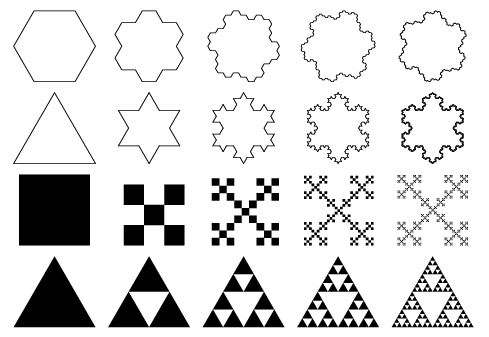
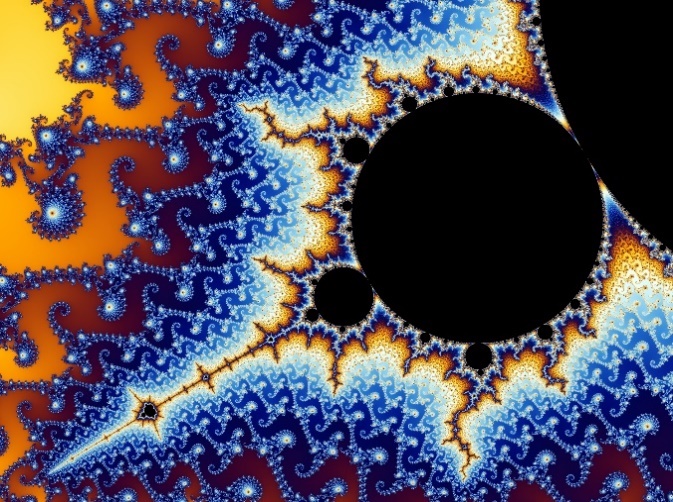
### 3.2.1 Example of Perlin Noise with Fractal Brownian Motion

At first using a pseudorandom number generator, we create a basic noise map of a certain size (AxB). Each bitmap/step is called octave. (Image n=1) Then create a new map using the basic section of n² of smaller size (A/n \* B/n) and scale it to the size of initial bitmap using interpolation function of the cosine. Then repeat last two steps for every new octave until desired accuracy is acquired. At last combine the resulting bitmaps by adding the pixel values multiplied by the frequency assigned to each image. For the initial n, the frequency of the bitmap is f = 1.0, while for the remaining bitmaps it is f (n) = f (n + 1) / 2. (Herman, 2009)



Perlin Noise stages

## 3.3 Fractals

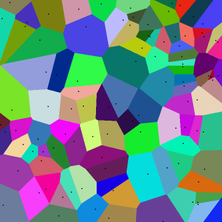
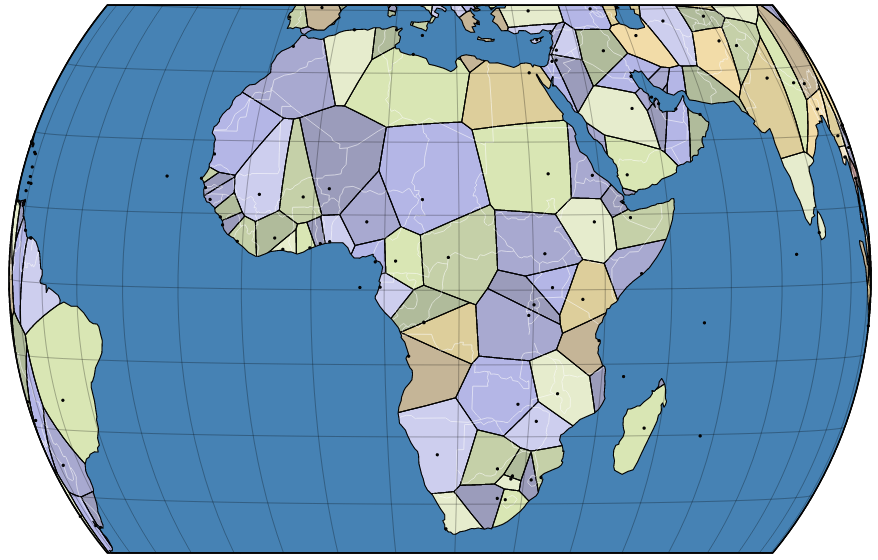
This is a never ending pattern. It is an infinitely complex pattern that is self-similar across different scales. It is created by repeating a simple process over and over in an ongoing feedback loop. Trees, hurricanes, clouds, rivers etc. can be generated by a computer calculating simple equation over and over. They are popular in mathematical visualisation since they look stunning even when created with use of simple pattern. You can zoom into it and patter and shape will continue repeating forever. (What are Fractals?, 2014)

Example of Fractal

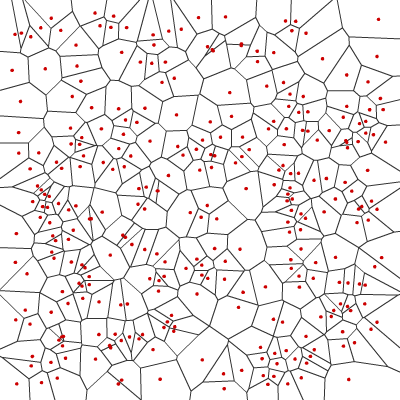
Stages of Fractal Creation

## 3.4 Voronoi Diagram

This are also called also Dirichlet tessellation. It has many features useful in procedural texture generation and is often used to simulate structures having a cellular structure. In a very simple way, this algorithm works by drawing a predetermined number of feature points on the surface of the texture and then assigning each pixel a brightness proportional to its distance from the nearest checkpoint. Brightness can also be calculated based on the sum of the distances from the set number of nearest control points.

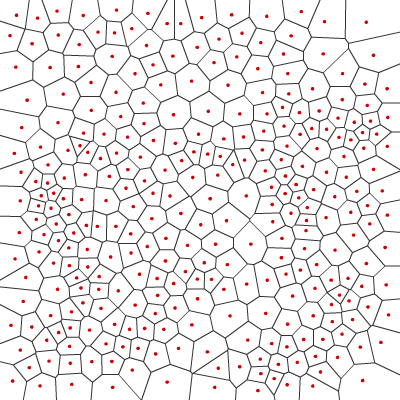
 

The first step is to generate random polygons but that will result in too obstinate result.



No Lloyd relaxation

To achieve more elegant looking outcome Lloyd relaxation algorithm is to be used which will result in more evenly distributed outcome.



With Lloyd relaxation

It is also advised to run algorithm several times to get even more appealing results. (Amit, 2010)

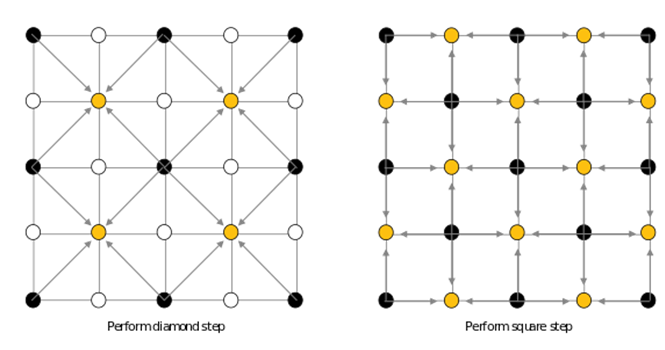
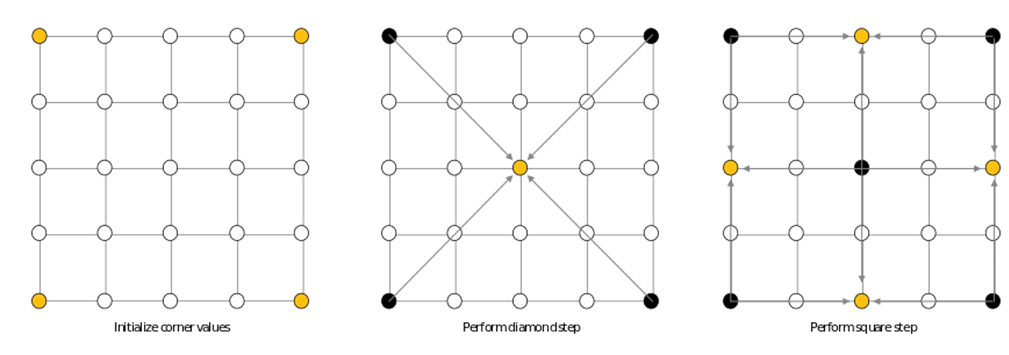
### 3.4.1 Lloyd's relaxation algorithm

This algorithm is used to determine the smallest distance between all pairs of vertices in a directed graph without cycles of negative length. The non-cycle condition is due to the fact that in a graph with negative cycles the smallest distance between some vertices is undefined, because it depends on the number of passes in the cycle.

## 3.5 Diamond Square

It is used to generate a two-dimensional, dynamic array of real numbers containing fractal noise. The DS algorithm is an extension of the concept of pseudo-random fractal generation through recursive divisions and averaging. In the case where the goal is to generate a two dimensional texture filled with noise, the operation of the algorithm starts with drawing four starting values for the cells in the corners of the texture. Then the alternate execution begins, square step and diamond step, to give value to all cells.

The square step consists in determining the values of the cells in the centre of gravity of the squares formed by the cells that already have values. The new value is calculated by averaging the values of the four cells constituting the vertices of the square and adding a random value from a certain range. In the diamond step, the values of the cells in diamond centres are determined. In this case, this is also done by averaging neighbour cell values and adding a random variable. An important part of this process is the narrowing of the range from which a random variable is generated after each cycle. In this way, the fractal characteristics of the generated noise are obtained.



## 3.6 Biomes

One of the stages in the generation of the world is to divide it into biomes, or zones of difference the climate, fauna and flora. Biome for each field is calculated as a function of two parameters: height above sea level and temperature. Thanks to using Perlin noise as terrain generator, the field height can easily be obtained by reading and digitizing its brightness. The temperature is calculated in the same way. Using The same method generates an image of heat distribution.

## 3.7 Seed

This is a combination of pseudorandom numbers, thanks to which the map generator creates a map or other objects. Which allows players to obtain the same map/object as their friend as long as both of them use the same seed.

## 3.8 Comparison of algorithms <TO BE DONE LATER>

In order to compare algorithms if they fit procedural generation of buildings and to find which is better, faster, takes less resources etc., at first methods to build a building needs to be created, otherwise there is no point in testing algorithms on their own as they need implementation beside it.

# Chapter 4: Methodology & Design

## 4.1 Research

At first Research focused on history of procedural generation, its beginning, how it all started, where was it used and by whom. Also the direction it is going, what it can achieve and what it can’t. What different games or software use procedural generation for and what type of algorithm they use. Unfortunately most developers don’t share information about which algorithm they use for the product and why. Later research advanced to searching for advantages and disadvantages of procedural generation, and of course there are many of them. Later research on the types of algorithms that exist and that can be used for procedural generation. And which algorithm may be used for procedural building generation, as to achieve that, the best would be to create new rule based algorithm made specifically for procedural generation of building. In this way it will be optimised best for this type of job. To test which algorithm is best for this job at first rule based algorithm that will build a building basing on the provided numbers eg. Perlin Noise, Diamond Square etc. needs to be created, otherwise there is no point in testing algorithms on their own since they need implementation at first.

## 4.2 Research Question

* How do you reassure best efficiency and speed?
* Pros and Cons of Procedurally Generated Content
* Can you achieve similar level of complexity and quality by using procedural generation as with pre-made models.

## 4.3 Proposed Solution

At first algorithm for building creation that will be provided with numbers needs to be created before any of the algorithm can be used. As for creation of buildings rule based algorithm would be the best fit. Buildings can be broken down to rules, it may have several floors, each floor have rooms, every room has door and windows, if it as multiple floors, it needs stairs, rooms need to have both ceiling and floor and also to be of minimum and maximum size. Windows shouldn’t be placed between 2 rooms rather on the exterior of the building unless for some reason requested by the user. Doors need to have enough room to be opened. There are many more different rules that building will need to have defined before the implementation can be started. After breaking down the building to rules, algorithm for its creation need to be implemented, at start just random numbers can be used to not overcomplicate things and instead to focus on one task at a time. Later different random number generation algorithm can be used to test which is best for this type of job.

## 4.4 Vision Document

### 4.4.1 Purpose

Purpose of this project is to create algorithm for procedural generation of buildings, to compare different types of algorithm and to choose the best one for this kind of job.

### 4.4.2 Scope

This project will create procedurally generated building. The number of buildings generated will be up to the user, it will be possible to create single building or whole city rich in different type of buildings if necessary. The complexity of the building is to be defined by the user, the building may be from garage, studio apartment up to family houses, villa or even skyscrapers.

### 4.4.3 User Environment

This is going to be started from unity engine where user can define how many building he want to generate, what constraint the buildings to have e.g. user wants it to be 1 floor and 4 room building, or 15 floors and 40 rooms building.

# Chapter 5: Implementation

## 5.1 Prototype

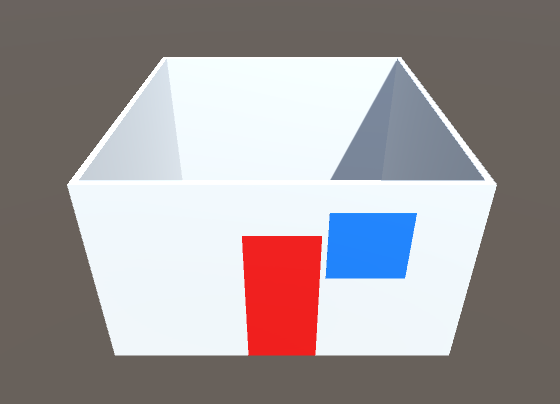
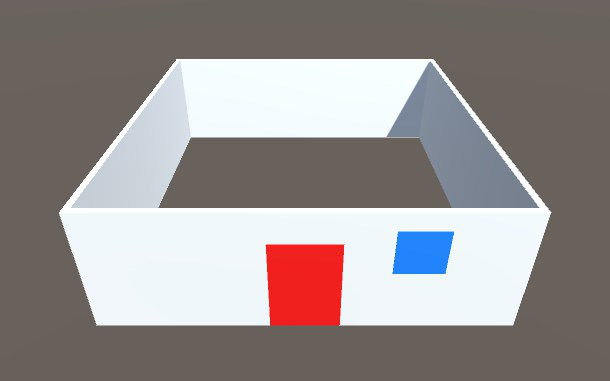
Prototype will make object e.g. walls, windows and doors by instantiating cubes and changing their material, position, rotation and scale until they resemble chosen object.

|  |  |  |
| --- | --- | --- |
| Task Num. | Details | Status |
| 1 | Function to create wall | Complete |
| 2 | Function to create door (needs to take in consideration position, rotation of the wall) | Complete |
| 3 | Function to create window (needs to take in consideration position, rotation of the wall) | Complete |
| 4 | Function to create Room (use previous functions, when placing doors and windows take in consideration possibility of overlapping of the mentioned) | In Progress |
| 5 | Function to create floor and ceiling | To be Done |

|  |  |  |
| --- | --- | --- |
| Sprint Number | Start Date | End Date |
| 1 | 2/11/2017 |  |

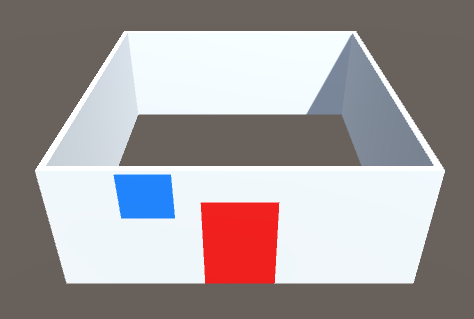
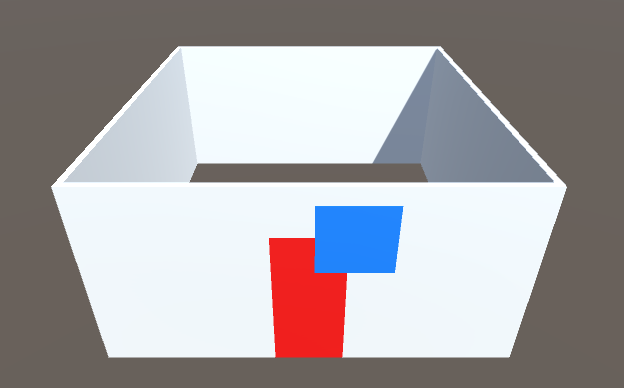
### 5.1.1 Progression

Prototype with partially implemented room function that utilises, functions to create walls, doors and windows. Currently misses check for overlapping doors and windows, and can make windows only on front wall.



2nd run

1st run



4th run

3rd run - overlapping windows and doors

# References

Amit. (2010, September 04). *Polygon map generation, part 1*. Retrieved from Blobs in Games: http://simblob.blogspot.ie/2010/09/polygon-map-generation-part-1.html

Bubriski, J. (2016, November 08). *Procedural Generation 101 (for games)*. Retrieved from Johnny Code: https://johnnycode.com/2016/11/08/procedural-generation-101-for-games/

Flafla2. (2014, August 09). *Understanding Perlin Noise*. Retrieved from Adrian's soapbox: http://flafla2.github.io/2014/08/09/perlinnoise.html

Herman. (2009, April 25). *How to Use Perlin Noise in Your Games*. Retrieved from Dev.Mag: http://devmag.org.za/2009/04/25/perlin-noise/

Jordan Mecom, J. a. (2015, December 26). Retrieved from jmecom: http://jmecom.github.io/blog/2015/diamond-square/

Lorditch, E. (2015, February 05). *The Science Behind Oscar's Award-Winning Trees And Tresses*. Retrieved from PhysicsCentral: http://www.physicscentral.com/buzz/blog/index.cfm?postid=4512988575527688739

Lowe, P. G. (2015). *Fractal Brownian Motion*. Retrieved from The Book of Shaders: https://thebookofshaders.com/13/

Mathigon. (n.d.). *Fractals*. Retrieved from Mathigon: https://mathigon.org/world/Fractals

procedural-generation. (2016, July 21). *Procedural Generation*. Retrieved from Procedural Generation: http://procedural-generation.tumblr.com/post/147756123743/biomes-in-biogeography-a-biome-is-a-set-of

*What are Fractals?* (2014, March). Retrieved from Fractal Foundation: http://fractalfoundation.org/resources/what-are-fractals/