Newcastle University

School of Computing May 2022

Undergraduate Dissertation Games Engineering

Reducing The Time and Costs of Game Development with Procedural City Generation

Tiger Kato (190353780)

Word Count: 15590

Supervisor - Dr G, Ushaw

School of Computing, Newcastle University

Abstract

In the last decade alone, the gaming industry has seen huge advancements in terms of graphics, content, and usability, as well as the development of new generation consoles, virtual reality, and mobile gaming. This change has put a lot of pressure on game companies to create the highest quality games in a short period of time. As they must try and meet the growing expectations we see from players today.

For a new game to stand a chance against its competitors, the company must keep the time and costs of creating the game low, while maintaining the quality high. They must also find ways to keep players engaged for a prolonged period of time, due to the high volume of popular games available. Games are now being made with the intent of adding new features throughout the course of its popularity, adding to its cost of production.

In this dissertation, I intend on finding ways to reduce the time and costs of generating a realistic city through the use of procedural generation. I will look at the extent to which one can take a minimalistic approach when it comes to designing an environment which must also feel fresh and exciting for the players.

Declaration

"I declare that this dissertation represents my own work except where otherwise stated"

Acknowledgements

I would like to thank my project supervisor Dr Gary Ushaw for providing me with exceptional support throughout this dissertation period, along with Dr Richard Davison and Dr Giacamo Bergami, who joined us in our weekly progress meetings to guide me in the right direction. I want to also thank all the other staff at the School of Computing department for the progress I have made over the 3 years, to allow me to take part in this project with the skillset I have gained.

Table of Contents

Introduction 9
1.1 Context 9
1.2 Problem and Rationale 10
1.3 Aim and Objectives 10
1.4 Dissertation Structure 11
1.5 Changes 12
Research 14
2.1 Procedural Techniques 14
2.1.1 L-Systems 15
2.1.2 Voronoi Diagram 15
2.1.3 Perlin Noise 17
2.1.4 Summary 17
2.2 Reducing the Costs of Development 18
2.2.1 Development Costs 18
2.2.2 Summary 19
2.3 Geometry in Cities 19
2.4 In the Real World 22
2.4.1 In Game 22
2.4.2 Architecture 23
2.4.3 Design 24
2.4.4 Summary 24
2.5 Evaluation Techniques 25
2.5.1 Evaluation 25
2.5.2 Level of Detail 26
Design and Implementation 28
3.1 Introduction 28
3.2 Planning 28
3.3 System Requirements 29
3.4 Tools and Technologies 30
3.4.1 Unity 30
3.4.2 Visual Studio 30
3.4.3 C# 30
3.4.4 Blender 31
3.5 System Architecture 31
3.6 Development 32

3.6.1 Introduction 32
3.6.2 Procedural Buildings 32
3.6.3 Procedural Skyscrapers 34
3.6.4 Roads 35
3.6.5 Regions 37
3.6.5.1 Vegetation 37
3.6.5.2 Initial Changes of the River 38
3.6.5.3 Bezier Curve 39
3.6.5.4 Residential 39
3.6.5.5 City-centre 40
3.7 Generation 40
3.8 Summary 41
Results and Evaluation 42
4.1 Overview 42
4.2 Time and Costs 42
4.3 Aesthetics 44
4.3.1 Vegetation 44
4.3.2 Houses 45
4.3.3 Procedural Buildings 46
4.4 System Testing 49
4.4.1 FPS 49
4.4.2 GPU 52
4.5 Overall Evaluation 53
4.5.1 Objectives 53
4.5.2 Plans 53
4.5.3 Engineering Approach 53
Conclusion 55
5.1 Overview 55
5.2 Aim and Objectives 55
5.3 Personal Development 57
5.4 In the Future 57
References 59
Glossary 61
Appendix 62
Testing 62
Analysis 64

Table of Contents

Figure 1: Procedurally generated city in The Sinking City 9
Figure 2: Tree designs using L-systems 15
Figure 3: Turtle System [8] 15
Figure 4: A simple Voronoi diagram with 40 points 16
Figure 5: A map of the US using coloured Voronoi cells 16
Figure 6: Perlin noise used to determine heights of buildings in a city by @mrdoob 17
Figure 7: Building plan with different sections 18
Figure 8: Skyscraper – an example of rotation 19
Figure 9: Pyramid shaped building – an example of scaling 19
Figure 10: Arc de Triomphe, Paris, France. Designed by Jean-François-Thérèse Chalgrin 20
Figure 11: Dubai, UAE 21
Figure 12: An image from a gameplay of GTA-V 22
Figure 13: A block of apartments in White City, on the outskirts of London's city-centre 23
Figure 14: Picture of London's city-centre taken from the river Thames 23
Figure 15: Cobbled streets in Rome, Italy 24
Figure 16: A street in Kabukicho, Japan 24
Figure 17: Scene captured from Battlefield 1 27
Figure 18: Agile development model 28
Figure 19: High-level system architecture 31
Figure 20: 3D model of a door inside a wall, designed on Blender 32
Figure 21: Procedurally built apartments using different textures on walls 33
Figure 22: 2D design of apartments with different sized floors 34
Figure 23: Procedurally generated skyscrapers 35
Figure 24: 3 sets of rules used for the L-system algorithm 36
Figure 25: Procedurally generated roads using L-systems 37
Figure 26: Procedurally generated vegetation 38
Figure 27: Bezier curve using 4 control points 39
Figure 28: An image from the completed city 41
Figure 29: Gameplay image from The Forest (2014) 44
Figure 30: A street of houses in America 45
Figure 31: A street of houses from the procedural city generator 46
Figure 32: Skyscrapers from the city generator 47
Figure 33: Skyscrapers in Shanghai 47
Figure 34: A block of apartments in Cardiff, Wales 48
Figure 35: Apartment blocks generated in the city 48
Figure 36: Average FPS when generating parts of the city (population 5) 49
Figure 37: Average FPS when generating parts of the city (population 1) 50
Figure 38: Average FPS when generating parts of the city (population 10) 51
Figure 39: Average FPS with LODs (population 5) 52
Figure 40: GPU usage after generating the city 52

Table of Tables

Table 1: Device specification 42

Chapter 1

Introduction

1.1 Context

Procedural Content Generation or PCG is the algorithmic creation of game content with limited or indirect user input. The creation of such content follows a set of rules and designer inputs to generate random, but structured content on its own [1].
A procedural city generator holds the same concept but concentrates most on the generation of a city, rather than the terrain, or specific objects.

PCG has become very popular amongst games such as Minecraft and Terraria for their procedural terrain generation. Where players can explore an unseen, exciting world every time a new map is created. This gives players a purpose to keep playing the game and a sense of excitement to find new things.
In terms of procedural city generation however, there are very little published games for players to enjoy, and even less so in a 3D environment. One game named 'The sinking city' uses a 3D procedural city generation approach in their open world investigation game. The design of this city allowed the developers to build a 2 by 2-kilometre urban city map in a very short space of time. The team at Frogwares claim that their program can create almost the entire map just by drawing out the city grid [2].

Figure 1: Procedurally generated city in The Sinking City

1.2 Problem and Rationale

Over the last decade alone, the gaming industry have has seen a huge increase in volume and seen substantial changes in the graphics and mechanics. “The Video Gaming Industry is now estimated to be worth $178.73 Billion in 2021” [3], which is just over 2.5 times its value back in 2012. This shows just how much the demand for new, high-end games haven risen in the last few years.

Creating these new games however, entails a significantly larger cost of production, which must be met by a popularity of the game for the company to survive in the competitive markets we have today. Thus, these games must have high-performing graphics as well as enjoyable new content, while maintaining the price of the game itself relatively low, to stand a chance against its competitors.

Currently, the cost of developing a Tripe A game can cost a company anything from $80million and $150million on average” [4] which roughly take an average of 2-3 years to produce. This is a huge growing problem in the gaming industry, where changes in demand for content are very frequent. Also, with such a variety of games available, keeping the players engaged for a long period of time is becoming harder every year, making it less sustainable to create these games.

To add to this, another problem that developers must face is the observation that games are predominantly hits driven; a study in the UK revealed that the top 99 titles (only 3.3% of development) account for 55% of all sales [5]. This means that even with top quality graphics, new game content, and frequent updates, these games have to either be part of a well-known game company or are extremely lucky and become a hit amongst players.

Games such as ‘Cyberpunk 2077’ took *CD Projekt RED* 8 years to develop but could hardly compete with the market in terms of content and graphics. In this case, the hard work and determination poured into this project was not necessarily reciprocated by the fans who were quick to revert to other games. Players are also required to have a minimum of 70GB of storage space in order to play this game which is where the next problem arise. Games trying to meet its high expectations can often be met with higher system requirements. If we are not able to reduce the size of the game, players could struggle meeting the required capacity or even face problems with their hardware.

This project focuses on creating a procedural city generator which creates a realistic, enjoyable environment for players to explore. The buildings and architecture throughout the city will draw out natural and urban aesthetics to the scene using inspirations from different cities around the world.

To overcome the problem of increasing time and costs on developing an environment, the project will also look to reduce the number of objects used as building materials. Notably, this will help build large games in a shorter period of time, as well as reduce the content size to help lower the minimum system requirements.

1.3 Aim and Objectives

The aim of this project is to generate a realistic procedural city and identify how much one can reduce the time and costs of development while maintaining a good level of design.

After conducting research on different methods of reducing the time and costs of developing a game, I decided on three key concepts to use for my work. These include:

- Planning ahead to reduce waste and make thorough research on related/existing projects.
- Reuse objects when possible. The walls used on some buildings could be replicated but with a use of different colours and textures, it reduces repetitiveness in the city.
- Focus on adding detail to those areas where users pay more attention to and reduce/remove detail on areas where they are less likely to see.

The next part of the aim I wanted to focus on is to find the best way to maintain a good level of design while reducing costs. This will be challenging of course. The two elements are complete opposites and would mean sacrificing one to develop the other. Cutting too much time and costs would result in a repetitive, low-quality game but pouring too much of my time into the design will only increase the costs of development. However, the aim here is to find a balance of the two.

I have created a set of objectives to ensure I cover every aspect of my aim in detail.

1) **Analyse three different techniques used for randomising the map of the city, where roads and buildings are placed in a structured way throughout.**
2) **Discuss and identify the most effective way to reduce the costs when developing a large map**
3) **Explore different geometry types within buildings and road structures and identify what makes a city look realistic.**
4) **Develop an algorithm that will generate different areas of a city, including a city-centre, with taller and more unique buildings.**
5) **Evaluate the time taken to create unique buildings that are only generated a few times as well as the time taken to create buildings that are repeated throughout the city.**
6) **Evaluate and compare the differences in three cities which use a different number of objects or building materials.**

1.4 Dissertation Structure

- *Introduction*

The first chapter of this dissertation explains one of the current problems developers are facing in the gaming industry, together with my personal take on this problem as the motivation. To ensure my results are relevant and useful in this field of topic, I have set out clear aims and objectives which will be followed thoroughly during the course of this project.

- *Research*

Chapter two then looks into the background of procedural generation through existing projects, articles, and primary research. The topics covered are:

- Procedural techniques
- Reducing the time and costs of development
- Geometries in cities
- In the real world
- Evaluation Techniques

- *Design and Implementation*

Chapter 3 is about what was done in the project and why those decisions were made. It describes the steps I took throughout the course of this project to meet my aims and objectives.

- Planning
- System Requirements
- Tools and Technologies
- Software Architecture
- Development

- *Testing and Evaluation*

On the fourth chapter, a detailed evaluation of the project is made through testing and analysing key features against the proposed objectives.

- Time and costs
- Aesthetics
- System Tests
- Overall Evaluation

- *Conclusion*

The final chapter is a conclusion based on how well I have achieved the project's aim and objectives, together with any future plans I have for improvements.

- Aim and objectives
- Personal Development
- In the Future

1.5 Changes

Since the project proposal was finalised at the start of March, there have been a few changes made, some of which I had not initially factored.

At first I wanted to design lots of different buildings manually and compare the time and aesthetics to a procedurally built one. It is still my aim to design the procedural buildings myself, but I will not have enough time to make individually designed buildings as well. Instead, I will use existing projects and factors from real life when evaluating my creations.

Another idea was to design a wide range of objects to be used as the building blocks, and the number of these objects used could be changed when comparing the size of different cities. However, I realised that instead of changing the number of objects used, changing the number of buildings generated would be a much better idea. This way, the variations of buildings will not be reduced, but the size of the city can be altered.

Chapter 2

Research

2.1 Procedural Techniques

The term 'procedural' means relating to an established or official way of doing something. In relation to PCG, this means that it must follow a set of rules or algorithms established through scripts, in order to generate content procedurally.

George, K. and Hugh, M. explains that procedural generation of any sort must describe the entity to be generated through a sequence of instructions, rather than as a static block of data [6]. The instructions can then be used to generate entities, and the algorithms within it should allow the generation of instances to be varied in characteristics. These characteristics should be randomised, but only to an extent. They are strictly determined by the instructions and therefore should not see any anomalies where the entities do not follow a structured pattern.

In the paper written by Ebert et al. [7], the authors narrowed down 3 of the most important features of procedural techniques that allows for an effective and most structured procedural generation:

- *Abstraction*
- *Parametric control*
- *Flexibility*

Abstraction: The first and most important feature states that we must abstract the complex details of a scene or sequence and write them into a mathematical function or a set of procedures. This allows us to call and manipulate the data when and where needed to. This will be achieved using scripts and abstracting the desired goals into an algorithm.

Parametric control: Through the use of abstraction, we are able to assign parameters to the instructions, such as the minimum and maximum number of trees to generate within a specific radius. These parameters can also be set to change procedurally to create an even more diverse output at run time, such as a procedurally generated building with random width and height.

Flexibility: The entities one can capture, and produce is limitless. There are no bounding constraints we have in the real world, and therefore we are not necessarily limited to generating the exact same model abstracted but can alter the output to our desire. This will of course allow me to create a city of any size, to help me achieve some objectives.

In this project, I will not only be looking to procedurally generate a city, but most of the building and infrastructure within the city will also be procedurally built. This way, I can explore more complex and unique architectures that differ with every new map. I will also look into procedural geometry when generating buildings, to add some shape and variety into the city, and closing the gap to reality, discussed in objective 3 and 4.

2.1.1 L-Systems

L-systems were first introduced in 1968 by Astrid Lindenmayer who at the time, used it for simulating the development of multicellular organisms. However since then, computer scientists have found many other uses for L-systems, to procedurally generate entities in a realistic manner. For example, creating branches on a tree can be done through an L-system algorithm. It allows the designer to create naturally curved branches without manually designing each one. The designer also has the ability to randomise the output of the algorithm, where the resulting shape and sizes have limitless possibilities, reducing repeated objects within a scene (see figure 3).

I noticed that this would be a great way to generate the roads in the city I create, to allow me to produce realistic road patterns where roads stem off one another. The type of L-system I will be implementing is called the *turtle system*. It takes in a set of random, or explicitly specified values, to draw out a map where the road entities will be placed. To visualise this, use figure 2 as an example. The values used are 'F' for a movement forward, '+' for a positive rotation of 45 degrees, '-' for a negative rotation of 45 degrees and '[]' is used to save and load positions.

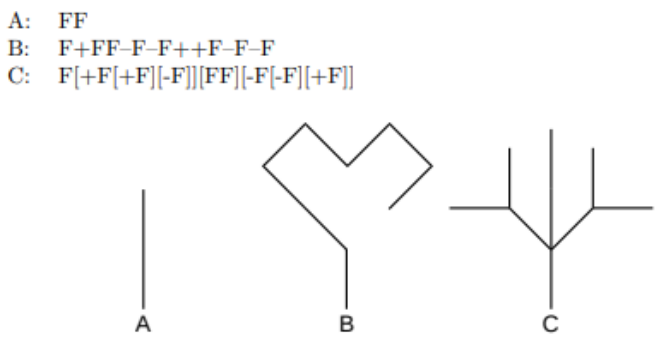


Figure 3: Turtle System [8]

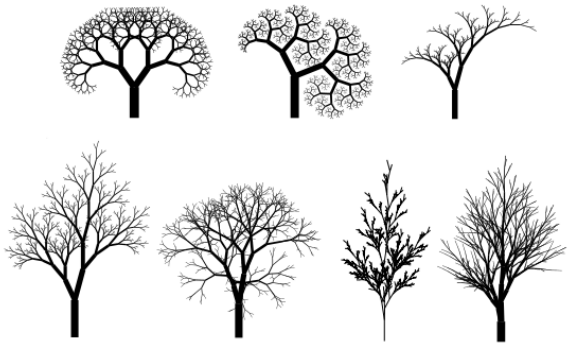


Figure 3: Tree designs using L-systems

In an article by Martin Jormedal [8], the author discusses the effectiveness of generating road maps using L-systems, and the developer's capabilities to go beyond the more common methods of generating roads in a realistic way. He describes many forms of L-systems which are individually suited for different tasks. However, I will be using the turtle system described above which is the most suitable for this project and the given timeframe while providing me with the opportunity to achieve objective 1 and 4. This is because a complex road structure is not one of the main focuses of this project. I must concentrate on meeting all objectives first.

2.1.2 Voronoi Diagram

Voronoi diagrams were first introduced in 1644 by René Descartes but is more commonly known by the work of Georgy Voronoi, a mathematician who studied the n-dimensional case in 1908.

“A Voronoi diagram is created firstly by randomly placing points on a Euclidean plane. The plane is then divided up into tessellating polygons around each point, creating *cells.* The shape and size of these cells are determined by the regions which are closer to its centre point than any other.

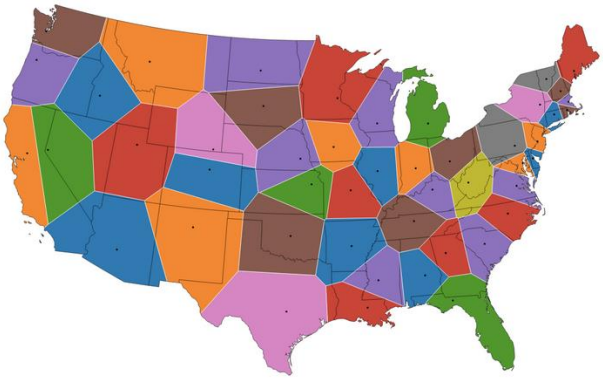


Figure 5: A map of the US using coloured Voronoi cells

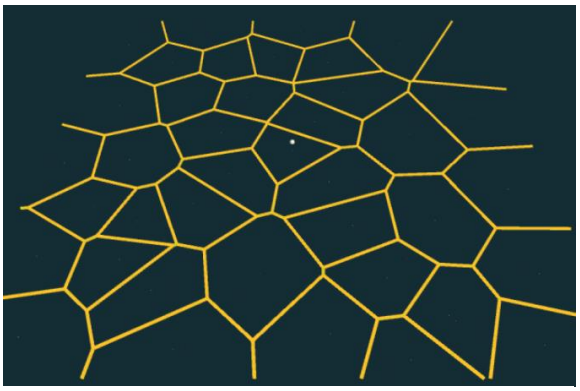


Figure 4: A simple Voronoi diagram with 40 points

As seen in figure 4, each cell inside a Voronoi diagram can be coloured in, to separate itself from other cells. I could use this to my advantage to create different regions within the map. For example, the 2 cells nearest to the centre of the terrain could be given the same colour. This will be the allocated city centre, where buildings and infrastructures such as shops, landmarks and motorways will be generated. The cells nearby could have different colours which represent rivers, parks, apartments and so on. It is also important to notice that the number of cells and regions are completely adjustable to suit the programmers needs.

Also seen in figure 4, multiple cells can have the same colour regions. In which case, users may see some repeated patterns within the map but could save the developers the time and costs of designing brand new areas from scratch.

I also noticed that the lines diving each cell could be recognised as roads or motorways that circle the entire map. The size and shape of each cell will differ every time a new diagram is created and will also depend on the number of cells involved. Therefore, implementing this idea in a procedural city could be very effective as the base of every map starts with the road structure. I will however, be careful when using this method to generate the city, due to the fact that one city does not require a large number of cells. Author Yakin Najahi realised that with too many cells, comes a large number of intersections and is an irregularity to see so many streets surrounding one cell [9]. My solution to this problem is to create significantly less cells (5 to 10) in which case there will be less streets surrounding each cell, as well as having a good number of regions to explore the city.

2.1.3 Perlin Noise

The development of Perlin noise by Ken Perlin was to introduce natural looking textures through computer programming by defecting textures with fractal-like detail as *noise*. Such textures can be especially useful for generating heightmaps for a given terrain but can also be used to determine the heights of buildings around a city. Figure 6 here is a good example of how the heights of buildings can be designed using Perlin noise.

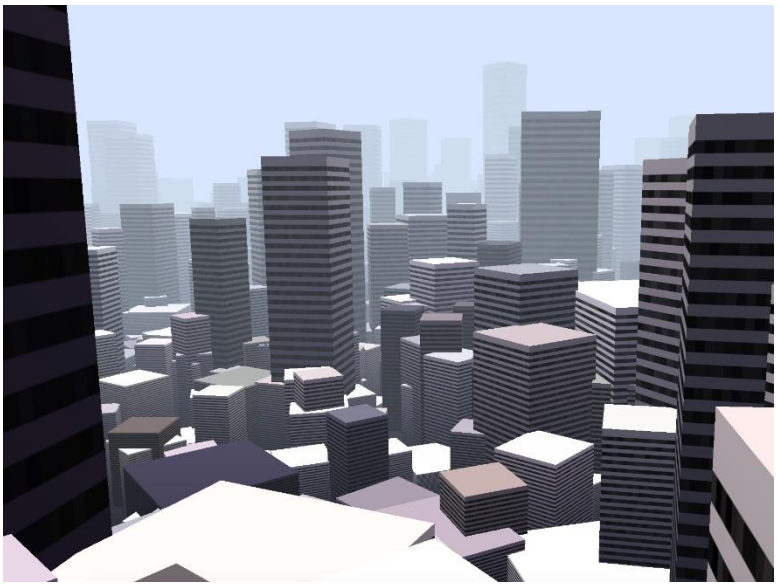


Figure 6: Perlin noise used to determine heights of buildings in a city by @mrdoob

Using Perlin noise this way is capable of drawing out realistic features of a city. When looking at most cities from above, you can see patterns of tall buildings in a cluster, as well as small buildings/houses in clusters around it. These patterns are scattered in and out of the city and can be seen all around the world. This is partly to do with the purpose of those buildings; whether that's residential or commercial, it often creates a small circle of said building types in the same area.

An article written by Frank, E. and Olsson, N. [10] explains their use of Perlin noise when generating a city in detail. They were able to randomly determine the heightmaps of the generated buildings as well as create districts, similar to the clusters mentioned above. This could be a great way to meet objective 4 to create different areas of a city. It may also be useful to evaluate different sized cities, mentioned in objective 6, as the maximum heights can be predetermined this way.

2.1.4 Summary

In chapter 2.1, 3 different techniques have been studied to meet the first objective which are L-systems, Voronoi diagrams and Perlin noise. Each one affects the generation of the city in a different way but will be crucial in the development stage to generate random but realistic cities.

If no changes are made, these three techniques will be fully justified later in this project, in chapter 3. The L-system algorithm will be implemented to generate roads on the map, the Voronoi diagram will be used to split the terrain into multiple cells, and Perlin noise will be used to determine heightmaps of buildings and the terrain.

2.2 Reducing the Costs of Development

For the second objective, I looked into the different ways one can reduce costs when developing a game. Arguably the most effective way for this in my circumstance is to use less objects for the creation of buildings and reusing the same prefabs in multiple areas. This reduces the need to design different buildings from scratch and also requires less storage space.

2.2.1 Development Costs

There are many ways a team of developers could potentially reduce the time and costs of creating a game, from developing the game overseas for cheaper labour, to creating code that is able to run on all platforms, removing the need to reprogram parts of the game.

The area this project will be focused on is reducing and reusing prefabs wherever feasibly possible. I must however, ensure that a good level of design remains, while reducing the costs. To meet Objective 2 to a high standard.

One speaker from Enodo Games discusses his way of generating huge maps without creating problems on the hard drive [11]. While discussing the challenges they faced generating an entire map of Paris, he talks about the overwhelming number of architecture designs throughout the city. To implement them into a game, he decided to create a program that would procedurally generate buildings with numerous different layers. For example, the building blocks used on the first floor could be completely different to the selection of building blocks used on other floors. Another way is to split a building into sections and using a different walls/structure per section. Use figure 7 as an example. The building blocks on the bottom floor are different to the ones above and includes a space for an entrance. The blocks horizontally up from the entrance have a smaller window than those beside it. This is an interesting concept to introduce into this project. If done well, the architectural designs throughout the city will be very realistic and should see minimal repetition.

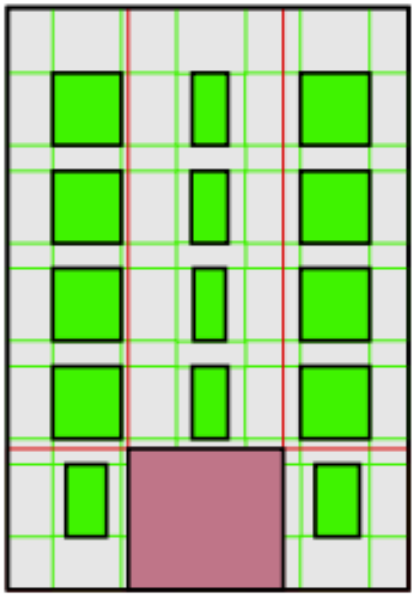


Figure 7: Building plan with different sections

He then talks about the ability to extrude in or out certain parts of a building as well as rotating and scaling building blocks to create entirely new architectural designs. Looking at the figures 8 and 9, you will notice that the two buildings share a similar rectangular building block. These buildings could be recreated procedurally, through the same method of scaling and rotating the objects used. This led me to think of ways I could implement his ideas into my own project to create realistic procedural buildings while saving time and costs on creating them one by one. Exploring the scale and rotation of these buildings could also be used to meet the third objective of exploring geometry within objects.

Instead of using multiple different building blocks for one building, I could further reduce the costs by changing the colour or texture of the blocks instead. This way, buildings will have a range of colours and patterns throughout the city and therefore create a more natural look to the environment, as well as reducing some of the impact on the memory for storing the objects.

Figure 9: Pyramid shaped building – an example of scaling

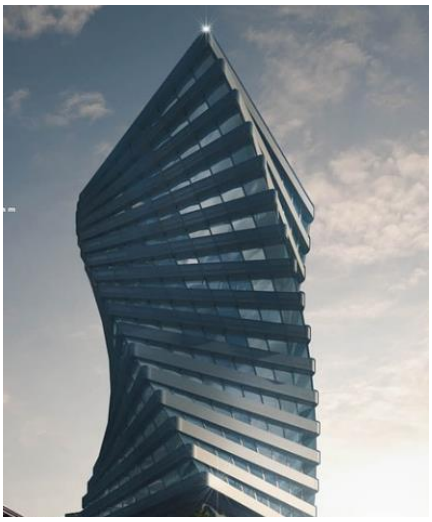


Figure 8: Skyscraper – an example of rotation

2.2.2 Summary

2.2 details the method I will be taking to reduce the size and costs of the project. In chapter 3, I will present this approach using images from the city I generate. It has given me an insight to the engineering approach I will be taking to develop this project.

2.3 Geometry in Cities

For thousands of years humans and geometry have been very closely related. Some find the beauty in natural shapes so fascinating; they try recreating it by hand. Geometry can be found in almost anything around the world, including every-day objects. The mathematics behind these shapes have been studied heavily over time, to help us deepen our understanding of geometry and its significance to our society.

Within cities, geometry is often found in road structures and architectures, but can also be found in the shape of the city itself. “Planned towns display a geometry of straight lines and smooth curves, built on a directness of movement which can only be imposed from above, embodying some sense of man's direct control over nature through technology” [12]. The author here describes the nature of a man-made design of a city. It is interesting to put further thought into this with the design of this project. It should take into consideration both straight and smooth curves throughout the designs of all objects if a natural environment is to be achieved.

Figure 10: Arc de Triomphe, Paris, France. Designed by Jean-François-Thérèse Chalgrin

The images above show a clear and well-designed structure used to build the city of Paris. Although this may not be so clear to see from below, the beauty of such craft can be spotted all around the world.

As well as the overall city structure, architectural designs have seen patterns of geometry from many years ago. “Wherever planned development has taken place, man has invoked the doctrine of visual order and imposed simple, regular geometrical forms or shapes on cities using the geometry of Euclid and drawing inspiration from such city building as far back as the Greeks” [12]. The author here mentions Euclidean geometry, which is a mathematical system used to study geometries on planes and solid figures. The mathematical element here is important in programming, which will allow me to abstract the concept into an algorithm; described in chapter 2.1.

Although I do not plan on creating buildings based on geometries, I think it is an important aspect of a city to shorten the gap from reality. It also provides a structured environment if it was to be used in a game. Generating buildings and roads in completely random places would deteriorate the aesthetics as well as create a confusing environment to play on.

One of my primary focus for this project is to create a realistic city. Figure 11 shows a great example to further develop my understanding of what an ideal city would look like inside a game. There is a high concentration of tower blocks and skyscrapers in the centre of the city, with interesting road patterns in and around the city centre. There is also a river flowing through, as with most other big cities, which will certainly be a good feature in the environment I create.

Figure 11: Dubai, UAE

Apoorva Ajmera discusses the effectiveness of structured geometry within a city in detail. "Geometry, in combining forms and space in to a single essence, not only facilitates purpose but communicates meaningful organization of elements can be achieved with conscious use of geometrical ordering principles" [13]. That meaningful organisation is certainly something I should consider when it comes to the development stage. With objective 1 stating that the city must be in a structured manner, and objective 3 assesses the realistic features.

The different styles in the architecture should also be noted here, where some are much more unique than others. In chapter 3, I should always consider the shape and size of each build, and how it should be randomised to an extent. However, I should also see some form of ordering in terms of volume and height where possible to increase realistic designs.

2.4 In the Real World

2.4.1 In Game

Rockstar Games is the creator of some of the most popular games in the last decade, with Grand Theft Auto V, and Red Dead Redemption 2 in a very high position of the best-selling game of all time according to HP [14]. A small part of GTA V's success could be down the fact that the game is based on a real location. Creating a game based on real-life events or places can be advantageous depending on the target audience, and in this case, being able to control your character freely around an open world game, allowed players to explore their imaginations in a highly realistic environment.

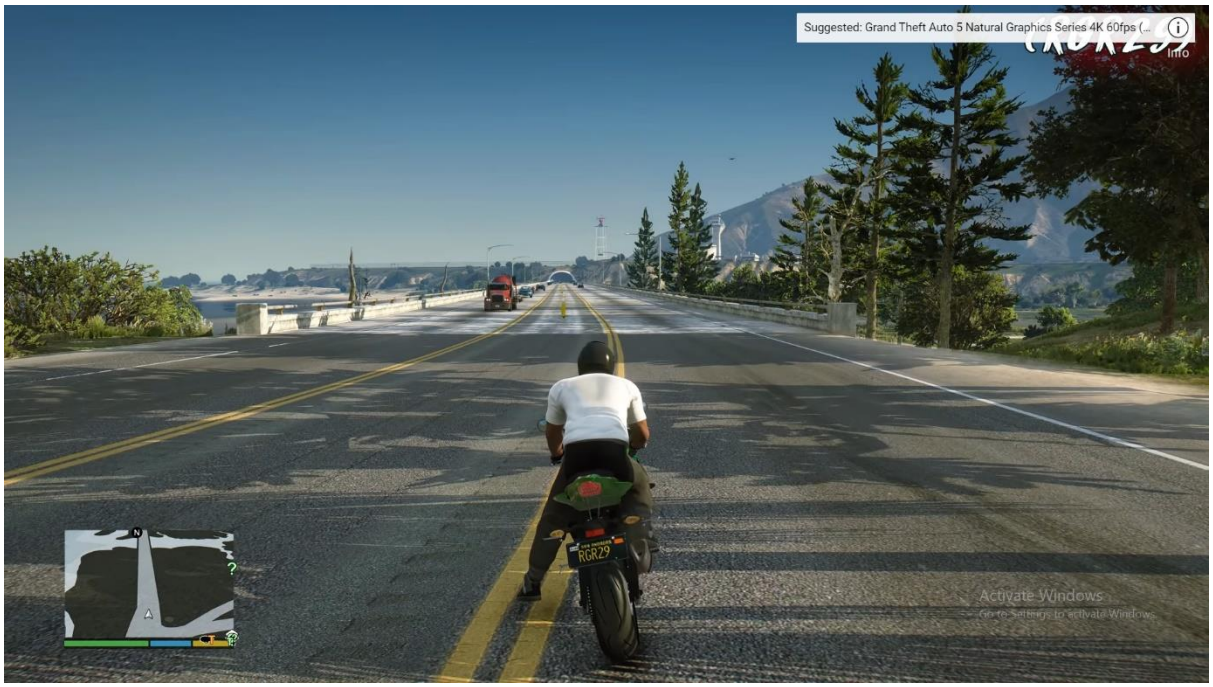


Figure 12: An image from a gameplay of GTA-V

I could use a similar approach when designing the base of the city I want to generate. Certain features or characteristics from cities around the world could be used to model some of my designs. In which case, will the city be guaranteed to feel more realistic? Not necessarily. It all boils down to the content itself and how well it is generated.

A study by Thomas Lechner et al. on agent based procedural city generator showed that a completely randomised city map can look realistic and generate cities that share similar features as seme from the real world [15]. This article proved to me that a well-designed algorithm, comes hand-in-hand with a well-designed city. In the city I generate, I will of course study the basis of cities in real-life. However, I want the results to be randomised rather than copying the structure of a real city, which also helps to achieve the first objective.

2.4.2 Architecture

To ensure the generated scenes look realistic, I must understand some basic features of the target content before designing it on Unity. To start with, I had a look at the layout of a typical city around the world. Yakin Najahi wrote “when investigated closely, these cities do not always look realistic. For example, skyscrapers should be more concentrated in downtown area and a downtown is more likely to be located near a river or a lake. Also every city needs some common locations and buildings like a main square or plaza, police stations and fire departments in every neighborhood, a hospital in an easy access location, roads should be made on flat surfaces if the choice is possible, etc.” [16]

Although these features may not seem very important from a distance, when players are faced with constant unrealistic designs in a game, it becomes easy to resort to other alternatives. He also makes a great point about police stations and hospitals being part of the city which most maps are lacking.

To achieve objective 4 to generate different areas of a city, I wanted to look into the types of buildings I should use. I decided to take some inspiration from the buildings typically located in the city-centre, as well as the residential areas around it.

Figure 14: Picture of London’s city-centre taken from the river Thames

Figure 13: A block of apartments in White City, on the outskirts of London’s city-centre

Using an example of London in both the figures above, there are clear distinctions between the two areas of the city. Figure 13 is full of skyscrapers, unique buildings, and very little wildlife, whereas the area in figure 14 have similar-looking apartments clumped together with nothing but a small greenery in between the buildings. Note that as Najahi mentioned earlier, we see the river located in the downtown area of London. These features make it easier for a player to distinguish the difference between the different areas within a game and offers distinctive aesthetics to each region of the map.

2.4.3 Design

Finally, to fully encapsulate the beauty of PCG, I am considering the architectural designs used in different cities from around the world. Some cities capture their historical heritage and traditions from hundreds, if not thousands of years ago. With computers, the different designs of each city can be pre-determined by the programmer. Therefore, with every new map, a different style of city could be determined, which appreciates its own designs and patterns.

To do so, I studied some streets which share little to no similarities in tradition:

Figure 15: Cobbled streets in Rome, Italy

Figure 16: A street in Kabukicho, Japan

The two streets from the above figures have very little in common – from the buildings, vegetation and street signs, each place has their own unique identity, and this is most often repeated throughout the rest of the country. To have the ability to pull out these features into a procedurally generated city would be very interesting and would certainly offer an exciting playing field.

2.4.4 Summary

This part of the chapter analysed areas in the real world that can be used as a model in the city that I produce. It does not necessarily have to copy every detail but should be used as a model to refer to with the design in chapter 3.

2.5 Evaluation Techniques

To achieve objectives 5 and 6, I must study the different ways to evaluate a PCG, as well as the techniques used to generate the results to be analysed in chapter 4.

2.5.1 Evaluation

The final and most important part of this dissertation is the evaluation stage in chapter 4, where I will reflect on the work and analyse how well it meets the objectives from chapter 1.

However, evaluations can often be biased and without evaluating the most key elements, it will not be effective nor useful for anyone reviewing this paper.

The evaluation of game content can be very challenging. We do not always know the designer's intents, styles, and goals, or the user's views on the scene with such a large variety of players and their play styles. One way of evaluating the city is to use *map sketches*. In a paper by Antonios, L. et al. describes the effectiveness of creating quick sketches of a generating map and using mathematical algorithms to automatically determine a good level [17]. However, a sketch of a huge city with hundreds of buildings will be extremely hard to compare and with lots of generated levels, there may be little to no differences in the quick sketches.

An article published by Shaker et al. [18] discusses the difficulty in evaluating PCG as the generator could be capable of producing thousands of unique levels. The first difficulty is that with so many levels, it could take the developers too long to evaluate one by one. Secondly, it becomes more and more difficult to point distinctive comparisons across all levels. It becomes harder to judge what is acceptable and what isn't. And to know what ratio is acceptable between good levels and bad levels. If amongst 10 levels, you find 2 that are exceptional, would you modify the generator or keep it as it is?

They then go on to talk about evaluating the 'expressive range' of the generator. Which refers to the scores given to each of the produced levels, based on leniency and linearity. Linearity measures the ability to produce levels that range in shape and size, and leniency evaluates the toughness of each level by combining the difficulties and ease the players will face. By using these 2 outputs as metrics of a graph, evaluating large numbers of levels become very easy, and shows anonymities in a much orderly way.

Although I will not be evaluating the city I produce using those specific metrics, it has provided me with a better understanding of an evaluation technique I can certainly use in this project. The evaluation of different cities I generate could compare subjects on a slightly larger scale such as:

- The framerate when generating cities with different numbers of buildings, different types of buildings, and different size of the city
- The processing time of generating a level
- LOD and the impact on the hardware
- The repetitiveness of buildings

2.5.2 Level of Detail

One of the main aspects I will be evaluating is the impact on the hardware mentioned above. To be able to change the output for the sake of comparison, I decided to use Unity's built-in level of detail feature, which can modify an object's mesh with regards to the distance from the camera. Doing so can ease the pressure on your hardware without reducing the quality of the object.

In an article by James H. Clark [19] which describes the procedures and techniques involved with improving the level of detail, the author mentions different ways to improve the graphics or the performance of the algorithms that generate them. It introduced a new way of designing procedural objects using meshes that would activate accordingly to the camera.

In a different paper, Jabłonski and Martyn talks about the two fundamental algorithms within LOD management [20]:

- *LOD evaluation* – determines when to change the objects LOD. Depending on the observer's distance from the object, its LOD must change accordingly. If players are close enough to see the object, but the LOD is too low, it will damage the aesthetics of the scene.
- *LOD transition* – deals with how the detail actually changes and the specific mesh to be used in every level of transition.

These two points must be carefully evaluated when changing the LOD of the generated objects in the city I produce. When used correctly, it could drastically improve the performance at runtime as well as the ensuring the FPS remains at a good level while travelling through the map.

It must be noted using LODs on objects can have a negative effect on the visual aspects of the city. If the LOD evaluation is poorly determined, players will be able to clearly see the object in its reduced form, and therefore bringing the overall quality of the scene down.

To ensure the city looks realistic, LODs must be tested thoroughly to find the best setting for an object.

As seen in figure 17, the LOD of the area in the centre of the screen (around 10 metres from player) is set too low in relation to the placement of the camera. This should be prevented where possible, but this is all determined by the device and how much it is able to compute in each frame.

Figure 17: Scene captured from Battlefield 1

In chapter 4, I will evaluate the performance of the cities I generate, while modifying the LODs of objects. Depending on the size of the finalised city, this may be crucial to run the program smoothly.

Chapter 3

Design and Implementation

3.1 Introduction

Chapter 3 describes the process behind each step of this project and the reasons why they were taken. A section on the minimum system requirements is made to ensure the aim and objectives mentioned in chapter 1 are met to a high standard.

3.2 Planning

Once we had selected our own project paths, we were straight into the planning stage with a weekly meeting in our groups, to guarantee progress was made. This meant that not only was the planning stage crucial early on, but we also needed some visual development to our ideas which could be presented for additional feedback. Therefore, an agile development model was adopted to allow me to take all necessary steps in a shorter period of time.

The agile development model will allow me to adapt to the frequent feedback I receive from my project supervisor. If the first plan needs some changes, it is much easier with this approach to go back, make the changes and implement it into the development stage as shown in figure 18. With a traditional waterfall model, all steps are taken one by one, leaving little room for adjustments further along the line.

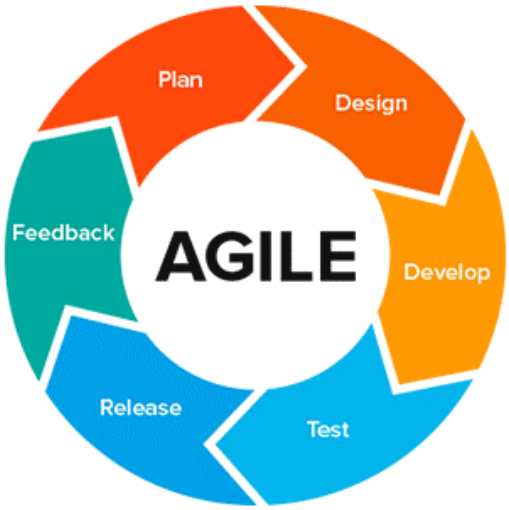


Figure 18: Agile development model

To ensure I gathered sufficient information through the background research stage, I created a list of system requirements which the project had to follow stated in [3.3 System Requirements]. Doing so

allowed me to conduct effective research and made sure that the design and development stages would run smoothly.

After carrying out the research in chapter 2, it became apparent that one of the techniques showcased would not be required for this project. Not necessarily because they are not useful, but so I have more time to use and develop the research that aligns more closely with the system requirements.

Perlin noise will not be needed for either the heightmaps of the terrain or the buildings. This is because I realised that big cities are often flat and have very little texture on the terrain. It would also create problems when generating the city itself, where roads and buildings would then need adjustments to fit the terrain. I knew this would waste precious time, by side-tracking from the system requirements. In terms of building heightmaps, I realised that clusters of similar sized buildings could just be generated through some extra lines of code in the algorithm. Again, this would greatly reduce the time I spend on developing parts of the project which are not vital for meeting my objectives.

3.3 System Requirements

I have created a list below, of the minimum system requirements which are essential to meet all objectives for this project.

1. **City Generation**

   - Area must be divided using a Voronoi diagram, using at least 5 different regions

   - Generation of the city must be randomised as to where objects are placed

   - It must be easy for a designer to generate and modify a city

   - Every city must include a river, house, apartments, skyscrapers, and shops

2. **Building Generation**

   - Buildings must be procedurally generated

   - Designers must be able to generate buildings to specific inputs of size

   - Buildings must not be repetitive aesthetically including the shape and size

3. **Size**

   - Objects must have LOD adjustments in place

   - Designer must be able to specify the size of the city to be generated (population or terrain)

3.4 Tools and Technologies

Creating large-scale projects often entails a wider selection of tools necessary to achieve specific objectives. In this project, I will be using Unity, Visual Studio, and blender; described below.

3.4.1 Unity

Unity is the game engine I decided to use to build my project on [21]. The main reason I picked Unity over any other game engine is because I have previous experience with it. I have not yet had an opportunity to explore the key differences myself in similar game engines such as Unreal [22]. However, through some background research I found that unreal uses C++ as their programming language, different to the C# used on Unity which I am more familiar with. C# is considered more suitable for game development and is faster than C++. There is also a significant difference in the amount of assets available on Unity compared with Unreal. Unreal has around 10000 assets while Unity has 31000 assets available to download on the asset store [23]. Although I will be designing some of the building materials on blender myself, I know that it will be too time consuming to design every single object from scratch. Therefore, having such a wide variety of assets available to use to my advantage can give me more time to spend on developing the objects I am designing myself.

Unity also has downloadable packages which could be helpful in the progression of a PCG. These packages are free of use and can be downloaded at any time – such as ProBuilder. Perfect for modelling 3D objects and detail editing.

On the other hand, Unreal engine supports AAA quality graphics which Unity does not. This would have been ideal in this project to further improve the realistic features I want to achieve.

3.4.2 Visual Studio

Visual Studio is the programming IDE I used to build my project with [24].

The main reason is that I can use the C# language. C# is a statically typed language, so the code is checked before it is turned into an application. Visual studio is a perfect tool to support this, with syntax highlighting, keyboard shortcuts, and code completion, to allow programmers to code much faster than without an IDE.

To add to the previous point, Visual Studio offers programmers to debug their code with ease. Allowing them to go through their source code and inspect variables without the need of compiling and running their whole project with every test.

3.4.3 C#

C# is a general purpose, multi-paradigm programming language, based on C and object-oriented programming (OOP).

Since Unity is one of the most popular game engines, there are plenty of other developers using Unity and C#, and therefore have a huge supportive community behind them. It should also be noted that C# is very easy to learn and can be forgiving for starting developers. It is important that

such high-level language is easy to understand, so programmers including myself can look for help and work out how to implement it in their own project.

To add to the greatness of C#, it has a built-in garbage collector. A garbage collector is a memory manager that can free memory and unused objects automatically. This is crucial for maintaining the application performance as well as removing the need for me to write the additional code myself.

3.4.4 Blender

Blender is an open-source 3D graphics software tool often used for 3D models, visual effects, and animation [25]. Blender is a perfect tool to design some of the 3D models that will be used as the building blocks for some of the architecture in this project, for two reasons.

The first reason is that blender can create very realistic objects in a small amount of time. Designers are capable of creating almost anything in 2D and 3D, with an easy to use user interface and helpful built-in tools to help them accomplish their goals. One of these tools is called Archimesh, which can turn a simple cube into a detailed window or door in just a few clicks. Doing so will drastically reduce the time I spend on creating the models, while keeping the designs to a high standard. Chapter 2.2 explains the need to reduce the costs of development and compared to the time it takes to create one of these objects, the design is extremely good. Making this a great tool for this project.

The second reason is that since I will be using Unity to display the created objects, I needed a software that can import and export files through Unity. Blender can Export its content to many formats including FBX, which Unity can then read with no extra work for the developer.

3.5 System Architecture

To understand system architecture I will be using, I needed some identification of its major components along with its connectivity to other components. Figure 19 shows a visual representation a high-level design. The user's inputs along with the object system produces the rendered result on the GUI.

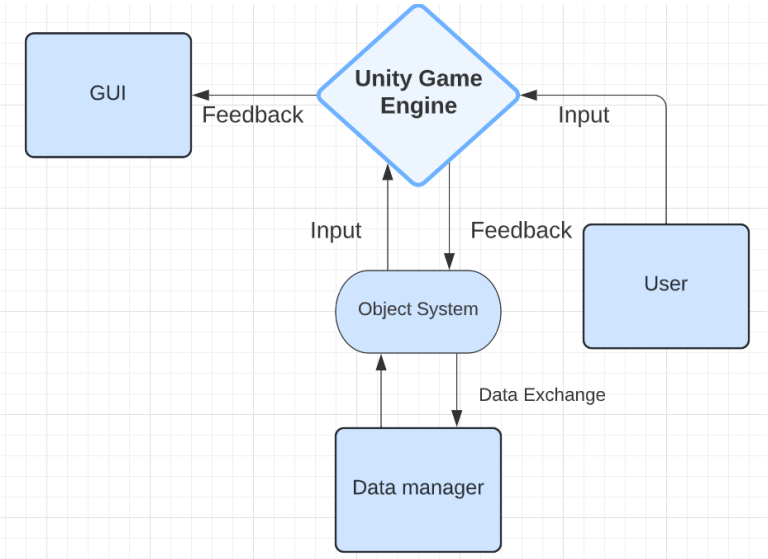


Figure 19: High-level system architecture

The data manager is capable of storing all the data handled in the project. It is fed through to the game engine for the users to explore.

The game engine compiles any necessary scripts before run time, fetches the user inputs to fill in variables stated in each script, and produces the outcome through the GUI.

3.6 Development

3.6.1 Introduction

This stage covers the progress I made with the overall design of the project from start to finish. It will cover the first four objectives and leave the two evaluation objectives for the next chapter.

The first thing I wanted to tackle was creating procedural buildings. Although I knew that a procedural city generator could create tons of interesting levels, I was stuck on the fact that these cities are generated using a set number of prefabs as buildings. This meant that although the placement of these buildings may differ in every map, the buildings themselves were always the same. In my head, the whole idea of procedurally generating a map is for the players to enjoy a completely new scene they have not yet explored – giving me the idea of procedurally generating buildings as well as the map itself.

3.6.2 Procedural Buildings

I needed a starting point for generating these buildings. I spent some time on researching different ways to generate buildings to determine the best approach and came to a conclusion of starting with apartment blocks. They seemed very versatile. A standard rectangular building would simply look like a block of apartments, but with a few tweaks, they could also be used as shops and factories. I knew this would be important using the research in 2.2 and 2.4 [pages 18, 22]. This meant I could reduce the cost of development as well as create a wide range of buildings seen in real life.

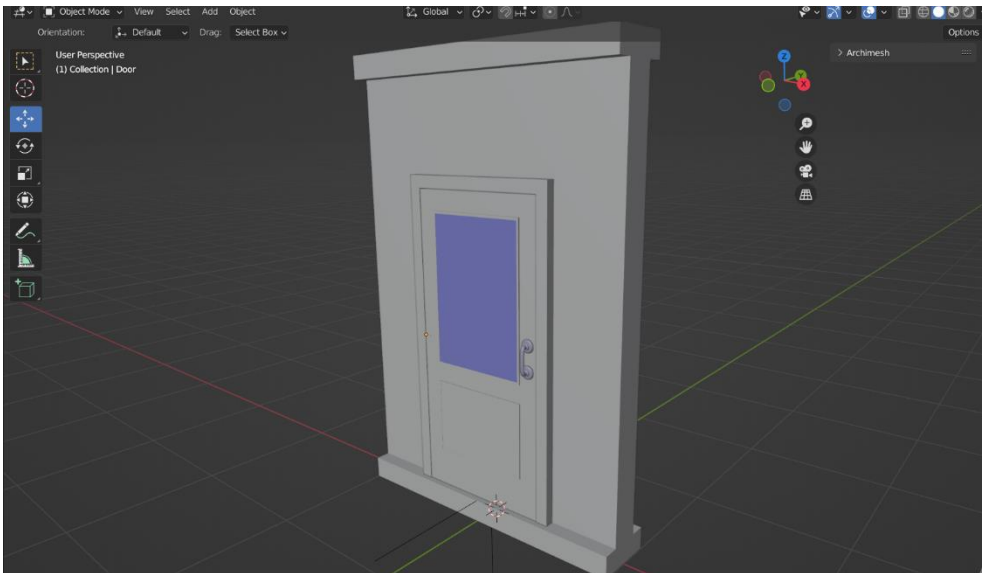


Figure 20: 3D model of a door inside a wall, designed on Blender

Before I begin with the scripting, I needed some basic building blocks such as walls, windows, and doors. I went into blender and created very simple but detailed objects using the built in tool Archimesh, described in part 3.4. Shown in figure 20.

Once I had sufficient wall pieces, it was time to start creating the script. For the building to meet the system requirements, I had to create a script that allowed the users to input variables to decide the structure of each build. These variables would describe the building to the script before it compiles it and renders it onto the scene. I quickly realised that to allow user inputs, all building blocks had to be the same in height and length, to ensure there were no irregularities in the walls generated.

The first prototype was very bland. All walls used the default grey colour as its material, each build was flat and rectangular, and had no character to it. I noticed that there was a lot of features missing, and that it needed a lot of work to make it come to life.

To do so, I went back into blender, created some more interesting walls, downloaded some texture packs, and tested these out with the apartments. These turned out much more vibrant than the first build, and now had walls using materials like bricks, which you often see in real life. I gathered some more materials to be used for the textures on the roof and floor objects, as well as the pillars on the sides of a building. This is further progress towards another system requirement, to not produce repetitive buildings, as seen in figure 21.

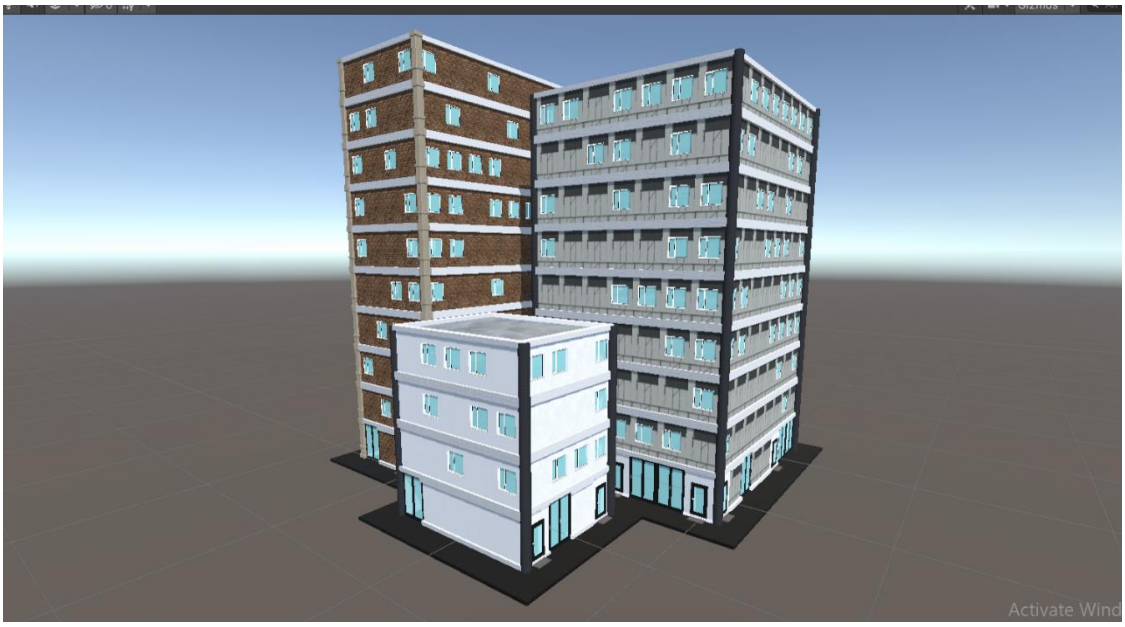


Figure 21: Procedurally built apartments using different textures on walls

To make these builds even less repetitive, I decided to allow users to decide how often a wall material changes using chance. A low percentage here would mean most walls have a random chance of changing its texture from a set list, creating some peculiar and unseen buildings. This may become more distant from a realistic build but has an opportunity to be used in an area like the slums, where there are often no symmetries in the building's textures and colours.

This will also dramatically help achieve the main aim of this project; to reduce the costs of development. The building blocks can now be reused numerously by just changing the outer design.

The exterior differences these buildings offer, will hopefully be enough to provide a wide variety of buildings for a large map.

Now that the buildings have colour, just one thing remain to make them more realistic. Adding the small detail, described in the designs in [2.4.3 Design]. I downloaded some assets that could be used diversely, to test out the overall designs. These included billboards, chimneys, balconies, and window shades. These features are set to randomly generate on each build, which for a small added detail, made a big difference.

Since the buildings so far only generate simple rectangular boxes, I decided to let the algorithm randomly modify the floors on each building in a similar way to the design draw in figure 22. Although this is typically seen in factories, it created interesting shapes which could still be used as apartments and shops. It also hugely widened the variety of buildings that would generate, achieving both objectives 1 and 3 together.

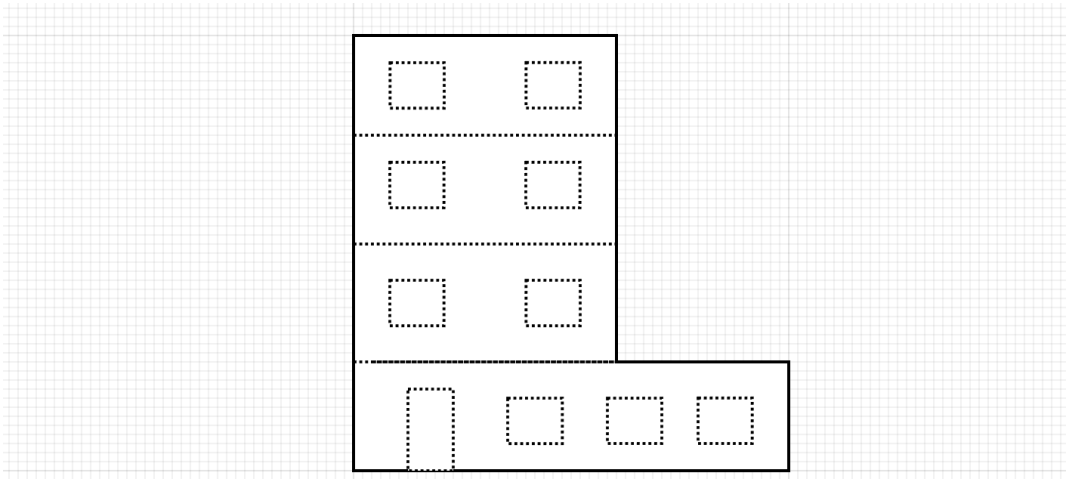


Figure 22: 2D design of apartments with different sized floors

All these features mentioned above, including the individual wall pieces can be freely modified by the user. If the designer wanted to use their own walls, textures and decorations, it is as easy as dragging it into the specified space in the inspector window of Unity.

3.6.3 Procedural Skyscrapers

To meet the system requirement of creating a city-centre with more interesting buildings, the next stage was to develop skyscrapers.

Now although I could have designed completely new building blocks for this, I decided to be minimalistic with the number of prefabs I used. This was done to meet my second objective of reducing the time and costs of development and I had to find out for myself, how different I could make this look from the previous build. I did however, create two new rooves on blender which allowed me to create triangular buildings. This idea stems from the third objective to explore different geometries, as well as following some examples from the real world.

The script for this build is structured in a very similar way to the apartments to allows users inputs and modifications to the design. I mostly used glass panes for the overall image as well as a pyramid-shaped roof. The shape and size of the roof varies with each build, some are pointy, and some are flat. Again, as seen with the apartments, different floors can be of different sizes, often seen in famous architecture and skyscrapers, and the wall pieces and their textures are randomly selected for each generation use figure 23 as an example.

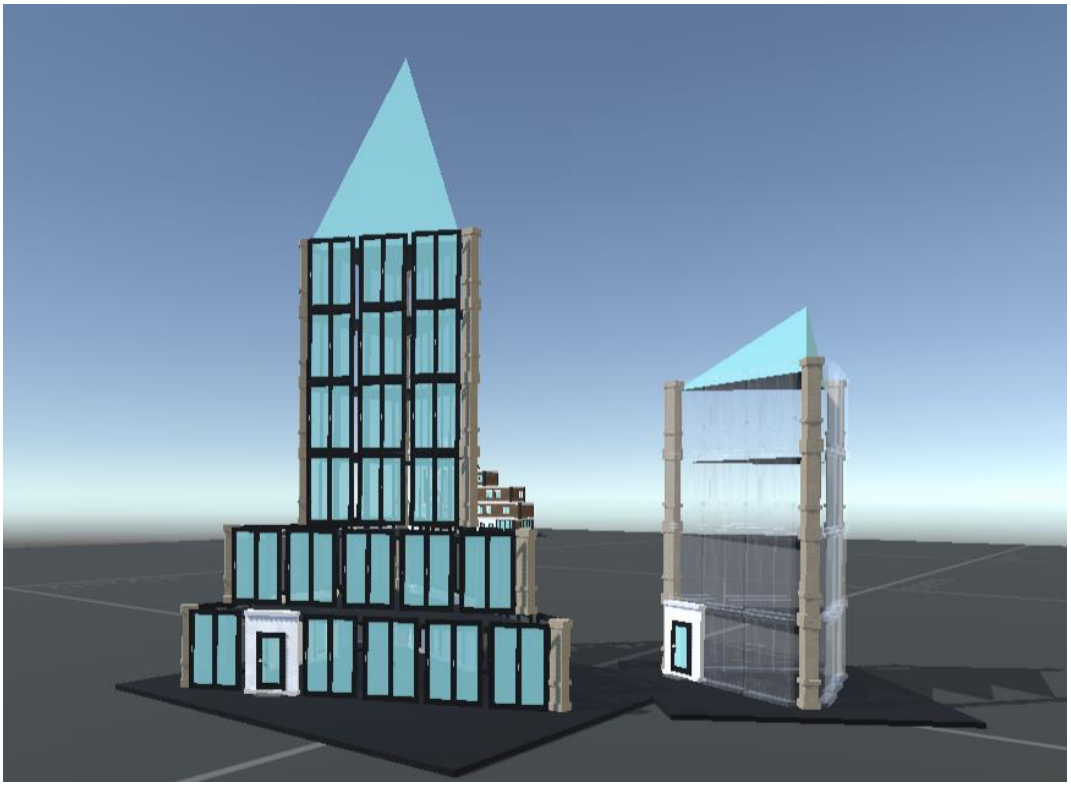


Figure 23: Procedurally generated skyscrapers

3.6.4 Roads

Now that I have made progress with the buildings, it was time to design the roads and how they would be generated throughout the city. To meet objective 1 on randomising road maps in a structured way, I decided that L-system patterns were perfect for this project (research at [2.1.1 L-Systems]). L-systems use algorithms to follow a strict rule to produce paths that stem from one another in set directions. Using this to generate the roads in the city meant that the chances of two road patterns being exactly the same were very thin.

I decided to let the program randomly select one of three rules to use when generating these roads, allowing more randomised road patterns to follow objective 1.

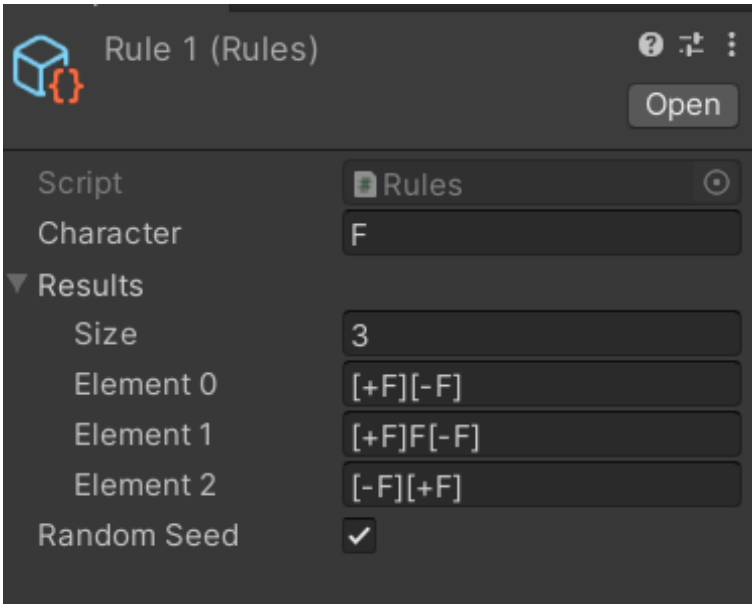


Figure 24: 3 sets of rules used for the L-system algorithm

Figure 24 is an image of the inspector window on Unity, which looks at the one of the scripts I have created. As you can see, there are currently 3 rules for the program to randomly select when generating the roads. Each rule has been thoroughly tested to make sure it produces high quality road structures with no errors.

Although the roads would generate using random rules, I had at least 4 areas that required roads. As of now, the program generates the exact same pattern for every rule. This is not ideal for this project as the outputs had to be randomised. To remove the repeated patterns, I added an ignore chance which the user can input of up to 100%. The higher the percentage, the more likely the algorithm stops in its current direction. Doing so gave me a much wider range of outputs with much more realistic results.

When I started creating the road materials using Unity's shader graphs, I stumbled upon a small problem, where I noticed that I could only create straight roads and 90 degree turns.

Although the roads in real life are not completely perpendicular to one another, I noticed that to have curved roads, I would need a whole new way of generating them. This is because the textures on the road objects would need to curve with the road and the intersection between the two curves had to be interpolated, to leave no gaps on the road. With the limited time I have on my hands, I knew this was something I had to leave for future development.

The produced outcomes of the generator can be seen on figure 25. Although this may work for this city, I would like to see more realistic roads in the future.

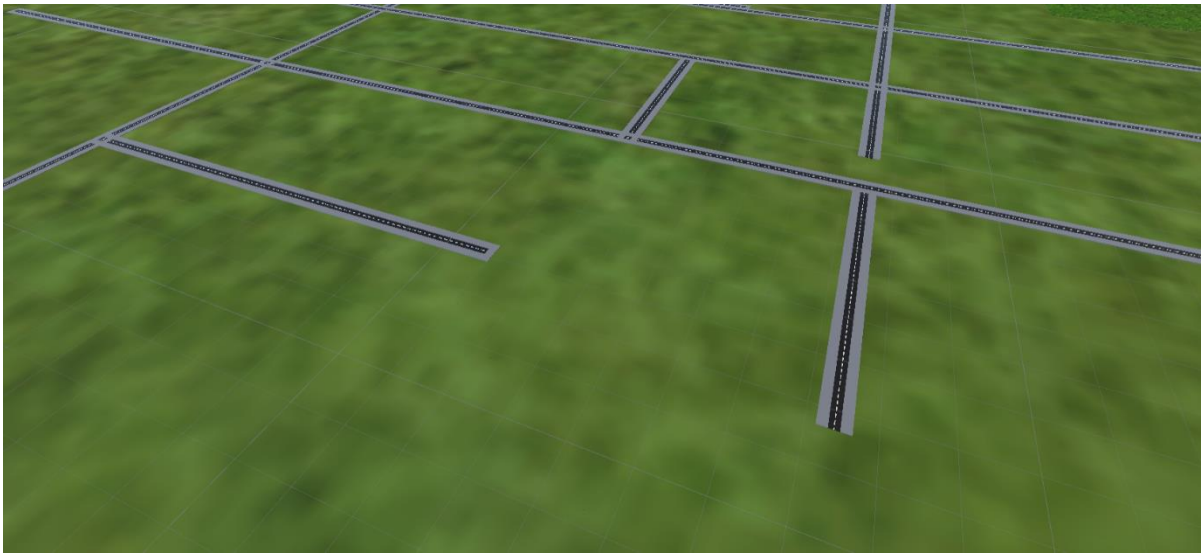


Figure 25: Procedurally generated roads using L-systems

3.6.5 Regions

For objective 4, I wanted to develop an algorithm that will generate different areas of a city which are randomised with every new map. I wanted to make use of Voronoi diagrams [2.1.2 Voronoi Diagram] to separate the terrain into 8 sections. Two for the city-centre, one for vegetation, one for a river and the other four for residential uses. The residential areas would then be separated into 4 sectors: apartments, wealthy households, urban houses, and commercial buildings. Covering more than the minimum number of regions stated in the system requirements.

I created a script that would scatter 8 points on the terrain, which becomes the centre points of each cell (*centroids)*. These points were random to a degree, but to ensure not all 8 points would generate next to one another, I had to create zones in which only one centroid could generate inside them.

To randomise the output of the city, the program randomly selects a cell and gives it a pre-determined region. This way, as well as the placement of objects, the regions themselves are picked randomly to decrease the chances of seeing the same map twice. I did however, keep the two cells in the middle of the terrain as the city-centre, to make sure the map has some structure to it. Randomising the regions completely would look messy and could face problems with the road networking.

3.6.5.1 Vegetation

The first zone I worked on is a park/forest area, which is usually seen just outside of the city-centre to give the city some form of vegetation. I downloaded some trees, foliage and rocks and created a script that would randomly generate them in the designated area.

To provide a firm structure on the overall design, I created a form of hierarchy with the objects to be placed. To put them in order, the user must select the desired density (out of 10) for each type of

object: tree, foliage, and rocks, which determines the likelihood of an object from that type to be generated.

These plants also randomly vary in size and rotation. Before each one is generated, the program randomly selects a value between – 0.75 and 1.25 to multiply the object's scale by. Adding rotation to it also explores the real nature of its form, as these plants are often tilted to a degree. This again, makes sure this area of the city looks slightly different in every map.

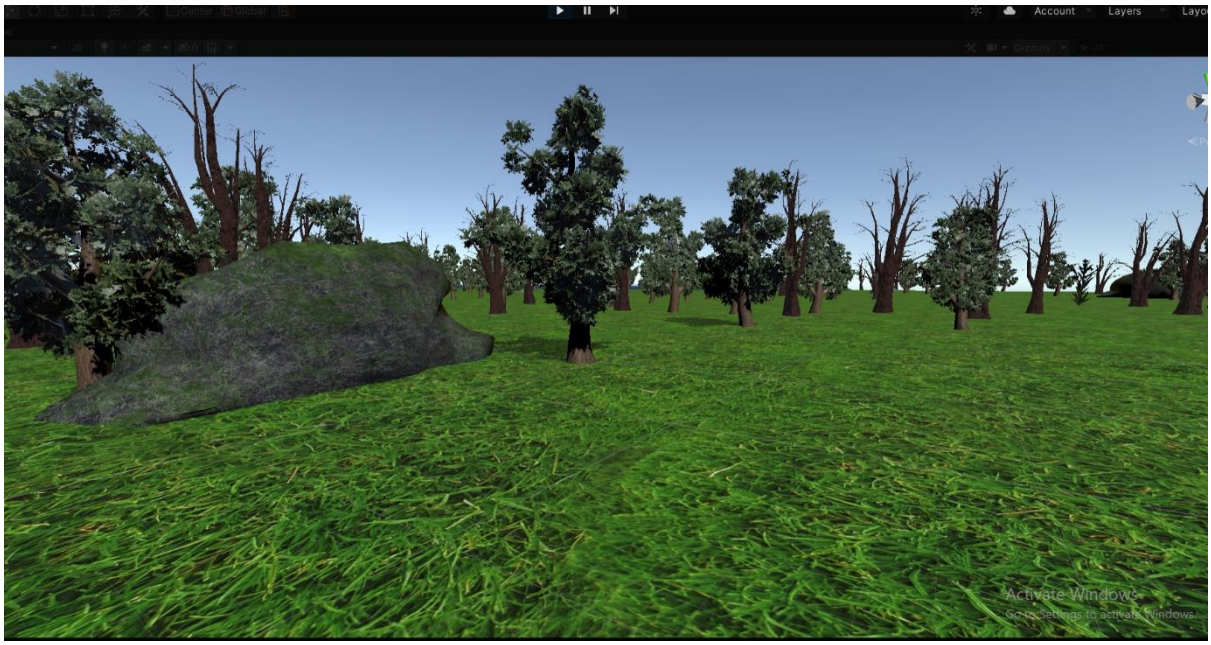


Figure 26: Procedurally generated vegetation

3.6.5.2 Initial Changes of the River

As mentioned in the system requirements and studied in the research stage, I now had to design a river that would run near the city-centre. However, I quickly recognised the difficulty when designing this, as the river would need some form of curvature throughout.

The easiest option here would be to pre-determine the river flow and its direction and create a prefab of it which can then be reused in every map. Due to the fact that the regions are selected at random, I would need a number of these prefabs so that the river points towards the centre. However, the aim is to reduce the time and costs, designing individual rivers could take hours of work and would require more space to store them.

Instead, I created a script that would generate a river from one of the selected regions and flow through to the city-centre. To add curvature, I drew a straight line from the centroid of a random cell to the city-centre. Then, using the 2 nearest centroids as the control points, I was able to draw a Bezier curve which would then be replaced by water, creating a naturally flowing river.

3.6.5.3 Bezier Curve

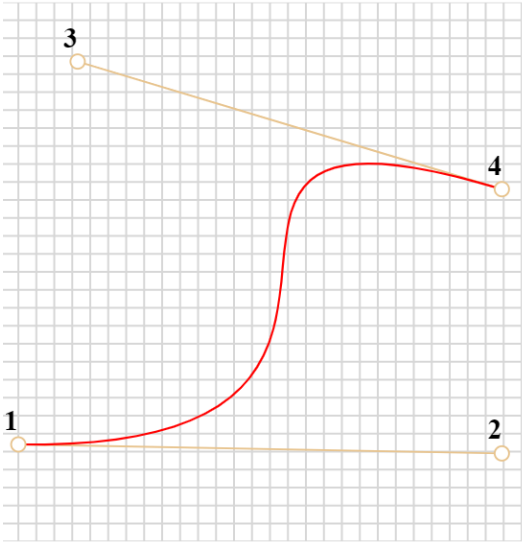


Figure 27: Bezier curve using 4 control points

In the example above, the line starts at point 1, and ends at point 4. Points 2 and 3 are control points used to determine the curve of the line. In my case, point 1 is the start of the river, point 2 and 3 are the neighbouring centroids and point 4 is the city centre.

It has created a very natural curve to the line; a realistic feature for objective 3 as well as using random control points with every map, to meet objective 1.

For the maths behind it I used:

$$P = (1-t)^3P_1 + 3(1-t)^2tP_2 + 3(1-t)t^2P_3 + t^3P$$

"Given the coordinates of control points P_i: the first control point has coordinates $P_1 = (x_1, y_1)$, the second: $P_2 = (x_2, y_2)$, and so on, the curve coordinates are described by the equation that depends on the parameter t from the segment [0,1]." [26]

3.6.5.4 Residential

After the two non-residential regions are selected, L-system roads could be generated inside the other cells. The road structure is the basis of each map, and the city will be built around it. As the terrain is fairly large (1km x 1km), I decided to only iterate through the L-system a few times, so buildings weren't being over produced. Overloading the system by generating thousands of objects at once could be too much for some user's device to handle.

Once the roads are in place, buildings can be placed beside them. To generate different sized cities as one of the requirements, users are able to dictate the size of the population. The number of buildings generated throughout the city depends on the size of the population. To separate different regions, different types of buildings are generated per region, therefore creating clusters of similar builds in the same area.

I decided to factor in some variables with the procedural buildings in different areas to add some realistic features. One region would mostly see simple apartments, consisting of rectangular builds,

some randomness to the wall textures, and include balconies and window shades. The commercial buildings are much more randomised in shape and size and will strictly only use one material throughout each build. They are also produced without balconies as they are more often seen in apartments. The buildings in the city-centre will be used as shops. The walls on the first floor of these shops are made with random materials, and any floors above will use one material.

Now to make each street come to life, I needed some small objects that would fill in the little gaps while keeping the design prices low. I decided to reuse the trees seen in the nature region as well as download some simple streetlights and signs. Trees would be randomly placed between houses, and the street decorations are placed on the pavement.

3.6.5.5 City-centre

The city-centre was slightly more difficult to design than the other regions. The main factor for this is that the river ends in the middle of one of the cells. This meant I needed a whole new road system where a bridge would cross over the river. This proved to be very challenging but with the use of Unity's built-in physics engine, I was able to detect nearby colliders; using them to avoid any clashes when objects were generating.

I downloaded a bridge and some curved roads and created a script that would generate them near the end of the river.

To add to the progress on objective 4 and using the research on the real world [2.4 In the Real World], I then procedurally generated skyscrapers, apartments, and shops around the city-centre. The shops have much more variance in shape and size compared to the apartments. Some only have 1 floor, some have multiple stories of different sizes, some tall and some small. These variations help to keep the overall city randomised but is meaningful to show realistic features and meet the system requirements.

3.7 Generation

To make the program simple to use for any designer, I created buttons which will process all necessary scripts at once. When running the program, a Voronoi diagram will automatically generate using lines that disappear after 50 seconds. In an empty object, I have added together the 3 buttons, used to create the city. The first will generate the nature, using two regions for a river and a greenery. The second button then generates roads in all other regions. The final button generates the buildings which is determined by the population density, an input from the user.

See figure 28 for an image taken in the city-centre:

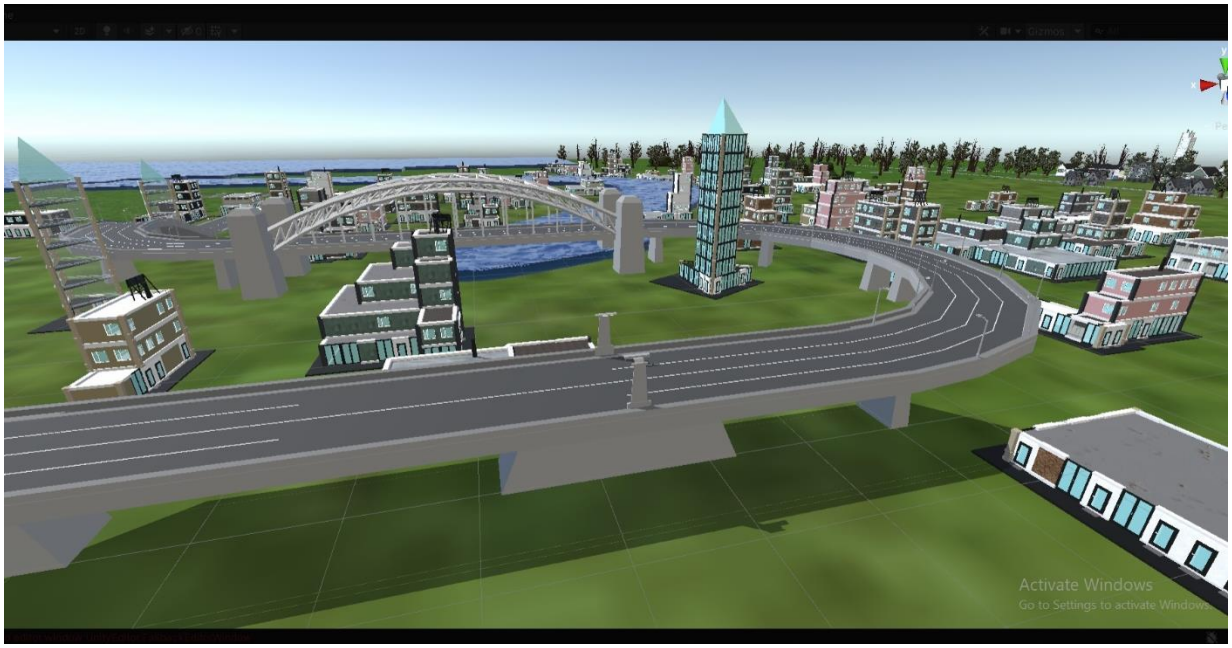


Figure 28: An image from the completed city

3.8 Summary

This chapter has briefly summarised each step of the development stage. Some of the findings can be used to evaluate the overall project in the next chapter.

It introduced a set of system requirements that is necessary to meet all objectives, as well as covering the tools and technologies that would be used throughout.

Through the course of the development stage, an explanation of each step's process is given. Along with why it was introduced, or how it meets a specific system requirement/objective. If a technique or mathematical algorithm was used in a script, a simplified description would follow, to explain the logic behind it.

Screenshots of the project were taken from the Unity scene view to provide visual representation of the points made.

Chapter 4

Results and Evaluation

4.1 Overview

This section will analyse the results taken from the finalised project and compare the findings with the aim and objectives. Where necessary, results will be displayed on graphs and figures, for a visual representation.

Extra images are displayed on the appendix, for further justification of points [11].

Any tests in this section will be undertaken using my personal desktop. Below is a table of the hardware information of the device I will be using.

Table 1: Device specification

Operating System	Windows 10
Processor	AMD Ryzen 3 3100 4-Core Processor 3.59 GHz
Memory	8GB
Hard Disk	1TB
Graphics	NVIDIA GeForce GT 1030
Form	Desktop

4.2 Time and Costs

The cost of development comes hand in hand with the time it takes to develop a game. This is because companies must pay the developers their salary, the machine costs, and its maintenance throughout the course of its development. In the case of this evaluation, I will only be focused on the time taken to design and create objects, scripts, and the generation of the city, as I cannot reduce costs elsewhere.

Reducing the costs to develop a large map is one of the main objectives for this project. Designing individual objects from scratch could take years for developers to accomplish, depending on the size, and detail of the project. My approach to this problem was to procedurally generate objects, and therefore reducing the time of development by creating a program that would design those objects itself.

The first model I decided to design was a plain wall to be used on houses and apartments. It was my first time using Blender and so getting used to its environment, and learning the basics took some time. I wanted each wall to have a slight extrusion at the top and bottom to create some basic

details, which would also be implemented on the windows and doors. Finalising these three models took roughly 2 hours.

Creating a script to generate a building is of course added to the time of development. This was also the first time I had tried to generate buildings through a script, and so I needed some time to study the field. With many failed attempts and trying different methods, the first apartment was procedurally generated after 3 days of work (about 15 hours in total).

Without a doubt, creating these scripts took much longer to put a building together than manually doing it in the scene view. However, it is the first step towards reducing the time of development. Once the script is made, I have the ability to generate hundreds of buildings in a matter of seconds.

Once the procedural buildings were complete, I moved onto designing the procedural skyscrapers. This took significantly less time as I could reuse most of the scripts used for the apartments. I also used the same building blocks I had already made for most of the structure, but with different materials. All in all, the skyscrapers took me 3 hours to create, which can now be procedurally generated infinitely.

In the finalised city, the user is able to determine the population, and therefore the number of buildings to generate. If I was to generate 100 apartments, 100 commercial buildings, and 10 skyscrapers, the program usually takes about 2 seconds to run. In these 2 seconds, the area of its placement is checked, placed beside a road or a nearby space, rotated in the right direction and procedurally generated. The chances of two exact same building being created twice is very thin. For example, out of a thousand apartments, if both width and height is randomised from 1 to 10, and a list of 10 wall textures, the chances of 2 apartments being generated the exact same is 1%. This figure does not include the fact that buildings can be generated with more than one wall material, different corner pieces, different roof material/object and not to mention there are levels to each build, which can lessen in size on any floor.

Although the apartments and skyscrapers may not have huge differences between each build, the time and costs saved from this approach is immensely large and is capable of creating tens of thousands of buildings in just one click. Some other houses and shops were downloaded from the Unity asset store to increase the variety of buildings within a city. These are free and take no time to use, as they fit right into the algorithms.

In terms of road generation, creating the shader graphs for each road type took a very long time, but again, procedurally generating them meant road networks were generated in one click of a button and meant very little time needed for putting them together manually.

Overall, this project took me three months to develop. It can generate up to a thousand buildings (can be increased by modifying a script), roads in and out of the city-centre, a forest, and a river within seconds. For some perspective, GTA-5 has around 800 buildings and had “six to eight people on the design for around nine months to a year” [27] for a model of the map to be made. Which, in hindsight, look considerately better than the city I can produce but I am content with the amount of time I managed to reduce.

Although the time of development was greatly reduced through this method, there are of course areas of improvements to be addressed. Firstly, the designs of each build were extremely limited. Granted, they change in shape and size, use random wall textures and other features but the overall structure shared similarities. For the best results, I would only generate about 200 procedural

buildings and use completely different architecture elsewhere. With a large map, a player would want to see more variations in and around the city to explore.

Secondly, a wider range of building blocks should have been introduced. Indeed, designing as few models as I could meant a reduced development period. However, adding more unique objects to the program would produce much broader range of buildings and as a result, more realistic streets. Needless to say, this will increase the time of development but possibly only by a fracture. If the algorithm is flexible enough to introduce more specific objects into the buildings, they can be used repeatedly without having to completely change the structure of the generation.

Using this finding to answer objective 5, It is definitely more time consuming to manually design the objects one by one. However, procedural generation does restrict the output to an extent, so a good balance between the two should be seen.

4.3 Aesthetics

The next part of this project is to evaluate the aesthetics of the city. The aim was to find the extent to which a developer can reduce costs while maintaining a good level of detail.

4.3.1 Vegetation

As mentioned previously, different parts of the city generate different things. I decided to spend the least time developing the river and forest areas because I thought they were the least important, in relation to the city.

I decided to compare my results with a fully developed game called *The Forest*

Figure 29: Gameplay image from The Forest (2014)

The forest is a realistic survival game with stunning graphics and creative game content. The shot taken in figure 29 shows the player looking at a mountain in the distance, with a forest on either side of them and a river in between. Although the level of detail does massively increase as the player gets closer to the object, you can depict from this image, some reused objects. The same trees and foliage here are clearly used a number of times to fill in empty spaces whilst keeping the costs low.

A similar approach was taken with the nature region in my project as previously mentioned (see figure 26). Looking at this scene from The Forest, putting more work into the nature may have been better for the user experience and the overall realness I am trying to achieve. Although forests aren't very common in or near a big city, a well-detailed park or lake, would be very good to see in the future of this project.

4.3.2 Houses

In the other regions, residential buildings are placed around nearby roads. Two types of houses are used to determine the wealthy, and less wealthy regions. Both house types were downloaded from the Unity asset store for free.

With regards to realism, the houses generated in the wealthy region are extremely detailed. The modern houses have great exterior designs as well as some interior, with the use of lights. The houses in the less wealthy regions are slightly less detailed and are not as lively. There are a total of 4 different houses per house type and 3 types of trees that randomly generate around them.

The idea is taken from a real street in America as shown in figure 30, to try and capture some realistic features.

Figure 30: A street of houses in America

Figure 31: A street of houses from the procedural city generator

The result of a wealthy street from my program (figure 31) has turned out exceptionally well and shares many similarities to the image on figure 30. In this case I would say that this part of the city is very realistic.

As for areas of improvements, I wish to add more houses into the algorithm with a wider range of colours and designs, to further venture towards realism. Although the houses in figure 30 share similar designs as each other, if I was to generate 200 houses, having 40 to 50 house designs would be ideal. This will of course, increase the costs of development drastically which had to be avoided in this project.

I also noticed a driveway and a small front lawn with every house on figure 30 is something I missed out. These features could be the key to improve the aesthetics of every street, as well as it being more than ideal if the map is used for a game.

4.3.3 Procedural Buildings

The final comparison is made on the procedurally generated buildings. Arguably being the most important aspect of this evaluation stage, as this is where the most costs were reduced. I had to make sure these buildings were also be met with a good level of design to achieve the aim.

The generated skyscrapers use glass for the walls from top to bottom as well as a glass roof. This is common for skyscrapers around the city-centre as seen from the designs in figure 32. The shapes of these builds are also more interesting than a standard rectangle, which I have tried to capture in the designs myself (see figure 33).

Figure 33: Skyscrapers in Shanghai

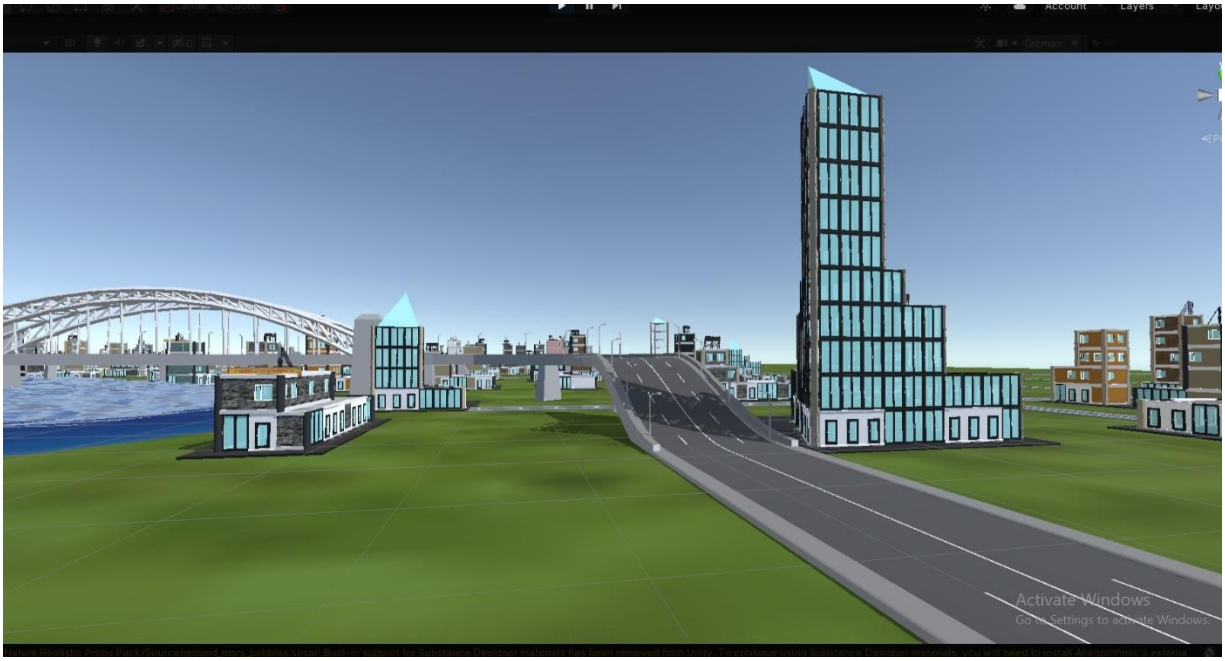


Figure 32: Skyscrapers from the city generator

In my opinion, I think they could have been improved slightly to produce much more detailed architecture, as well as adding more uniqueness to each one. If I was to further develop these skyscrapers, I would like to explore more shapes and styles, possibly adopting similar designs from figure 35, for a better interpretation of the objectives.

Lastly, I analysed the designs of the procedurally generated apartments. Once again, to create buildings as closely related to a real design, I used real apartments and the research on [2.4 In the Real World] as a reference.

Figure 34: A block of apartments in Cardiff, Wales

Figure 35: Apartment blocks generated in the city

The shape, size and the details of the apartments generated in figure 35 share some similarities with the apartments in figure 34. I have selected wall textures that are very similar to the ones used in this image as well as the windows and balconies. If only a small number of these apartments were to be generated, I would definitely say they show realistic features.

The one problem this algorithm has, is that the buildings can start to look very repetitive if too many are generated. I would like to have seen some buildings without a completely flat surface all round. For example, all the walls in the centre of a building could have been pushed back slightly and turned into balconies, also described in chapter 2 [2.3 Geometry in Cities].

I would also like to add more prefabs that could be used as the building blocks of each build. I think the more variations a player can see, the more realistic it will feel for them.

4.4 System Testing

4.4.1 FPS

When generating maps on a large scale, it is important to evaluate the system and hardware running the program. Firstly, I wanted to make sure that the FPS does not shoot uncontrollably high when generating objects. Although there is a limit on the number of buildings that can generate, spawning in a thousand buildings at the time can be a struggle for many devices.

I decided to test the FPS with each of the three generations that occur, as well as the FPS with different population size. I have created graphs of the results below. (Use appendix for visual representation [Testing]).

Each bar represents the average FPS over a 30 second period after each generation. The average is taken from the minimum and the maximum FPS over that time period.

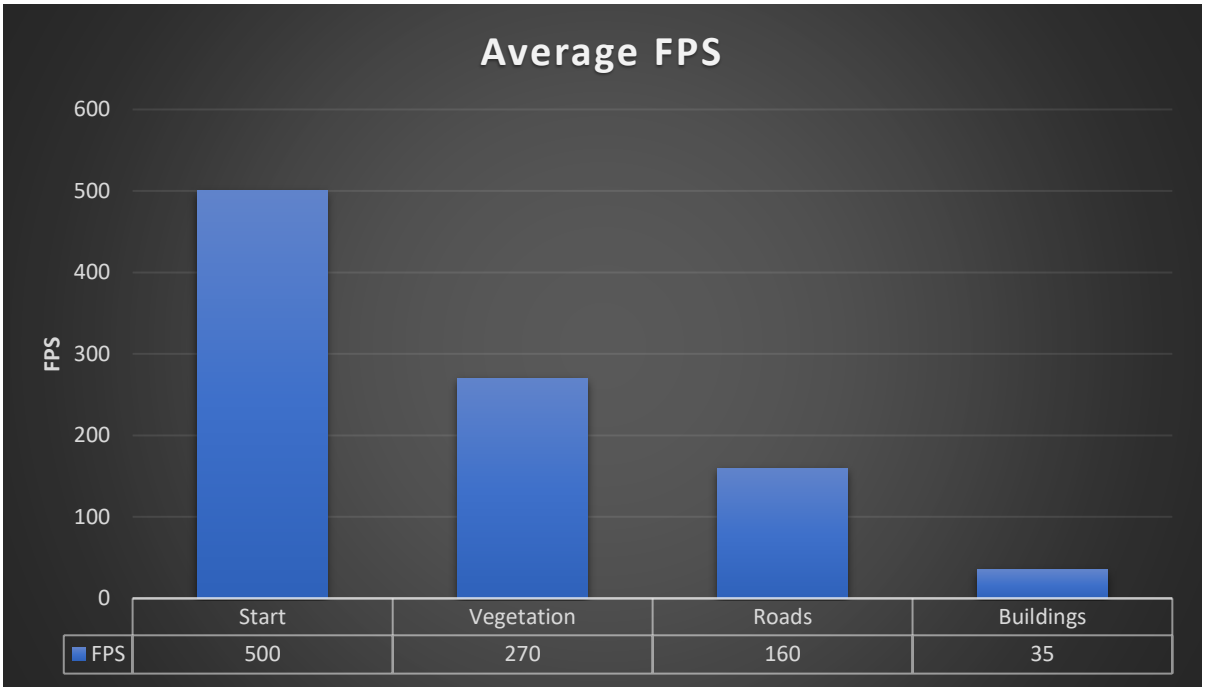


Figure 36: Average FPS when generating parts of the city (population 5)

The above chart shows the average FPS of the program after each generation with a stationary user. 'Start' means no generation, 'vegetation' is the generation of the river and nature regions, and the 'roads' and 'buildings' are self-explanatory. In this case, I used a population density of 5. This means 100 from each of the four building types are generated as well as roughly 100 buildings in the city-centre. LODs on objects are not yet used to see how my device copes

There is a clear anomaly here as the FPS drops by nearly 80% from the road generation to the buildings. This means too many buildings have been generated for a smooth gameplay, or the LOD needs implementing.

With the second test, I set the population density at 1 where roughly 100 buildings are generated in total. With an 80% reduction in buildings from the previous test, I expected a much higher FPS as a result. However, there was only an increase by 20 FPS which is a good indication to prove that the FPS rate and the impact on the hardware is not solely based on the number of buildings generated.

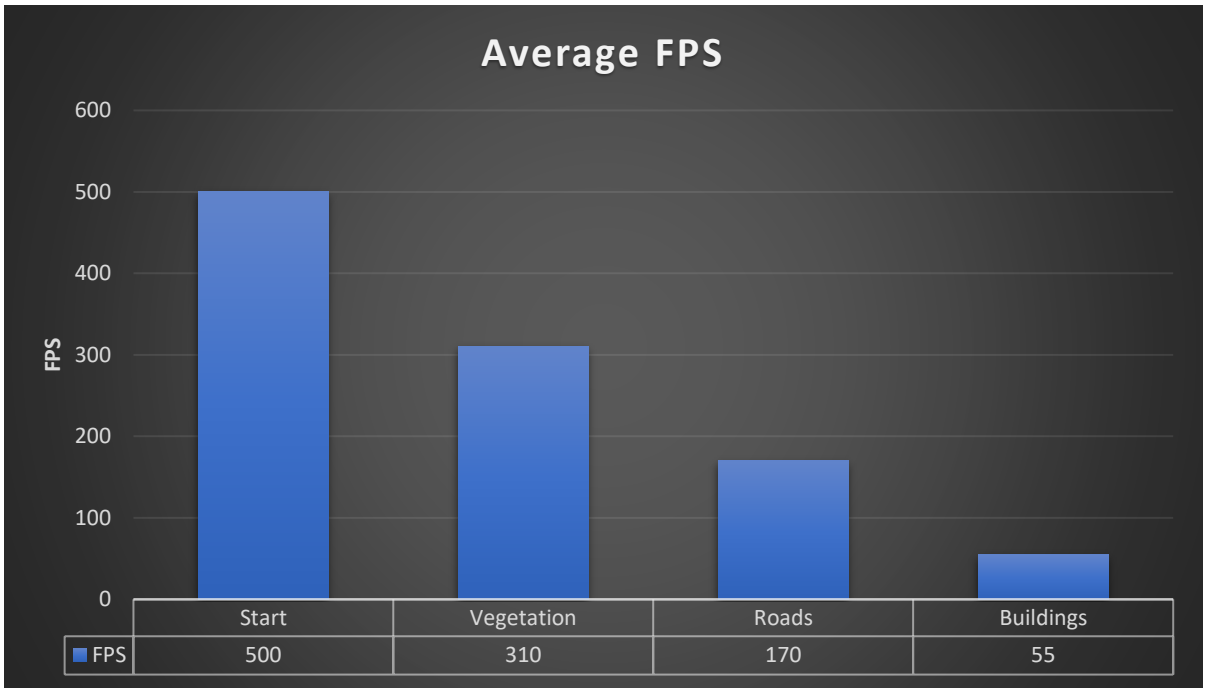


Figure 37: Average FPS when generating parts of the city (population 1)

The generated city itself has been affected though, as there are now not enough buildings to call it a city. I would say at least a population density of 4 is required in every map.

The final test on the city size was done with a population density of 10. This means that roughly 1000 buildings are generated throughout the map. As expected, the FPS drops significantly, and is too little for users to enjoy anything of this program.

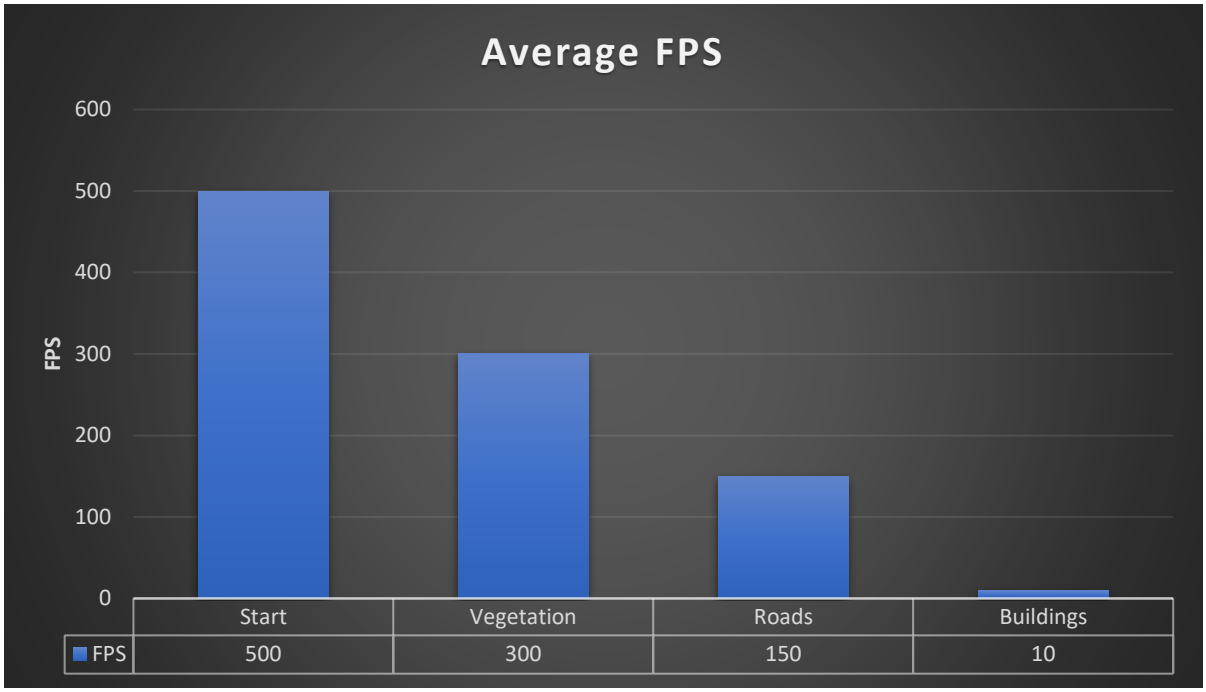


Figure 38: Average FPS when generating parts of the city (population 10)

To further improve the FPS while the program is running, I have introduced LOD levels to some of the assets that are being generated in the city. LOD dictates the amount of detail an object will display according to the distance of it from the player's view [2.5.2 Level of Detail].

There are different levels to LOD which reduces the quality of an object as the level changes. The final level of LOD is for the object to be completely culled. This means that the player is deemed to be too far to see the object and therefore the mesh can be removed from the scene. As expected, this will greatly increase the performance as the impact on the GPU and memory will massively reduce at the same time.

Below is the average FPS of the program using LODs on buildings and roads.

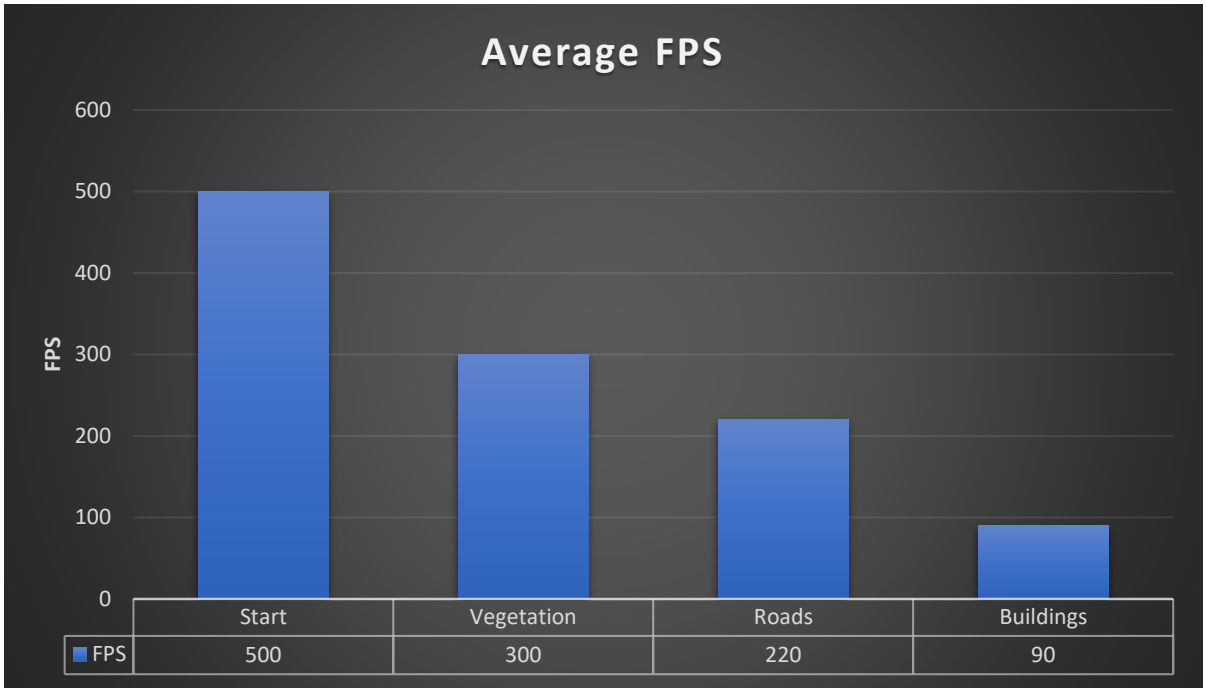


Figure 39: Average FPS with LODs (population 5)

Figure 39 is a perfect example of using LOD to improve the performance of a game. For a city with the same population size as the one used in the first test, the FPS is much higher after generating the buildings. We can see a 55 FPS increase here which is high enough to have no problems with the frame rate. There is also no need to reduce the number of objects, meaning the user can enjoy a much bigger city.

Although the algorithm generated around 500 buildings and roads in 6 regions, most of it was culled due to the distance from the camera. This means that the meshes for over half of the generated objects were not displayed, making it easier on the GPU. This proved to be the best method in terms of performance and will allow developers to use the generator to its maximum potential.

4.4.2 GPU

The next part of the project I want to test is the GPU. The GPU is one of the most important components on a pc for players wanting to play the best quality games. The FPS mentioned above is directly affected by the GPU, and so putting too much pressure on it can result in the user having to reduce the quality of the game. Figure 40 shows the impact the program has on the GPU. I generated the entire city and flew around it in the scene view.

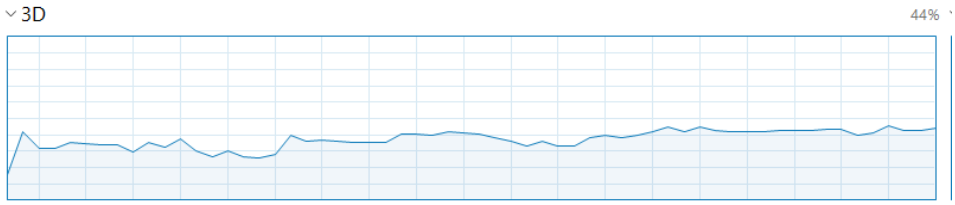


Figure 40: GPU usage after generating the city

The chart produced a 44% use of the GPU which is slightly lower than expected. A low GPU usage usually means lower FPS, which again, isn't great. However, it does mean that there is still plenty of room for the graphics card to be working at its maximum capability.

4.5 Overall Evaluation

4.5.1 Objectives

The purpose of the evaluations above is to meet the objectives 5 and 6 from chapter 1 [1.3 Aim and Objectives].

First of all, in terms of time and costs of development, this project has proved how quickly a developer can produce large cities from scratch. It has taken 3 months from the start of this project to get to the stage it is at now. I have also proven to effectively reduce costs in multiple areas which can be adapted into many other projects.

Secondly, the visual aspects were also an important part of the project. Although costs were greatly reduced, a fairly high quality city can be produced, given the time and resources available.

Finally, I discussed the performance levels of different generated cities. In the end, I decided that a population density of 4 or 5 and modifying the LODs on objects is the best way to create the city.

4.5.2 Plans

Most of my development stage went according to the initial plans and research stage. However, there were times where I would run into unexpected problems and had to find alternative methods.

The first problem I encountered was with the road generation. I did intend on having realistic roads as one of the features of this project but with the element of randomness involved, I struggled to make time for a complex design. This did mean sacrificing some of the realistic elements of the city, which I would like to see differently in future development.

The second problem was the shape and placement of the river. To create a realistic flow that was still randomly generated, I had to quickly adapt to some changes. The use of a Bezier curve was then introduced which had not been previously studied. However, the result of this could not have been any better, creating a very realistic flowing river.

The study on Perlin noise was not used in the end as I found an alternative method of implementing a similar design. After completing the city, I realised that this was a good decision as I would have spent hours implementing this into the project which would have been wasted.

4.5.3 Engineering Approach

The engineering approach of the procedural buildings allow users to be in complete control of the structure. This means designers could produce a much wider selection of buildings just by dragging their desired objects into the inspector tab in Unity.

One error I have found is that there are times were some parts of a building intersects with a road. Although I check the surrounding area by the expected size of a building, there are still errors in the generation stage which I cannot work out. This should definitely be changed in the future.

Chapter 5

Conclusion

5.1 Overview

In this chapter, I will evaluate my final project with the aim and objectives stated in chapter 1 [1.3 Aim and Objectives]. It will be assessed on how well I have done to achieve each objective and what I could have done to improve it.

5.2 Aim and Objectives

The aim of this project is to generate a realistic procedural city and identify how much one can reduce the time and costs of development while maintaining a good level of design.

The cost of development was significantly reduced with the design of procedural buildings. Although, because of this, the level of design was also reduced in certain areas. However, with the variety of objects that generate around the city, I think the level of design is at an acceptable level. To specifically identify the extent at which a developer can reduce costs, I would certainly say that I reduced enough, if not too much. A wider variety of buildings could be introduced and spend more time on dedicating unique designs.

The aim was then broken down into 6 objectives to enable me to reach a high standard:

1) **Analyse three different techniques used for randomising the map of the city, where roads and buildings are placed in a structured way throughout.**

   The first method for randomising the map is the use of a Voronoi diagram. Randomising the size of each region, as well as the content in each one has ensured random outputs with every generation. This was specifically important for this project as I wanted the layout of the map to have variations.

   The second method is the L-system roads. Within residential regions, road structures would generate using L-system algorithms. The road maps produced are randomised to an extent, but also follows strict rules to guarantee realistic results. To randomise these results even further, curved roads and small paths could have been added which would also make the city more realistic.

   The last technique is the use of a Bezier curve to generate the river. With the placement of each centroid being random in every map, the created curve always forms an interesting, natural shape.

   Although this is not so much of a technique, I also want to mention that the procedural placement and design of buildings, roads, and vegetation greatly adds to the randomness of every city. Although similar objects may be generated when comparing two generated cities, objective 1 is have achieved to a very high standard.

2) Discuss and identify the most effective way to reduce the costs when developing a large map

Using the research in chapter 2, I created a procedural city that reuse hundreds of objects to reduce the costs of development. This also heavily reduced the time of development to in some areas which in terms, reduces the cost as well.

For a program that can generate thousands of objects, there are hardly any stored prefabs. For certain, if most of the generated objects were pre-made, there would be a significantly larger cost of production as well as the content size.

3) Explore different geometry types within buildings and road structures and identify what makes a city look realistic.

The downfall I faced with this objective was that I could only implement roads that are perpendicular to one another, which is clearly not very realistic. However, shapes in buildings were designed well and so met the first part of this objective to a good level. Some are simple rectangular apartments, and some are triangular skyscrapers, using a pyramid shaped roof. Apartments with different shapes and sizes were also explored; some thin and tall, some flat and wide, and of course, some with different levels where random floors would generate smaller than the floor below it. These features did capture some of the designs that we see in real life, however, less box-shaped buildings would have been a great addition to make the scene more realistic.

4) Develop an algorithm that will generate different areas of a city, including a city-centre, with taller and more unique buildings.

The created algorithm uses a Voronoi diagram to split the terrain into a number of cells. These cells represent specific regions which determines the content of each cell. The two cells in the middle of the map will always be used as the city-centre. The different regions are set to generate different types of buildings specified in chapter 3 [3.6 Development]. To seclude the city-centre visually, procedural skyscrapers, shops, and a bridge over the river is set to generate, which will not be found in any other region.

5) Evaluate the time taken to create unique buildings that are only generated a few times as well as the time taken to create buildings that are repeated throughout the city.

Previously mentioned in chapter 4, creating the apartments from scratch to the first model took about 3 days of work [4.2 Time and Costs]. The scripts were then modified many times to add finer details through the course of this project. With the experience I have now, I will be able to recreate this in less than a week.

If compared to skyscrapers, I would say that the apartments took much longer to design. In fairness, there are much more variations to the apartments and uses more building blocks than the skyscrapers. However, this was necessary to ensure repetitiveness was reduced as I knew the apartments would be used in multiple areas. If made from scratch, the skyscrapers will take a day or two to recreate.

I decided to create a few things in the city manually for comparison. A simple carpark which is generated once in the city took about 30 minutes to design. It was made using some flattened cubes and some props to elevate the design. If I created buildings the same way, it would have certainly taken me longer to create such a big map.

In other words, the procedural generators have shaved hundreds of hours of work to design and implement objects, which also adds to the success of meeting the second objective.

6) **Evaluate and compare the differences in three cities which use a different number of objects or building materials.**

The result of this evaluation is made on chapter 4, which looks at the impact on the hardware between different sized cities [4.4 System Testing]. It also looks at the city as a whole, to determine which outcome is best suited for this program and for the purpose of the users.

The final decision is that a city with a population density of 4 or 5 is sufficient, which creates roughly 400 to 500 buildings. With well determined LOD on the generated objects, it will help keep the performance of the program at a high enough level.

5.3 Personal Development

Over the course of this project, I have developed a wide range of key knowledge and understandings that has shown huge progression as a programming student.

Starting with chapter 1, I was able to identify a problem and creatively come up with a solution to base this project on. This shows creativity and problem solving skills in a knowledgeable way.

Chapter 2 gave me an insight into conducting useful, effective research on the areas that matter the most. The studies had to help me achieve the objectives and so I had to learn how to find the most important information in an effective way.

In chapter 3, as well as my overall programming skills, more problem solving was needed to face some of the challenges. As seen with the roads and river generation, I ran into some unexpected problems. I had to then find ways to solve this situation, so the objectives would still be met.

Chapter 4 hugely improved my evaluation skills where I was able to make in depth analysis of this project using the final two objectives as reference.

As well as personal skills, I have made huge developments on C# and Blender which will certainly come in handy in the future.

5.4 In the Future

I have created a list of areas to improve\change for future development:

- Create procedural houses with hundreds of variations – a bigger variety of buildings in general would be good to see and the houses can adapt a similar approach to the ones currently used, which represent wealth in different regions.
- Explore more shapes with the procedural buildings – this will improve the aesthetic of the city as a whole. More wall designs and shapes could be used to help this.

- Implement curved roads – The straight roads on their own lacks creativity for the road structures and makes it less realistic.
- Implement different culture and designs from around the world seen in chapter 2.4. – as of now, every city looks similar from a distance. Randomising the base structure of the city using inspirations from around the world would allow me to create more realistic environments.
- Use different approach to find space for buildings to be generated – with the physics method I currently use, some errors can be found when generating buildings around roads. It is not often but should be improved.
- Improve the designs of the nature region for a wider variety. A selection from a forest, park and lake should be implemented in every city.

References

[1] Togelius, Julian, et al. (2011). What is procedural content generation? Mario on the borderline. Proceedings of the 2nd International Workshop on Procedural Content Generation in Games. ACM.

[2] Making it in Unreal: how Frogwares built The Sinking City so they could drown it [online] Available at: https://www.pcgamesn.com/the-sinking-city/frogwares-unreal-engine-4 [Last accessed 20 Feb 2022].

[3] (2021) Video Game Industry Statistics, Trends and Data In 2022 [online] Available at: https://www.wepc.com/news/video-game-statistics/ [Last accessed 26 Feb 2022].

[4] Surej, S. (2020) Former PlayStation exec on AAA games model: "It's just not sustainable" [online] Available at: https://www.nme.com/news/gaming-news/former-playstation-exec-on-aaa-games-model-its-just-not-sustainable-2694462#:~:text=Layden%20compared%20when%20the%20world's,is%20it's%20just%20not%20sustainable." [Last accessed 26 Feb 2022].

[5] DTI. (2002). From Exuberant Youth to Sustainable Maturity, 8 - 14.

[6] Kelly, G., & McCabe, H. (2006). A survey of procedural techniques for city generation. ITB Journal, 14(3), 342-351. [online] Available at: http://www.citygen.net/files/images/Procedural_City_Generation_Survey.pdf [Last accessed 03 Mar 2022].

[7] Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., & Worley, S. (2003). Texturing & modeling: a procedural approach. Morgan Kaufmann.

[8] Jormedal, M. (2013). Procedural Generation of Road Networks Using L-Systems [online] Available at: https://liu.diva-portal.org/smash/get/diva2:1467574/FULLTEXT01.pdf [Last accessed 04 Apr 2022].

[9] Najahi, Y. (2017). Urbis terram-designing and implementing a procedural city generation tool for unity3d game engine. [online] Available at: https://core.ac.uk/download/pdf/213002226.pdf [Last accessed 04 Apr 2022].

[10] Frank, E., & Olsson, N. (2017). Procedural city generation using Perlin noise.

[11] Unite Copenhagen. (2019). Design and modify a realistic procedural city in real-time. [online] Available at: https://www.youtube.com/watch?v=bvrA_438KkA&ab_channel=Unity [Last accessed 04 Apr 2022].

[12] Batty, M., & Longley, P. A. (1994). Fractal cities: a geometry of form and function. Academic press, chapter 1. [online] Available at: https://discovery.ucl.ac.uk/id/eprint/1370661/6/1370661_Fractal%20Cities%20Chapter%201.pdf [Last accessed 10 Mar 2022].

[13] Ajmera, A. (2020). Role of Geometry on Architecture. [online] Available at: https://www.researchgate.net/publication/342538094_Role_of_Geometry_on_Architecture [Last accessed 11 Mar 2022].

[14] HP. (2021). The Top 50 Best-Selling Video Games of All Time. [online] Available at: https://www.hp.com/us-en/shop/tech-takes/top-50-best-selling-video-games-all-time [Last accessed 13 Mar 2022].

[15] Lechner, T., Watson, B., Wilensky, U., & Felsen, M. Procedural City Modeling. [online] Available at: https://ccl.northwestern.edu/2004/ProceduralCityMod.pdf [Last accessed 1 May 2022].

[16] Najahi, Y. (2017). Urbis terram-designing and implementing a procedural city generation tool for unity3d game engine. [online] Available at: https://core.ac.uk/download/pdf/213002226.pdf [Last accessed 20 Mar 2022].

[17] Liapis, A., Yannakakis, G., & Togelius, J. (2013). Towards a generic method of evaluating game levels. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (Vol. 9, No. 1, pp. 30-36). [online] Available at: https://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewFile/7374/7585 [Last accessed 20 Mar 2022].

[18] Shaker, N., Togelius, J., & Nelson, M. J. (2016). Procedural content generation in games. Switzerland: Springer International Publishing. [online] Available at: https://link-springer-com.libproxy.ncl.ac.uk/content/pdf/10.1007/978-3-319-42716-4.pdf [Last accessed 20 Mar 2022].

[19] Clark, J. H. (1976). Hierarchical geometric models for visible surface algorithms. Communications of the ACM, 19(10), 547-554. [online] Available at: https://doi.org/10.1145/280811.280921 [Last accessed 23 Mar 2022].

[20] Jabłoński, S., & Martyn, T. (2016). Real-time rendering of continuous levels of detail for sparse voxel octrees. [online] Available at: https://core.ac.uk/download/pdf/295592341.pdf [Last accessed 23 Mar 2022].

[21] https://unity.com/

[22] https://www.unrealengine.com/en-US/

[23] Arora, S. (2021). Unity vs Unreal Engine: Which Game Engine Should You Choose? [online] Available at: https://hackr.io/blog/unity-vs-unreal-engine#:~:text=Unity%20has%20a%20wide%20range,making%20the%20development%20process%20easier [Last accessed 16 Apr 2022].

[24] https://visualstudio.microsoft.com/

[25] https://www.blender.org/

[26] https://javascript.info/bezier-curve

[27] MCV Staff. (2013). Inside Rockstar North – Part 4: The Art [online] Available at: https://www.mcvuk.com/development-news/inside-rockstar-north-part-4-the-art/ [Last accessed 17 Apr 2022].

Glossary

2D – Two-dimensional

3D – Three-dimensional

AAA – Triple A

GTA-V – Grand Theft Auto 5

PCG – Procedural Content Generator

FPS – Frames Per Second

LOD – Level of detail

IDE – Integrated Development Environment

OOP – Object-oriented Programming

FBX – Filmbox Format

GUI – Graphical User Interface

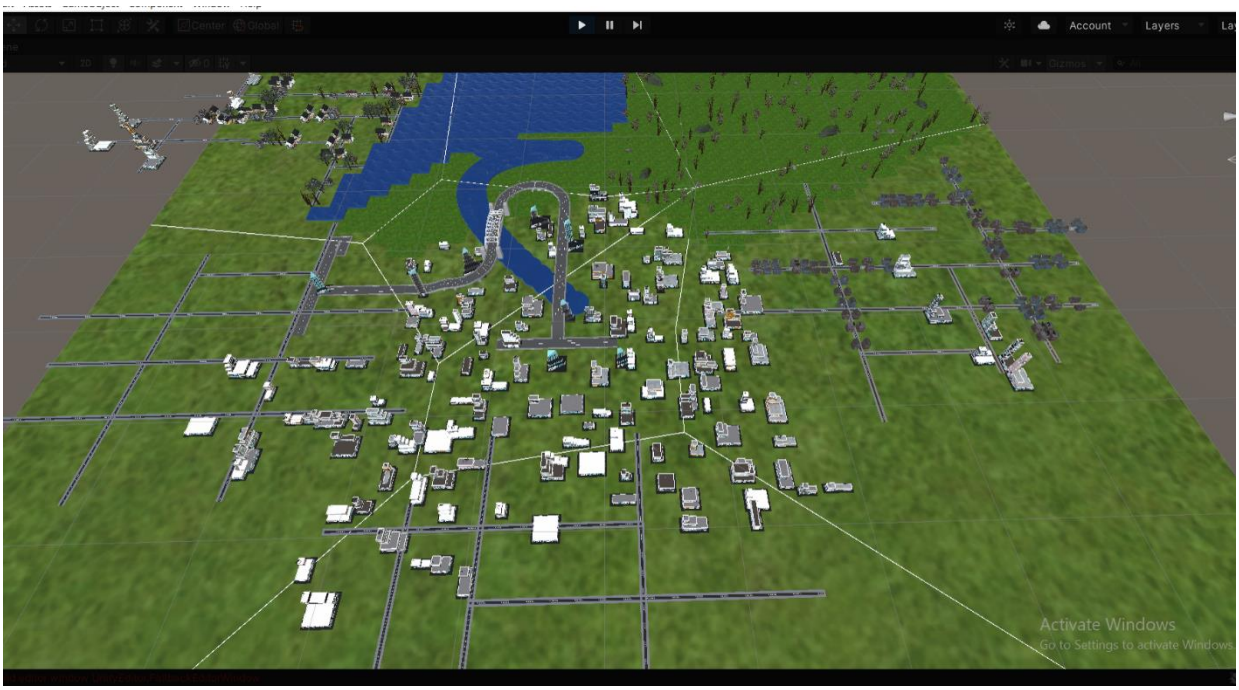
GPU – Graphics Processing Unit

Appendix

Testing

A city from above, with population size 3.

A city from above, with population size 5.

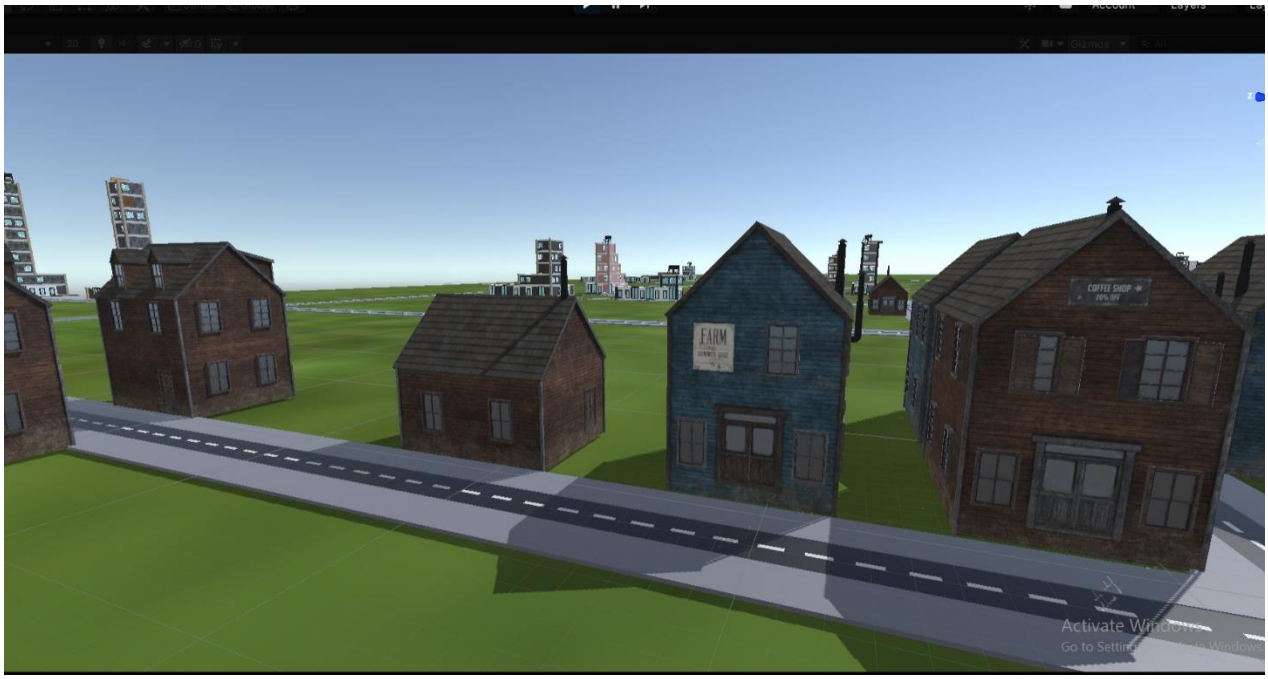


A city from above, with population size 8.

Analysis

A close-up shot from the residential area with both wealthy and less wealthy in shot

A street in the less wealthy region



A street of wealthier houses

Basic concept of commercial buildings in the city-centre