Procedural generation of Cityscapes

Final Report for CS39440

Author : Rhys Johnson (Rhj44@aber.ac.uk)
Supervisor : Fred Labrosse (ffl@aber.ac.uk)

29/04/2024
Version 1.0 (Release)

This report is submitted as partial fulfilment of a BSc degree in Computer Science BSc (G400)

.

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that :

- This submission is my own work, except where clearly indicated
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to a loss of marks or even the withholding of a degree
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current student handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name : Rhys Johnson
Date : 29/04/2024

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name : Rhys Johnson
Date : 29/04/2024

Abstract

Procedural generation has quickly seen adoption and considerable development in the Entertainment industry as a method of reducing work load while also creating larger and more detailed environments, This project presents an approach to the procedural generation of cities, with a focus on producing believable results and user control of the generation

Based in Unity the program uses procedural techniques to generate the terrain of the city, the road layout and the buildings, this paper explores the background of the project, the methods used for implementation, the design and how it was tested.

Contents

1 Background, Analysis and Process

1.1 Introduction

This project aims to create a system that can procedurally generate a cityscape, with a specific interest in exposing the variables controlling the generation to the user to allow a diverse range of cities that look like they could theoretically exist.

1.2 Background

Procedural generation is a technique of content generation where the user provides a set of start variables or conditions to an algorithm which then generates a random output based on those start variables, there are many different ways of designing a procedural system/algorithm but they all work on this basis. Procedural generation is becoming an increasingly used technique in the entertainment industry, as the demand for larger digital worlds and solutions increases. The time needed to create a city, a planet or animate a large group of people [**1**] simply grows to large for a developer or artist to hand make themselves and a procedural solution must be used.

1.2.1 Problem overview

The procedural generation of cities presents a unique challenge given both the complexity of city streets and the formation of cities themselves as they are generally formed over many years and are affected by a myriad of different socio-economic factors that will vary from region to region and over time to create the cities we see today [**2**]. This poses problems when it comes to procedural generation as to create a believable and coherent city scape the system must simplify these factors into a more generic city generating algorithm. Below are outlined some of the key problems faced in this project.

1.2.2 Believability and Realism

One of the primary goals of this project is to produce cities that are "believable" - which is to say that a person could see the generated city existing in the real world, this is separate I believe from the idea of realism which can be quite vague when it comes to computer generated graphics, James A. Ferwerda describes three types of realism in his paper "Three Varieties of Realism in Computer Graphics" [**3**]. For this project we use his definition of functional realism "the image provides the same visual information as the scene". While in his paper he focuses on the idea of someone being able to complete a task represented in a virtual environment despite its lack of photo realistic detail, here we use it as we are creating a simulacrum of a city. The general shape and terrain details are more important than producing photo-realistic reproductions of buildings and streets. Hence we reach a good intersection of believability and realism where the form and layout of the city is the most important aspect of measuring success with this project.

1.2.3 Roads

As the network that underpins a city, roads are the most important factor when dictating the shape and form of a city. They are dictated by many outside problems similar to the those described in 1.2.1. Mainly; the history that formed the city, terrain considerations and population centers. Other considerations must be made such as the hierarchy of roads, for example if we take a basic simplification of road hierarchy main, side, ally ; A main road wouldn't connect to an ally and then back to a main road and this must be accounted for the believability and realism of the city discussed in 1.2.2. Another problem to address is road patterns, outlined in a paper by the Ministry of War Transport [**4**] and implemented in another paper on city generation [**5**]. Road patterns dictate the overarching form of the city and are generally formed from the history of the city and how it has expanded over time. For example cities in the USA tend to have a grid layout which is due to the planning of cities from the ground up and developing alongside different technologies (namely cars) compared to European cities that have developed and expanded over hundreds of years (London itself has existed in one form or another for about 2000 years with archaeological evidence pointing to prior iron age occupation of the region [**6**]).

1.2.4 Terrain

Terrain as a primary factor in the development of cities is important to consider for this project, the hills and valleys, rivers and coastlines surrounding a city will have a profound impact on how and where the city develops. Terrain dictates where further development takes place in a city, the land use (residential, business, etc.) and whether development happens at all. Cities in the past have developed in areas with high access to vital resources: food (in the form of fertile arable land), water for agriculture, trade, hygiene and consumption; and materials used for further development [**7**]. With the advent of the industrial revolution, technological advancements have allowed humans to get appropriate resources from further afield and move services outside of cities (Infrastructure like Damns and power stations that have been moved outside or to the outskirts of cities) [**8**]. Cities nowadays generally start as centrally planned communities less reliant on the constraints of the past and history when it comes to accessible resources in the local area (examples include Brasilia the new capital of Brazil, the New Administrative Capital in Egypt and Dubai in the UAE) which makes the terrain of the location a more important factor when deciding on potential city locations.

Terrain plays a role in the local development as well, roads will generally avoid areas of overly difficult terrain (marshes, steep hills/mountains, flood plains, etc.) due mainly to cost of construction with more extreme environments requiring specialized engineering (tunnels, bridges). There may also be factors such as the environmental impact of roads on the terrain, the destruction of rare habitats or ecosystems is a consideration when building infrastructure [**9**].

1.2.5 Variation

Variation of output is one of the main goals of procedural generation but can lead to incoherent outputs if the algorithm is improperly tuned, for this project we need to be able to generate a diverse array of cities that are unique but also believable. This poses a

problem, as discussed above cities are the product of a diverse range of social, economic and geographical factors that need to be considered and implemented in a coherent manner. We must decide which factors we use in the generation of our cities and their level of impact on the final output. We also must decide which factors should remain a constant or integral to the generation and which to parametrize and allow the user control over, and in the case of user control the range of values which still lead to believable results.

1.2.6 User Control

The main problem with user control in this project is the complexity of the generation, providing too many options for tuning and control may prove overwhelming for a user and make it hard to generate a city that fulfils the goals for believability and realism, on the other hand not providing appropriate abilities to change generation parameters would impact the variation of the outputs. Extending from this problem is the general lack of transparency in procedural generation algorithms, without proper naming and labelling of control parameters or documentation of the system, users may find it difficult to work out what a parameter controls and the affect on the output. We must also consider the ability of a user to achieve a specific goal and how close they can get to a desired output through tuning the parameters. While there will always exist some fuzziness with procedural systems due to the nature of their randomness a successful system will allow the user to generate something close to the desired goal.

1.2.7 Tools

For this Project two major tools were investigated as potential candidates. These were Blender and Unity(3D).

Blender is an open source 3D modelling software [**10**]. It is mainly used for non procedural modelling of 3D assets for films and games but has a Python API which can be used to create any number of packages and extensions, a number of user created add-ons exist and show that a procedural city generator would be possible in blender [**11**]. Another powerful tool Blender possesses is the geometry node system which allows users to create procedural geometry by combining various nodes together, with each node representing some sort of operation (addition, subtraction as well as more advanced computational nodes). This can allow very complex results from relatively simple node networks [**12**]

Unity is game engine for both 2D and 3D development, while mainly used for making games it can also be used for a variety of 2d or 3d interactive experiences [**13**]. It uses a system of game Objects which hold components that define a specific behaviour, such as a physics system, renderer or custom script; this allows a very modular approach to the design of systems. It also has a very user friendly UI and allows values to be tweaked in real time for different outputs relatively quickly. Unity's primary language is C# which allows access to a huge amount of libraries for use in the project if needed.

Other tools considered Other tools were investigated and found to be inadequate in some way or another but they may prove useful for others attempting a similar project:

- CityEngine : CityEngine is an industry tool for the massive generation of cities developed by Esri, it's design was first laid out in a paper by Yoav I H Parish and Pascal Müller [**5**] which goes in to great detail about how they generate their procedural cities.

- Houdini : Houdini is a 3D modelling software extensively used in the film and games industry, similar to Blender it can be used to generate procedural geometry from a node network. The main reason that this program was not considered further was due to it being a paid program, it was decided that Blender being a free, open-source program warranted further investigation over Houdini. Despite this i believe it would be a tool worth further consideration for anyone attempting a similar project who has access to it, as it contains many powerful features over Blender.

- WorldMachine : World machine was investigated to generate the terrain that the city would sit on, it's a powerful editor for generating procedural terrain [**16**] and contains an extensive tool set for generating any type of terrain that would be required, unfortunately it was not used as the importation to Unity would not allow for automatic update when the user changes values in the Unity editor, it would require the user to generate the terrain, then import to Unity and generate the city on top which is not the end goal of this project.

1.3 Analysis

As discussed in the background many challenges exist in the generation of city, for the scope of this project practical limitations had to be decided early in development and planning as to make best use of the time allotted. More importantly the features most important to the generator had to be decided. The main areas of focus for development were:

- Roads - As explored in the background roads form the shape and form of a city and are therefore one of the most important factors to consider for this generator a number of techniques were analysed for suitability given the requirements of the system and feasibility of implementation given the time frame.

  - L-Systems : In his paper on procedural generation of road networks [**17**], Martin Jormedal explores a Graph based approach to road generation using L-Systems, The L-System was originally developed as a formal way for simulating the growth of bacteria but was expanded into more complex plant structures such as trees and ferns [**18**]. A L-System has: an alphabet of symbols which represent a components of the object to be modelled; a list of rules that dictate how symbols will be changed or replaced and an axiom or start condition. The L-system will apply an appropriate rule to the axiom replacing it with either a different or a sting of different symbols, the L-system will continue to rewrite this string of symbols based on the rules until either an iteration limit is reached or no more rules can be applied, the resulting string will then be visualised. Martin Jormedal uses a graph implementation of a L-system, the symbols of the L-System are the nodes of the graph and the connections between them the edges, this allows for cyclic road structures which is important for the believability and realism of the generated cities. L-systems offer a range of strengths when considering them for the generation of roads and the goals of this project, both the symbols and rules can be decided on or changed by the user for example for varying outputs while using the same under lying algorithm, it will also allow for limitations such as number of roads generated or maps sizes to be controlled by the user.

  - Wave Function Collapse (WFC) : WFC gets it's name from quantum physics, where the wave function defines a probability of an electrons position [**19**], when observed the system will "collapse" into a definite state. The WFC algorithm works similarly, it generally takes the form of a grid of cells and a tile set of potential outputs for a cell, given some axiom the algorithm will first assign each cell a list of it's possible states from the tile set, based on the axiom and rules imposed upon the system. It will then iterate through each cell and "collapse" its state to one of the possible tiles based on surrounding cells, rules and probability that the output tile has for that cell [**20**]. Marian used this algorithm to create an infinite procedural city [**22**] which is adjacent to the goals of this paper, although the final output of that project is not inline with the believability and realism goals discussed in 1.2.2. To make a believable road layout they must follow terrain and link population centers to one another, this could be achieved through the WFC algorithm, the axiom would be population

centers and the probability of certain tile states depending on the terrain and population values of that tile. it offers some of the same benefits as the L-system when it comes to generating roads but is limited by the grid system it lies on and that every possible permutation of a road on a tile must be made for the tile set initially, additionally if a user wanted to add a new tile output to the tile set they must also set all the adjacency rules for it which increases friction in the user experience which goes against the goals for user control in this project

  - Tensor Fields : A tensor is simply an algebraic object used to describe a property often used in physics and engineering to represent things like stress on a part or velocity of a fluid. A tensor field describes where these tensors are in a space. Street modelling with tensor fields has been explored extensively [**23**], this paper shows that excellent results can be achieved with tensor fields while allowing high amounts of user control over the final output. It also allows for multiple maps to be integrated into generation (water, terrain, other disallowed areas). This was an area of interest but unfortunately a decision was made to not pursue this method after some experimentation with it, as it was recognised that the expertise needed to execute it laid beyond the current skill set of the author.

  For this project the L-system method was used for road generation, during initial testing of methods the implementation of different rules for generation and variety of achievable results that the L-Systems produced was satisfactory for the goals of believability and realism we are striving for as well as offering the user a good level of control over the final output of the algorithm.

- Terrain - Terrain poses a challenge for procedural generation as it forms over millions of years and is affected by a variety of geological and atmospheric factors, modelling all these factors would be virtually impossible and far outside the scope of this project so to generate coherent geology primary features must be found to focus on and a level of detail must be decided. For this problems two main methods were analysed:

  - The first method was to not use procedural generation at all, instead terrain would be generated from a user supplied height map this was the method used in [**5**], [**17**] and [**23**] to great success. The benefits with this method is that it theoretically allows a user to input any number of maps to be used for the generation of road networks, it quickly allows users to generate different city layouts using the same terrain. It can provide a higher resolution for the terrain especially when real-world terrain maps are used. However it also reduces feedback to the user, terrain maps may require editing to achieve the desired goal and not being able to see the terrain in real time makes this process harder.
  - The second method was to use noise to generate terrain, specifically Perlin noise [**24**]. The advantages of using Perlin noise are it is an efficient algorithm meaning that it's suitable for using in real-time allowing users to see how changing generation parameters affects the terrain instantly. It also opens the ability to expose many parameters to the user allowing for the fine tuning of terrain features and

appearance over user supplied height maps. The main problems are that it is often hard to recreate realistic terrain with just perlin noise, other methods are often employed such as layering offset perlin noise to break up large shapes, or simulating hydraulic erosion [**25**].

For this project generating terrain through perlin noise was used many methods can be used to improve the believability of the terrain [**26**]. The speed of Perlin noise means users can quickly see in real-time how changing parameters will affect the terrain, it also allows the program to expose many parameters to the user, so they can fine tune the final result to their liking. There are constraints with using this system, it can be hard to add bodies of water and rivers; this feature was chosen not to be included in this project due to this, as the terrain could still reach the goals of believability and realism without them (Scott turner outlines some more challenges with procedural generation of terrain and other systems in a blog post [**27**]). For this project the variation and user control that perlin noise provides is more important than producing truly realistic terrain from the generation and as discussed in 1.2.2 the goal of this project is to focus on the shape and form of the city over photo-realism.

- User Control A few problems arise when considering user control of the generation, the program must ensure that user selected parameters are valid and also remain user friendly and provide proper feedback :
  - The main problem to address is which parameters to expose to the user for generation, and if a parameter is exposed what ranges of values provide a good variation in output and which lead to either incoherent results or the system not being able to generate(negative road length for example). For this project we aim to expose as many parameters as possible to the user so that they can produce a variety of results. If generative sections are split into components we can reduce the amount of parameters the user has to control at once, by breaking generation into : terrain, roads and buildings, we can allow the user to tweak the parameters of each without affecting overall shape and form of the city.
  - The next problem is which parts of the generation should be completely random and which should be deterministic, deterministic generation means that given the same input the system should always output the same result, the choices of techniques of other systems dictate this somewhat. For terrain, the generation can be made deterministic by always sampling the perlin noise from the same point, whereas for the roads; L-systems in this context would not be possible as rules are picked from weighted probabilities based on various factors (terrain, population, other roads) due to this it can not be guaranteed that the same result will be output every time.
  - Usability and understanding of what the parameters affect in generation is also important for user control, some effort must be made towards the presentation and naming of parameters as well as having a well documented manual. Without this it could be hard to achieve the variety and believability the project strives

for. The choice of Unity helps somewhat in this regard as it provides powerful tools for customizing editor windows and allows changes to be seen in real time.

1.3.1 Functional Objectives

Given the problems explored above the core objectives of this the generator are:

- **Terrain Generation**
  1. The system will use perlin noise to generate a diverse range of terrain height maps
  2. Users will be able to adjust the parameters of the perlin noise (amplitude, layers, lacunarity, gain, seed)
  3. The system will update the terrain in real time when parameters are changed
  4. The terrain generation shall be integrated with both the building and road generation, terrain features shall influence the layout of roads and buildings
- **Road Generation**
  1. The system will use a L-System to generate a road system that conforms to the terrain
  2. Users will be able to set the rules of the L-System
  3. Users will have control over generation parameters (such as max terrain height, the axiom and maximum roads).
- **Buildings**
  1. The system will generate buildings following the road pattern
  2. Users will be able to adjust the parameters of building generation (max height, max/min terrain height, max allotment size)
  3. The system will generate a variety of building shapes to generate a diverse range of skylines
- **Believability and Variety**
  1. The system will generate a wide assortment of city shapes and forms that look like they could exist in the real world
  2. The system will provide appropriate tools to the user to dictate the form and shape of the city, allowing for greater variety and user control.

1.3.2 Non-Functional Objectives

- **Usability and Accessibility**
  1. The system shall provide a user friendly UI that allows for the easy adjustment of the parameters for city generation

2. The system shall provide tooltips on the function of parameters to allow for further understanding of it's affect on generation

- **Performance**
  - The system shall be able to generate cities of a large scale in a reasonable time frame (based on factors such as map size and road/building count);

1.4 Process

The process used for this project was the spiral model, which consists of 4 main steps:

- Planning - Each iteration of the spiral plan starts with a planning phase, the main goals of the phase are decided and analysed looking primarily for the risks associated with the phase under scrutiny and any alternative solutions that could be used to achieve the objectives.
- Risk Analysis - The potential pitfalls of the phase are scrutinised, alternative methods are investigated and evaluated. Prototypes may be produced to test alternate methods.
- Engineering - Based on the above steps, a solution is implemented to the project.
- Testing - The implementation is tested to ensure it fulfils the requirements and to debug any faulty code. Also to ensure that it interacts correctly with any previous components.

This method was chosen as it allows for flexible planning and requirements during development, The problem was broken down into 3 main components of development; terrain, roads and buildings. each acted as it's own spiral process that fed into the main program to generate the city. The various techniques investigated during the planning process of each component are described in section 1.3, the main components of risk analysis were; difficulty of implementation, time required to implement and how the component would achieve other goals of the project (user control, believability and realism, performance requirements). Early in planning it was decided to develop these components sequentially as each feeds into the next. Roads require terrain data for generation, buildings require both road and terrain data for generation. Fortunately due to the spiral model this allows for flexibility in requirements and planning for later sections based on the success and testing of the previous component which reduced the risk that large sections of implementation would need to be changed later on. While this initial plan seems reasonable it obfuscates a potential down side of this method. The number of spiral phases for each component was initially unknown and would be decided and planned when each component was reached, this makes a time for completion difficult to arrive at and poses a risk that too much time could be spent developing one component at the cost of others. This was mediated by the decision to self-impose cut off dates for each component and considering it delivered (with the exception of fixing bugs or other unintended behaviours found during the development of subsequent components).

1.4.1 Consideration of alternate planning methods

During initial analysis of the problem a few planning methods were considered the primary options investigated are:

- Scrum : The scrum method emphasises short sprints of planning, implementation and evaluation, allowing for rapid prototyping and flexibility in requirements of the final product. Each sprint starts with a planning phase where goals for the sprint are decided and risks are analysed, the next step is the sprint itself where the plan is implemented, after the sprint there is a review and retrospective to evaluate work done and how future

sprints could be improved. A daily scrum also occurs to discuss work done, challenges and plan for the day. This method was not chosen as it is a solo development project, scrum emphasises communication throughout a team of developers without this scrum loses a lot of what makes it a strong development process, people to discuss ideas with, insights from team members and feedback from an outside view this can lead to a lack of perspective and tunnel vision on certain implementation methods. It also makes it more difficult to estimate completion time for components as estimations for capacity of work and technical complexity may be wrong ultimately leading to either over running on the time allotted or unrealistic goals for the component. Because of this it was decided a top down model that involved more initial planning for each component was more appropriate for the nature of the project and would allow for better time management.

- Waterfall : The waterfall method is a top down approach to project management with 5 main components. First a requirements stage where the problem is analysed and requirements are created to understand the scope of the project, Second a Design stage, a design document is created outlining software architecture and components, third the development stage where the system is implemented, fourth the testing phase where the software is tested to make sure it fulfils requirements, and finally deployment of the final system. This model has the advantage of doing all the planning upfront, goals and scope will be defined for the whole system upfront which allows for more rigorous risk assessment than the scrum or spiral model and a lower likelihood of components posing unknown challenges. However this comes with the trade-off of making the plan fairly inflexible it relies on knowledge of what problems could occur during each components development and each phase of the model at the start of development and due to a lack of prior experience on procedural systems of this scope it was hard to plan time frames for implementation of components and how plans for implementation could change during development. Whereas with the spiral model it would be easier to shift goals during development when unexpected problems occurred, it also allows for more frequent evaluation and feedback on components allowing for refinements in the design of the system and the estimated time for completion, this was important for a project where challenges for each component were difficult to identify early due to the complexity of the system and lack of previous experience.

2 Design and Implementation

2.1 High Level Design

The core of the system is three main components that control the generation of cities, these are the terrain generation, road generation and building generation. This was inspired by the approach taken by Yoav I H Parish and Pascal Müller in Procedural Modeling of Cities [**5**], here they follow a sequential approach to generation with each component using the data generated from the previous which is required for generating cities that follow the natural contours of land and adhere to the rules of the system

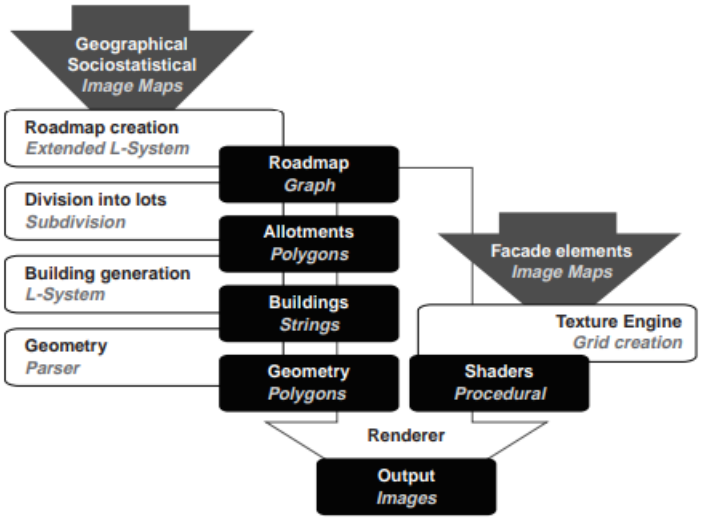

Figure 1: System architecture of CityEngine proposed by Yoav I H Parish and Pascal Müller

While their approach for terrain used image map inputs and a different method of building generation we can use this model as a starting point for the project. Our choice of tool (Unity) also impacts the high level design as it provides a variety of tools for visualisation and rendering which allows us to simplify the design somewhat. With these main components identified it is possible to create a simple architecture for our system

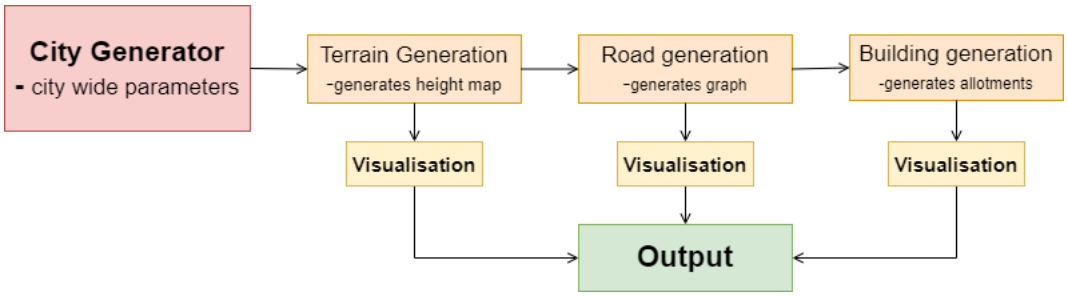

Figure 2: Initial high level design of the system

The above diagram shows a basic representation of the required components of the system; The user sets the city wide parameters then each component has separate parameter control relevant to the functionality, this is to provide encapsulation for the user and allow them to focus on the generation of each component separately if required.

The Terrain component produces a height map of normalised values (0 to 1) using perlin noise, it can exist as it's own stand alone generator which allows for real-time shaping of terrain due to speed of the perlin algorithm. This is immensely useful for creating terrain that fulfils the desired goals of the user. Perlin noise was chosen as it allows for a huge diversity of shapes and also a number of useful parameters for generation, this helps us fulfil one of the primary goals of the project which was a focus on user control and generating a variety of different cities. It can also be modified by other noise to enhance the final result, usefully for our case Unity has a component called the animation curve [**28**], which while generally used for changing object properties over the course of an animation can be used to evaluate values based on the curve allowing for even more user controllable options, this is expanded in the detailed design section for terrain.

The Road Generation component takes the height map and use a L-System to generate the streets, there are number of parameters to control the L-system, most notably rules can be made to modify the generation and set parameters such as max turn angle, road length and branch probability. The output of this system is a graph. The user can also define parameters for road generation based on the terrain. L-Systems provide a good balance between user control and variety in output, unfortunately due to their semi-random behaviour, L-systems can't recreate the same street map given the same inputs which somewhat limits the users abilities to refine a map once made but the variety and extensibility of customisation a L-system provides makes this a valid trade-off.

The final step in the process is the Building generation, initially this was also intended to use an L-System but early prototypes of the system proved that modelling 3D structures with an L-system is significantly more complex than the node based generation of the roads and the time needed to properly implement this system would have come at the cost of other important features such as user controls (for readers interested in this topic Procedural Modeling of Buildings [**29**] is a comprehensive paper on such a system). Instead the final decided system was to generate rectangular allotments along road sections these then under go a few random transformations that user defined limits to generate the range of building shapes required to give the city a non-uniform and more believable appearance. The allotment system uses defined properties to dictate depth and height to allow the dialling of values to achieve the desired goal.

Each component has a visualiser, Unity offers many different tools for visualisation, notably for this project the Mesh class which allows us to easily create meshes and display them with the gameObject and component system this was used for both the terrain and building system, the terrain is initially a flat plane with the same number of horizontal and vertical vertices as the height map dimensions generated, the height of these vertices then was then offset by the height value from the appropriate height map value. Another useful visualisation tool unity provides is the Line renderer which allows us to set start, end, width values and color of lines. This is used to draw the roads of the city, the disadvantages of this system are that it renders a 2D plane which turns to face the camera, this is fine at farther distances as it is almost unnoticeable but up close small movements can drastically change

the roads normal direction. The trade-off is accepted as they are efficient performance wise and allow for the generation of larger cities that are not as taxing on the users hardware as a mesh solution would be. Unity's shader system was also used for the colour of the terrain, the ability to script behaviours means the user can dictate the colour and various other factors such as colour blend and colours based on mesh normals.

2.2 Low Level Design

2.2.1 Terrain

The terrain consists of six components that handle the generation of the height map and visualisation of the noise texture and mesh results. Terrain information is stored as a normalised 2D float array (between values 0 and 1). The height map is passed to both the mesh and texture generator before finally being displayed.

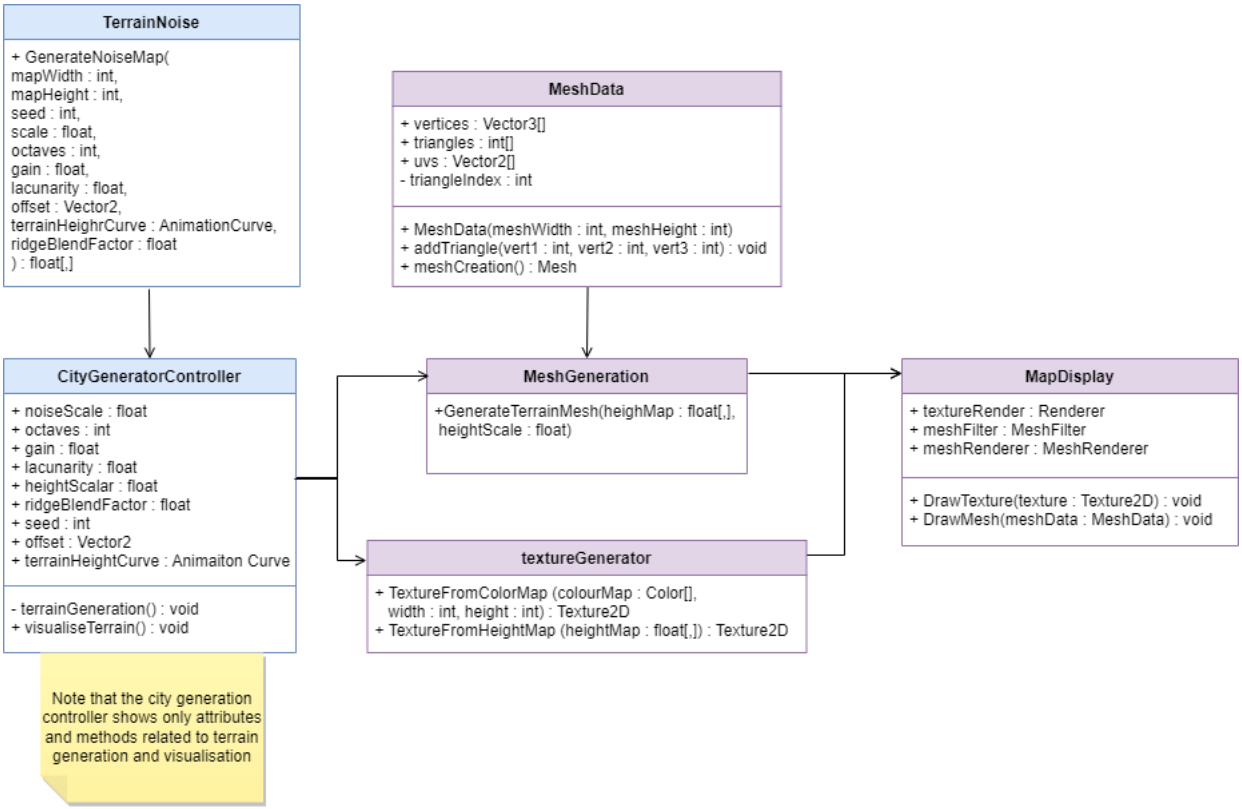

Figure 3: class diagram for terrain generation

The overarching controller of the system is the CityGeneratorController this centralizes all the attributes needed for the generation of the city, the diagram above shows the attributes used for the terrain generation. When called the controller will pass these attributes to the noise generator class. The attributes are

- mapWidth and mapHeight are the dimensions of the generated map

- Seed - The starting point for the algorithm to sample from. As Perlin noise is pseudo random we can set a seed to allow the recreation of a terrain map given the same inputs

- scale or noiseScale essentially is how much noise we can fit in a space/the zoom factor of the noise. A lower scale factor leads to more noise detail in a smaller space as well as vice versa. This allows the user control over how large the terrain looks for a generation. For the following figures have mapwidth and mapheight of 200

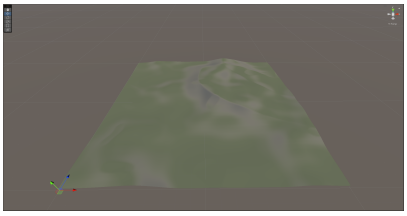

Figure 4: scale = 155

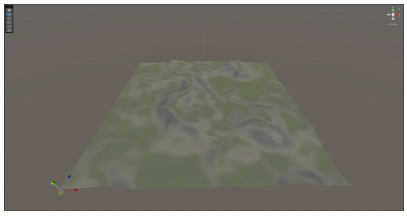

Figure 5: scale = 100

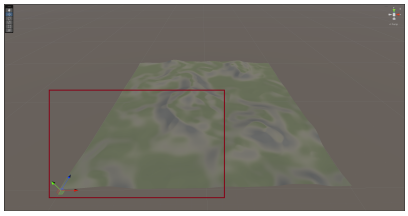

Figure 6: scale = 155

- Octaves - The octaves are the layers of noise, to produce realistic terrain a single layer of perlin noise is often not good enough as it is un-detailed and couldn't be parsed as realistic/believable terrain. For the following figures there was mapWidth and mapHeight of 200.

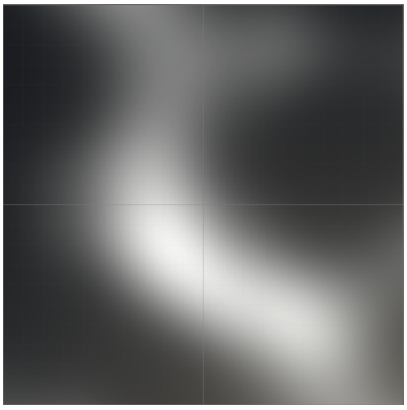

Figure 7: number of octaves = 1

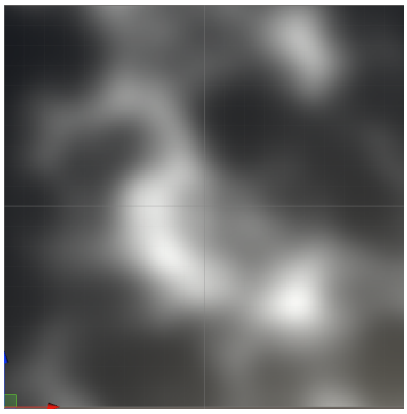

Figure 8: number of octaves = 3

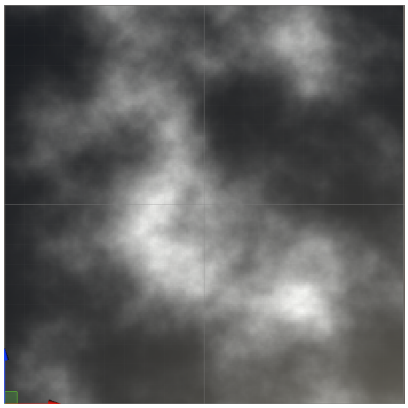

Figure 9: number of octaves=10

as seen in the figures above more octaves results in a higher level of detail of terrain, for each octave the sampled noise is offset which allows us to achieve this higher detail, although this comes with diminishing returns and exponential computation needed for higher octave levels.

- Gain - the gain factor dictates how much detail from layers affects the final noise, a higher gain will mean that details from higher octaves (normally they have a lower influence on noise to produce small terrain details) will mean more detail introduced by higher octave levels. This attribute has a range of zero to one with one meaning

that the details sampling from higher octaves will dictate the output of the noise, figure below show this change. For all below figures the map width and height was 200 and the octave level was 5

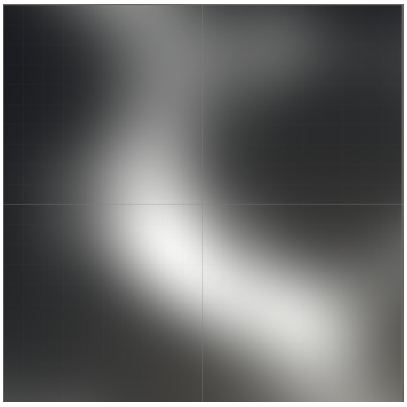

Figure 10: gain = 0

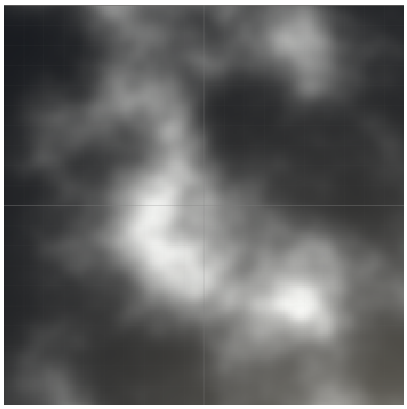

Figure 11: gain = 0.5

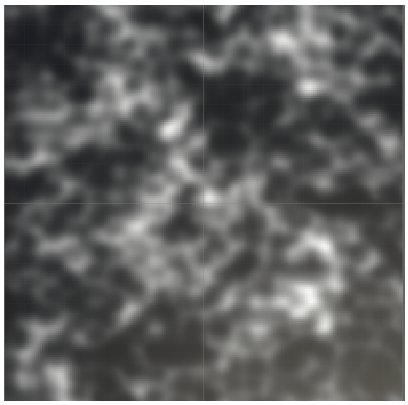

Figure 12: gain = 1

- Lacunarity - lacunarity controls the scale of each successive layer of noise (octave) relative to the previous octave, a higher lacunarity will mean each octave will have a larger scale than the previous one increasing the detail of the final noise map. For all the below figures Noise scale was 100, octaves were 3 and gain was 0.4

Figure 13: lacunarity = 1

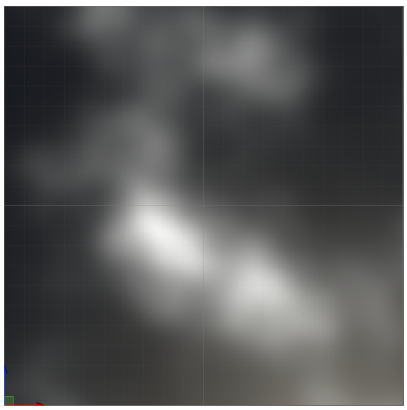

Figure 14: lacunarity = 2

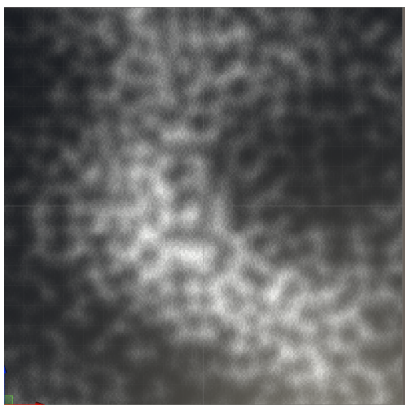

Figure 15: lacunarity = 10

- offset - the offset controls where the noise is sampled from, this is separate from the seed value which controls the origin position to sample from. The offset take the seed values and offsets it by the given x and y value provided. This allows users to scroll through terrain, increasing the ability for users to find a desired terrain to generate a city on.
- Terrain Height Curve - The animation curve is a powerful tool supplied by unity that allows us to evaluate values based on the curve, this means we can evaluate noise map values against this curve. This allows for powerful user control over the terrain shape

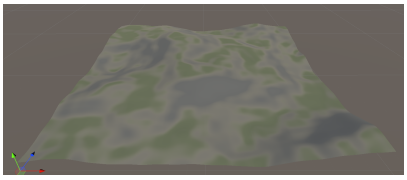

Figure 16: linear height curve (no change)

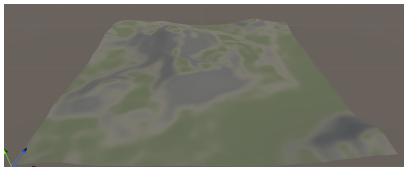

Figure 17: exponential curve (higher values are increased)

Figure 18: custom curve that introduces terraces

itself. Figures below demonstrate some of the the shapes achievable with this (all generations use the same values apart from animation curve evaluation).

These attributes allow the user to generate a huge variety of terrains to generate upon which the rest of generator is based, Perlin noise was chosen as it allows for good variation between outputs and user control over specific details, a few other benefits are that the same result can be achieved given the same inputs allowing users to focus on the other aspects of city generation (roads, buildings) while retaining an achieved terrain shape.

The generated height map is stored in a 2D float array which is then passed to the Mesh and texture generators for display. The Mesh generator generates a vertex position for each height map value, offset from one another by a set amount (in this generator we use 1 unit of the unity world space). The Mesh is stored in the MeshData class which simply holds all the vertices and triangles that form the mesh this is passed to the MapDisplay class which draws the mesh. This system has one major drawback which is that Unity has a limit for vertices in a mesh at 65535 this means that only meshes of a certain size can be produced, various solutions exist mainly the use of multiple meshes to represent chunks for larger terrains. Given the controls available in the rest of the city generator even with the limited mesh size a variety of cities that are still able to be generated so it was decided that this method would not be worth pursuing given time constraints and need to move on to other systems, although it exists as a good route for further extensions of the project.

The Perlin noise algorithm is very performative which allows for the real-time generation of terrains and updating terrains based on parameter changes this is extremely useful for users when trying to make incremental changes to achieve a desired result, the only exception being when the number of octaves is dramatically increased, it was found on our hardware that above 10 octaves lead to dramatically decreased performance while providing exponentially less detail for each octave this is as every vertex position has to be iterated over for each octave because of this it was decided to limit the octave number the end user can use.

2.2.2 Roads

The road generation is the most complex system of the city generator as as discussed above they are the main factor in forming the shape and form of the city. The generator consists of 5 main components required for operation, the city generator controller, the road generator which hosts the L-system used for generation, Road Section a helper class for the road generator to hold potential road sections for evaluation, Production rules which can be configured to change factors of generation and the road graph which stores the result of the generation :

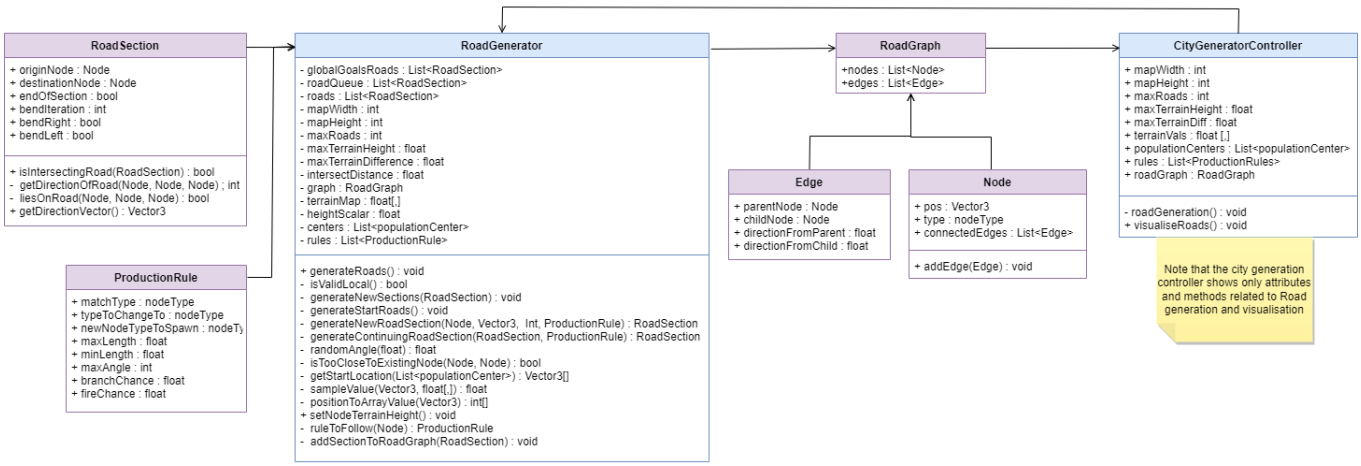

Figure 19: Class diagram for the road generation system

Before describing the L-system generation the surrounding data structures must first be explored :

- Road Section - The road section is a temporary holder for potential roads storing the origin and destination positions, whether the road should carry on generating (endOfSection) whether the road is bending left or right and how many road sections before it have bent. it also contains some helper methods for some basic operations : an intersecting check and getting the direction of the road section.

- Production Rule - Production rules are the rules that the L-system will apply when generating the roads, for this it was decided to use a Unity Scriptable object, these are data containers that can be created in the Unity Editor to host whatever values the user wishes. This allows a vast degree of user control as any number of production rules can be created and fed to the system as well as allowing for the quick editing of rule values for fine tuning of road generation.

- RoadGraph - The Road graph is the final output of the road generator storing all the nodes and edges between them, A graph was used as as explored in Procedural generation of road networks using L-Systems by Martin Jormedal [**17**] it allows for cyclic(connected) structures to be generated from the L-system (which normally out a string of instructions for visualisation leading to linear results) leading to much more realistic and believable results.

2.2.3 Exploring the L-system

The L-system uses a number of attributes for generation below is a brief description of the purpose of each:

- globalGoalsRoads - this is list of new road sections generated by the L-system from the previous iteration.
- roadQueue - each iteration the roads from the globalGoalsRoads are added to the roadQueue for verification.
- roads - list of roads that have been validated and can be added to the road graph.
- mapWidth and mapHeight - the bounds of the generation.
- maxRoads - maximum number of roads the system should generate.
- maxTerrainHeight - the max height roads can be generated up to on the terrain.
- maxTerrainDifference - the max difference between two terrain heights a road can span.
- intersectDistance - the distance that generated road sections can be from each other.
- graph - the final road graph.
- terrainMap - terrain values supplied by the terrain generator.
- heightScalar - the height scale of the terrain, needed for road placement on the terrain as the L-system only generates roads in 2 dimensions for simplicity.
- centers - a list of the population centers supplied by the user.
- rules - a list of production rules supplied by the user.

The L-system will begin with the initial road sections generated from the population centres in the road queue, next the system will check for each road if it is valid for local constraints, local constraints include : if the road section is too close to an existing node, if the road section intersects and existing road section, if the road section is within map boundaries, if the road section is within a valid terrain height, if the road section is too steep, if the road folds in on itself, if the road section nodes already have too many connected nodes and to make sure there is not an already existing connection between the two nodes. If all these tests are valid the road section is added to the road graph and the next continuing section is generated from the node. The L-system will stop generating when either the road queue is empty (either all the road sections are an end of section or no valid sections can be generated) or when the road limit is reached. Below is a flow chart to show this process.

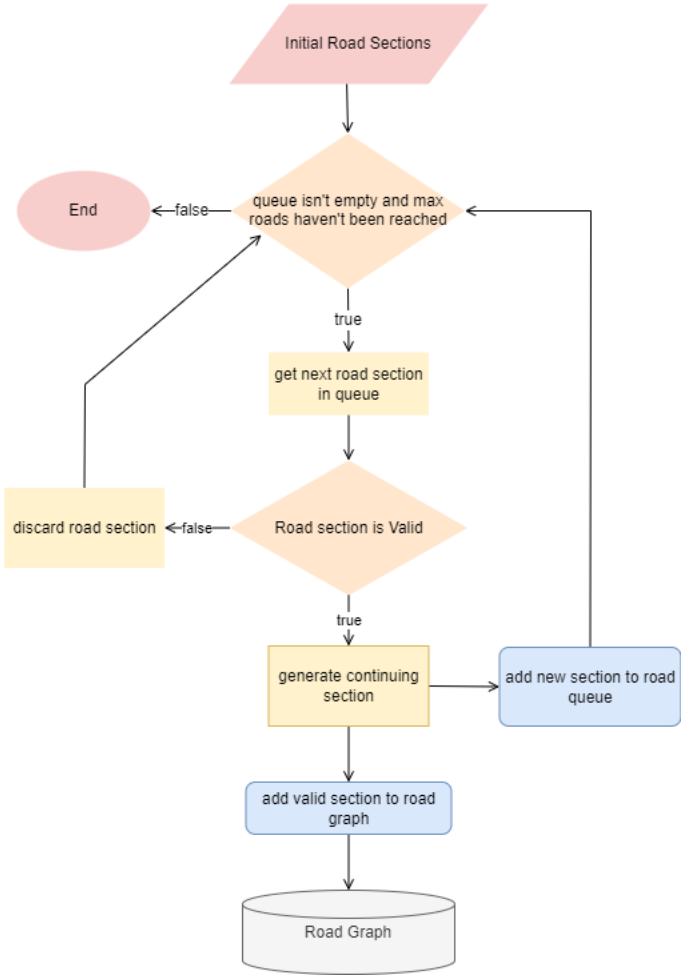

Figure 20: flow chart showing basic road section generation process

Production Rules while using random properties to decide generation, if the L-system had static ranges for generation parameters we would expect similar looking results each time despite not being identical to each other. This is something the project aims to avoid, a large variety of outputs should be possible while still remaining user friendly and accessible, it should not require the user to change the behaviour of the L-system itself just the parameters used. In search of this result we turn to the method used in Procedural generation of road networks using L-Systems by Martin Jormedal, This approach gives each road node a type (motorway, main, side) and a set of rules, each rule has a node type to match to and a set of parameters to supply to the system for generation of continuing sections. Unity scriptable objects were used to implement these allowing the user to create and set the parameters for each rule, the customisable parameters are : The node type to match to, the type of the new node, the maximum and minimum length, the max angle that the new road section can generate (from the current direction of the node), the chance that the road node will branch and the fire chance of the rule. These were chosen as they allow for good levels of user customisation while still allowing the system to generate coherent road systems. In the below example the L-system was given one production rule : maxLength = 3, minLength = 2, Branch chance = 2. The angle was changed between each to demonstrate how even one parameter with one rule can lead to different results :

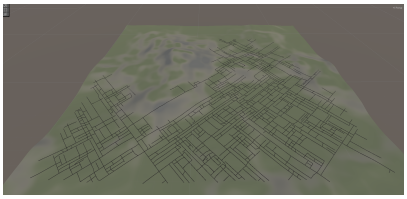

Figure 21: Max angle set to 0, the roads continue straight and branch perpendicular

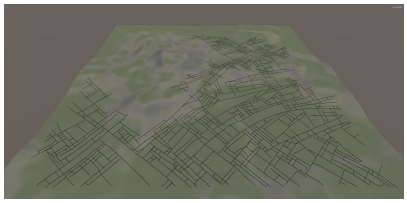

Figure 22: Max angle set to 90, the roads can generate in any angle between -90 and 90

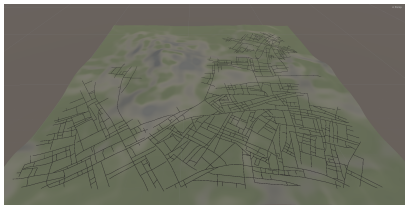

Figure 23: Max angle set 180, the roads can generate in any direction

Combining rules with various levels of road hierarchy (main, side etc), lengths and maximum angles allows for the the generation of more complex road layouts and is very extensible allowing the user to quickly adjust rules to achieve the desired goal, below an example of a road generation combining 3 rules :

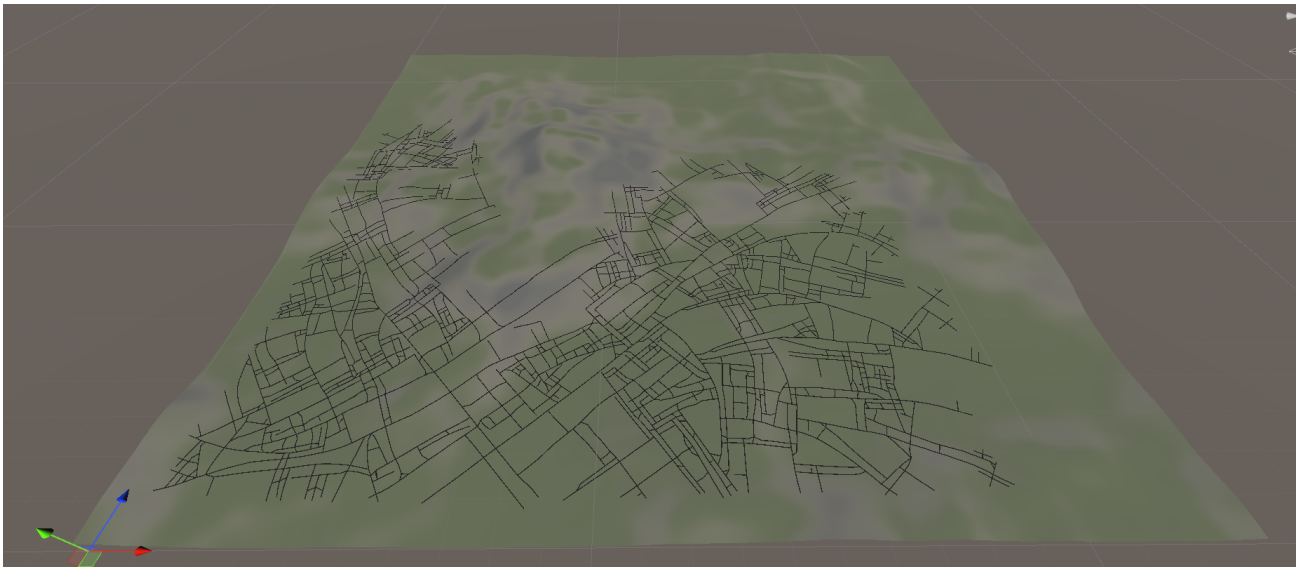

Figure 24: Road system generated using 3 production rules

Road bends bends in roads are an important part of the realism and believability of a road network initial prototypes proved to be flawed when it came to changing directions as each road section could alternatively go back and forth between left and right angles (relative to the origin direction) leading to jagged and artificial looking roads. The solution to this was to introduce road bends to road sections, each road section tracks which direction it should be bending (left or right) and also at which point in the bend they are (bend iteration). If the section has reached the bend iteration limit it will select a new direction to bend and generate a new section with a bend iteration of zero, if the section bend iteration is below the limit it will follow the current bend direction and the new section generated will increment the bend iteration. This leads to extended bends across road sections which allow for much more realistic generation of streets and also the ability for the user to control such bends below is an example:

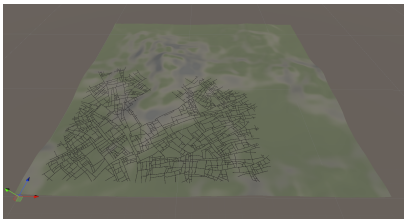

Figure 25: Number of sections in bend : 1, every section the bend direction is re-evaluated

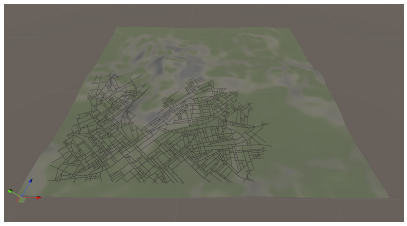

Figure 26: Number of sections in bend : 10, every 10 sections the bend is re-evaluated

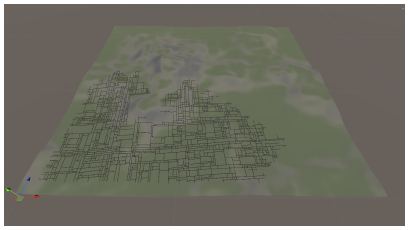

Figure 27: Number of sections in bend : 50, every 50 sections the bend is re-evaluated

2.2.4 Population Centers

In the real world population centers exist over large geographic areas and are related to the surrounding areas, here we hijack the language slightly to represent where a concentration of people exist in our cities, these areas become the CBDs (Central business districts) of our city as in reality these areas generally have larger buildings, The main purpose of population centers in our generation is to vary the height of generated buildings to represent a skyline and increase the believability of the output. For the population we use a float array similar to the terrain map, values of population are mapped between 0 and 1 based on user supplied population centers.

Population centers use Unity scriptable objects, so similarly to the production rules users can add and adjust them quickly and easily, allowing for fine adjustments to the final output and to get closer to the desired result. Population centers contain the following attributes :

- Location : where on the map the population centre is
- Population Value : the "strength" of the population dictates the height of generated buildings and how far the population centre has influence
- falloff Values : how quickly the population values decreases from the centre.

When generating the population map the algorithm bases the population value off the distance to the population and the falloff values, as well as this the user can define a minimum population, if the value falls below this value it is automatically set to zero, below are examples of different values being used for the population map :

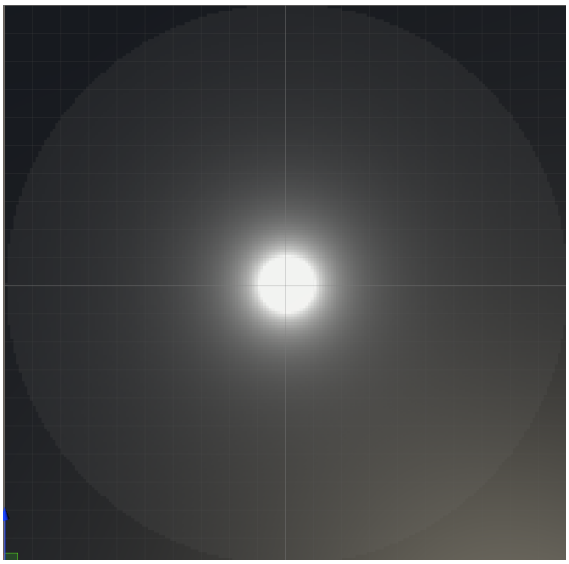

Figure 28: population and falloff value of 10, minimum population value = 1

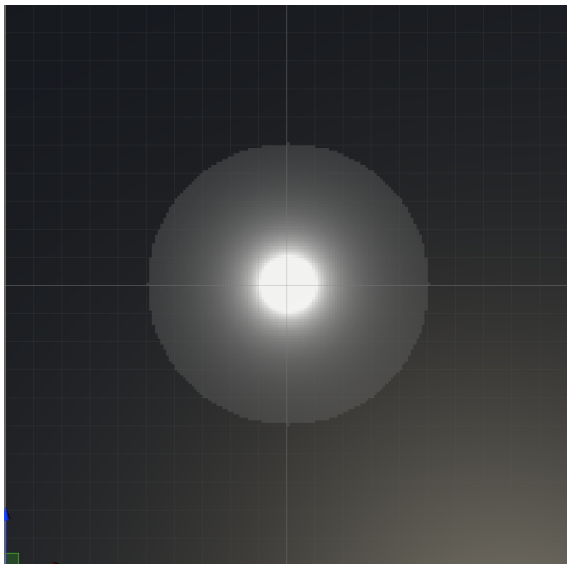

Figure 29: population of 5, falloff of 10, minimum population = 1

We can combine these population centers to form where the city will generate the highest buildings and create a smooth transition of building from "high" population values to lower ones, the figures below show how population centers can be manipulated together :

Figure 30: The combination of two population centers : population centre 1 (bottom left) has a position of (50, 50) a population value of 5 and a fallof of 10, population centre 2 (top right) has a position of (150, 150) a population value of 10 and a falloff of 10

We also introduce a global minimum population value for the user to adjust, if a population value falls below this value the population value is set to zero allowing for more customisable population falloffs; below is an example of different minimum population values (for the following figures population centre 1 (bottom left) has a position of (50, 50) a population value of 5 and a fallof of 10, population centre 2 (top right) has a position of (150, 150) a population value of 10 and a falloff of 10):

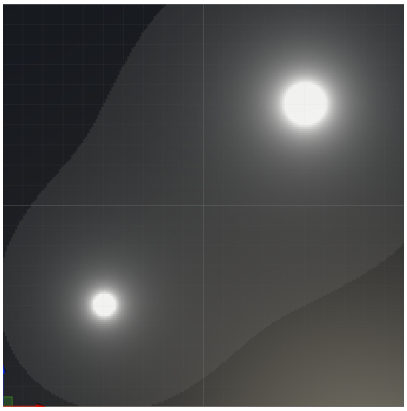

Figure 31: minimum population of 1.5

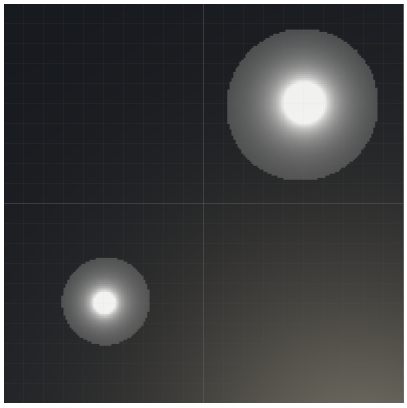

Figure 32: minimum population of 3

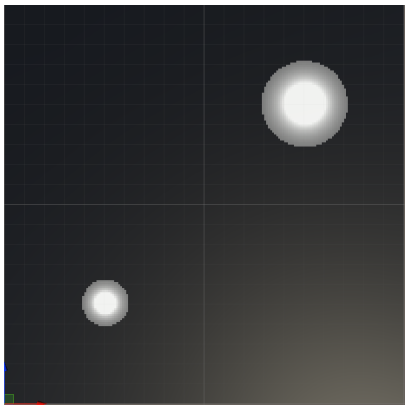

Figure 33: minimum population of 5

2.2.5 Buildings

When originally planning the project, it was aimed to build a L-system that would generate unique shapes and structures for each individual building, during prototyping of the building system it was found that : Complexity quickly became an issue, controls over user generation of buildings lead to difficulties, as it depended heavily on the population and terrain of the road on which it was placed leading to parameters feeling somewhat meaningless to adjust and it was hard to find any quantifiable difference between buildings generated with different parameters. The main problem was ballooning generation time, generation of buildings was timely at small sizes but when attempting to generate larger cities generation was taking a very long time, this is an issue as a goal of this project is to allow user iteration of cities to achieve a desired result, not being able to make small adjustments to generation parameters without having to wait a long time for the output is a poor user experience and will lead to a worse overall product so another approach was taken

The Approach this project took is much simpler than a L-system approach, for each road we generate an allotment either side, which consists of 4 vertices which are randomly offset from the initial positions (by a small amount).The allotments are first checked to make sure they are valid, they are checked for : If the allotment is intersecting with a road; If the allotment is intersecting with another allotment. Once all allotments are generated and validated They are then passed to a mesh generator which adds the top vertices (with random height variations) based on the population value that the allotment has. This leads

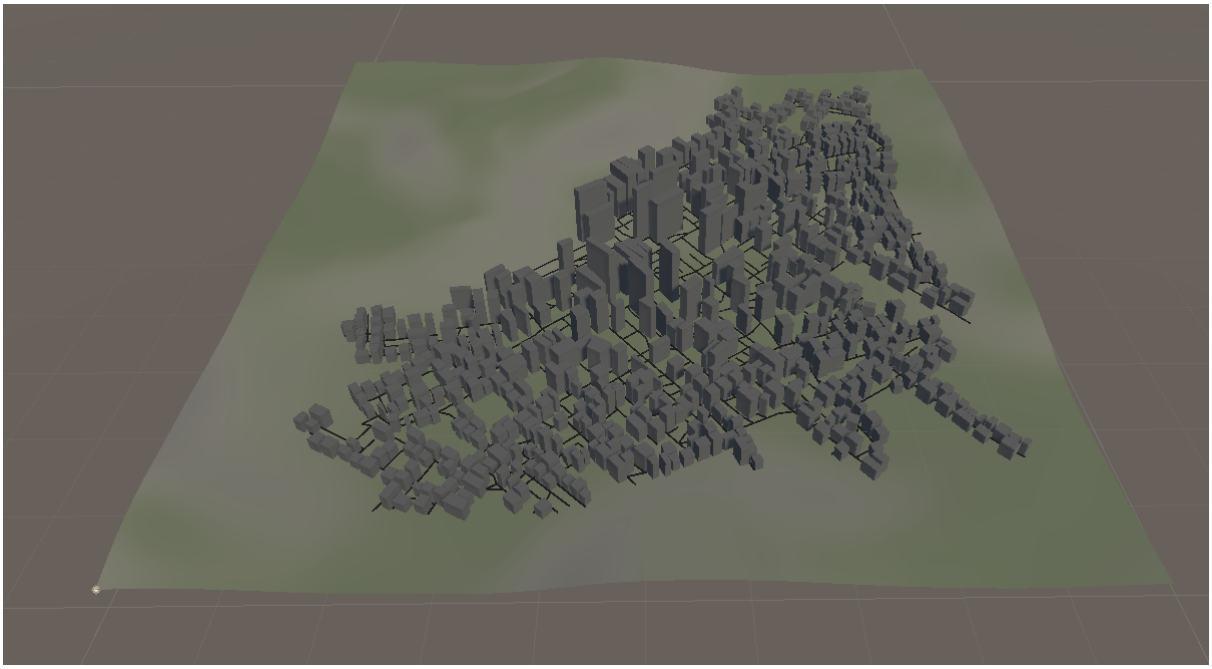

Figure 34: example building generation

to larger buildings near population centres and a smooth gradient of building heights down as the population centre gets further away, this produces more believable city scapes

3 Testing

Testing of a procedural system presents a few issues that need discussion:

- Variability : the nature of the systems generation means that each can vary greatly from generation to generation meaning it is impossible to exhaustively test all scenarios (especially given the number of controllable parameters), it can be hard to test edge cases as there are so many parameters interacting with one another for generation.
- Non-deterministic : The system uses a variety of random generators to produce the final output meaning it's hard to create test cases with a predicted outcome to show success or failure.
- Subjectivity : As this project is based on a visual output there may be different beliefs about whether a component fulfils the required believability and realism this project aims for

3.1 Strategy

The main problem recognized during the planning of the project was that it is difficult to create tests for the city generation as a whole due to the random nature of the output and program, baked into the design and functionality of the system is the generation of a varied output. This makes it hard to create pass/fail conditions for a test (excluding the program crashing or no generation which is only of limited value when testing). Instead we focus on the systems behind the generation separately for testing rather than trying to create a success criteria for the believability of the city as a whole. This helps with another problem which is the number of user controllable parameters and their interaction with one another, by splitting the components up to test first and then focussing on the parameters that are passed between them it is more likely that we'll be able to understand where potential issues exist. We split the system into 4 main components of testing Terrain generation, road generation, building generation and population map creation. Given the problems explained above it was decided to use a white box testing approach.

Regression testing was explored initially but flaws were found. Regression testing involves rerunning tests when updates are made to ensure system functionality and make sure it hasn't *regressed* to a state that's worse than before the update. this is good for creating and maintaining stability within the system and preserves any previous work from being broken by continuing components. It was decided not to use this method as it requires the maintenance of tests for components throughout the development process, as we are using a sequential approach to component implementation and we have a limited time frame to work within it was decided that the work of rewriting tests to fit small additions to previous components when needed by continuing components was not conducive to finding problems or worth the effort of test re-writing when needed. A further problem is test coverage, due to the number of parameters used for generation and complexity of some parts of the system it would be hard to create enough tests to cover all the needed parameter changes and variations. For these reasons another approach was chosen.

For the system components we tested using the exploratory testing method, due to the difficulty of creating a scripted test that can cover all possible parameter values and judge if the output fulfils the test criteria. Using exploratory testing allowed for the testing of each component and experiment with where the parameter ranges should fall, it also allowed for the catching of bugs earlier for each component reducing the chance that a following component would expose a bug and require time to fix. The exploratory testing method also works well with our choice of planning method (spiral) given that requirements for each component could change during the creation of a prior one given problems or implementation plan changes, exploratory testing allow us to focus on the areas of the program that have changed and find any potential issues quickly. It also allows us to test from a user perspective which makes it easier to find problems that affect the normal use of the system, as while edge cases are important to address and fix when they occur, with the complexity and number of parameters it would be hard to find and fix every single case that could occur. While exploratory testing approaches are often seen as black box testing here the testing exposes problems with generation so fixing components behaviour requires adjusting not only the parameter ranges but the effect the parameter has on the system which requires knowledge of how the system operates "under the hood".

4 Evaluation and Insight

As with any product based development cycle, elements of the city generation either surpassed or fell under expectations and progress on components was not as consistent as plans assumed when created. While background/initial planning for the project was (approach, scale, detail) decided early in planning and development, constraints quickly arose during the development of each component. Progress was also hindered by the need to learn/experiment with techniques that are unfamiliar which unfortunately reduced the scope of the project slightly. Despite these setbacks the project achieves the goal of producing cities that could be seen as believable and the user control of the system is extensible.

This section evaluates the approach, design, implementation, testing and final output of the city generator

4.1 Development approach

While this report is separated into background, design and implementation, and testing this was not the approach followed for actual development, as discussed in section 1.4 this project uses the spiral approach. The spiral approach separates these sections into each design iteration which is useful for allowing continuous development of components without trying to pre-plan for results that may ultimately be changed by research and implementation when the next component is developed, thus saving time overall on the development of each component. Continuing components are based on the result of previous components which allows for the adjustment of scope and features based on what both previous component functionality and time allows.

For each component there was a:

- A planning and background stage where previous work was examined, decisions about the overall scope and features of the component were decided and the best method to approach the problem with was settled upon.
- Designing the system and prototyping stage, during this stage the overall architecture of the system was planned out and any potential systems that might cause later problems were prototyped and tested.
- Testing, each component was tested at the end of development for bugs, edge cases and interaction with other components.
- Evaluation, the evaluation step was important as it allows for the scope of continuing components to be decided and evaluated, this step would generally feed back into the planning and development stage of the process for the next developed component.

This aligned well with the project goals as success of components depended on scope and implementation of previous sections, it also allowed for mistakes to be made during the prototyping/planning section without affecting the overall development timeline, an important combination when initial methods investigated were unfamiliar and needed to be learnt (the main example being L-systems) as well as implemented.

The allowance for flexibility throughout the project was hugely useful and helped to temper the scope of the original plans for each component as they were created, they could be properly evaluated given previous components and goals for future development. Although the trade-off for this flexibility meant that time-limits for each component had to be set. This was difficult because each component individually allows for large amounts of expansion to create more and more detailed and realistic city scapes, the time frame allotted requires that compromises must be made in the detail and realism of components.

The spiral approach also allows for good risk management, as the second component developed the roads were a challenge. Despite research into the accepted methods and best practices learning and implementing L-systems proved difficult and unwieldy at first. Due to the importance of roads to the overall generation it was decided to reduce the length of time spent on buildings and instead dedicate it to creating a better road system. This was not as much of a roadblock as it would've been with a plan driven approach as the in depth planning for building generation had not yet been done and allowed for the scaling back of scope from the original project plan while still staying on schedule to deliver the final product.

Testing is also streamlined with the spiral method. It allowed for the testing of each component separately through development meaning the focus of final system testing could be on the integration of all these components knowing that they were unlikely to produce problems outside of the areas that involved inter-component communication. It also allowed for development of continuing components knowing that the prior was ready to "ship" this reduced the risk that overrunning on one component would compromise the chance of delivering a functional product.

There are some challenges that come with the spiral approach. The largest that had to be dealt with was the uncertainty in time frames with this approach, early planning and background investigations lead to an initial schedule but as components were more deeply investigated and analysed throughout development, time frames for development had to be refined and scope adjusted to make sure the final product was completed. It can also lead to scope creep in components, if more time was available for the development of a component (in this project terrain was completed ahead of schedule) it lead to seeing if more features could be implemented to increase the realism and believability of the city in reality the best decision would've been to move on to the next component early once goals of the previous had been reached. This lead to some wasted work as features not investigated and planned properly in the initial phase of the planning were attempted and not able to be implemented fully by the deadline and had to be disregarded. The terrain as an example was finished about 2 weeks before the imposed deadline for the component, because of this attempts were made to implement a erosion simulation to make the terrain more realistic based on the paper by Xing Mei, Philippe Decaudin and Bao-Gang Hu, Fast Hydraulic Erosion Simulation and Visualization on GPU [**30**]. However this was not able to be implemented by the deadline for terrain which lead to 2 weeks of wasted work on a feature that didn't make it into the final product and would've been better spent on the planning and analysis of the road system.

In conclusion the spiral method proved to be an effective approach for this project due to the unknown factors going into development and skills of the author, it allowed flexibility in the planning and implementation of each component allowing for the delivery of a functioning product that fulfils the goals set out.

4.2 Design

The goals of this project were to generate a believable city with a focus on user control of generation, for user control we aim to give as much control over parameters as possible while still maintaining a system that generates coherent output. This proved challenging for a procedural systems as they are by design random so allowing user control was a challenge.

One of the main challenges to address was the modularity of the system. It was found early on in the background and analysis of the problem that a modular approach to each component of the system was the ideal approach as it would allow for the user to control each section individually during generation. This allows users to focus on the parameters important to each component and achieve the desired goal before moving on to the next. This approach also allows for some extensibility of the system generation, components can be added to the current generation stack to affect the final output. Although this modularity comes with the trade-off that generation components have limited interaction with one another during generation, there are the core data-structures required for generation (terrain map, population map) but better results may be achievable through generation in parallel and with greater communication between components.

The problem of User control was hard to overcome, initial plans allowed for the user to have limitless control over the parameters but this proved too problematic when it came to producing coherent results consistently this meant that parameter ranges had to be built into the design of components (these were decided during component testing), this is an unfortunate compromise that had to be made but was required for proper generation of cities and improving the user experience. Ultimately the introduction of parameter ranges for components was an overall benefit to the final product as it leads to an easier experience when generating cities.

Overall the design of the system architecture worked well for the final product, especially for user control of the generation, although this decision ended up costing the believability and realism of the output as the ability of the user to dictate how the system can generate means that there is a reliance on them to understand how parameters affect the output. Other approaches to city generation [**5**] and [**17**] produce in my view more believable results for cities but don't allow as much user control over that generation.

4.3 Testing

The explorative approach to testing worked well for this project as each component in the spiral model had a dedicated time to allow testing this meant that appropriate parameter values and ranges could be nailed down for the module being tested and how those parameter values would interact with the rest of system. It fit very well into the development approach taken as it allowed for flexible and user focused testing of each component and suit the single developer approach as findings didn't have to be shared with a team and fixes could be implemented almost instantly

Unfortunately while the approach worked well for individual modules and testing of them from a user perspective some more rigorous and methodical testing was necessary for the final generation. The the challenge of implementing Unit testing given the random output inherent to the system and size of the system late in development would've been time

consuming and potentially not of value if no bugs of significant influence could be found.

The lack of formalization of tests across the components also lead to difficulties when it came to testing different components, this could also be seen as a benefit as it allows for the tester to use their knowledge of the system to create appropriate tests but not having a formalised way of testing meant that the test section of each spiral iteration was extended as different use cases were explored, this may of been reduced if a set criteria of test goals were set in initial planning of the project

As a problem of both the system's randomness and the approach taken to testing, tests results are almost impossible to reproduce when it comes to generation, unless bumping up against values that the system cannot generate from. This limits the testing we could do to simply testing if the functionality of the system works and testing ranges for generation parameters.

Overall I believe this was a valid solution of testing but the constraints imposed by the randomness made it hard to test the system generation as a whole reliably despite this I believe overall the testing was a success although further research into testing methods for procedural systems may provide a better answer to this conundrum.

4.4 Tools

Which tool to use was a primary consideration in the initial planning of the project as objectives were fairly complex and not something many tools can produce on their own without significant retooling, it was required the program would be able to graphically represent the city and allow for large amounts of user control through it which narrowed down options considerably. The main tool decided was Unity which has some benefits and trade-offs of use discovered throughout the development of this project but overall was a successful choice given the skillset of the author and the requirements of the project

The Ease of use was a large draw when it came to deciding tool use for this project it's gameObject and component system are useful in generating modular systems that allow for a degree of customisation, it also provides a built in render pipeline which removed some of the challenges of the display of city and allowed more focus on the generation system. This ease of use is not at the detriment of extensibility as it stills allows for deep customisation of components which is a very useful mix for this project

Flexibility, the component system mentioned allows for a huge amount of flexibility when generating cities being able to slot components into game objects allowed for rapid testing of the systems with one another and supports the modular approach we aim for.

Unity has extensive documentation on every system as well as user forums and tutorials. This made it easy to find resources for problems and developers that had experienced similar issues in the past, this wealth of documentation meant that Unity generally didn't impede development rate even when working with unfamiliar systems.

In retrospective I believe that while unity can create great results, if my knowledge of python was strong enough Blender would be the slightly better option to use for this sort of challenge it's geometry and texture node system could be manipulated to create a system that generates a realistic city with proper texturing for the buildings (something not attempted in this project due to time constraints). However I think Unity overall provides a better user experience for adjusting generation parameters and allowing for a more modular

design. If the reader is considering a similar project it is suggested that an in depth analysis of both options would be appropriate to find how each could benefit the project.

4.5 Final Result

The final product of this project achieves most of the original goals of the project, the generated cities do resemble real built environments and conform to the terrain they lie on leading to what I consider Believable city scapes. The main original goal not implemented was a L-system to generate the buildings, while the variety of buildings in the final product is lacking the exclusion of this feature has not lead to cities that aren't believable in form and shape. The L-system implementation was more successful than envisioned after some initial stumbles in design. As it makes up the back bone of the city it's implementation success means that a huge variety of cities can be generated based on the parameters supplied making for a very extensible system. As mentioned above the building generation is not as good as was originally hoped but the time taken from this component and used on the L-system meant a stronger implementation of the more important component hence it was seen as a net benefit to the result at the expense of some realism and believability that a well implemented building generator would supply The success of the final product can be best decided by comparing to other procedural city generators, the best example being CityEngine originally built and proposed by Yoav I H Parish and Pascal Müller, Procedural Modeling of Cities. While their generation produces cities that are both more believable and more realistic (a slightly unfair comparison given the scope and time frame of the project) than this project can produce, the level of user control the final product allows, means that while the final result may be of a lower quality users have a huge amount of control as to how the city ends up looking.

4.6 Future Expansions

There are a vast amount of things that could be extended upon in this project given that cities are hugely complex one could devote an entire life to the procedural generation of cities and still have more to add to increase the believability and realism of a cityscape, given this below are some of the main threads of extension that I would further develop if time allowed:

- Advanced building generation - as mentioned above an L-system generator for buildings had to be cut from development due to time constraints but I believe that this could provide a huge boost in the believability and realism of the city and would be worth pursuing as an extension of this project
- Larger Terrains - It was chosen in this project to limit terrain generation to a 255x255 map so it fell below the vertex limit enforced by unity. Extensions were investigated but not pursued, multiple meshes using a set chunk size to allow for infinite terrain generation would prove a useful extension on the city.
- Textured Buildings - Buildings in our system are all one color, methods exist for the procedural texturing of meshes and i believe properly implemented could vastly improve

the believability and realism of the system especially when paired with a building L-system

- UI improvements - while Unity provides a very readable UI that is quite user friendly creating a custom UI to separate the components parameters from one another would provide a better UX and allow for easier generation of cities

4.7 Conclusion

In retrospective of the project as a whole there are a few areas I would approach differently given the chance to do it again.

Increased spike work/prototyping of systems, a few roadblocks occurred during the original development of the L-system which lead to having to readjust the schedule of component development, if more time had been spent analysing the requirements of road systems in the real world and previous work on procedural cities I believe this could've been avoided.

Increased focus on UX, a few times I found that UI work was neglected in favour of working on more functionality for the system. This was a hard act to balance as while the system was complex and required lots of work and tuning, this work would be rendered potentially useless by poor UI design that makes it hard to interact with said system. If I were to start again I would conduct a much deeper analysis and planning of UI at the start of the project to make sure UI design remains a focus throughout the projects life cycle.

In conclusion I believe the project was an overall success, users have a large range of parameters to generate cities from and the results (in my opinion) look good and reflect real life cities well enough for the scope of the project. This project has greatly increased my understanding of both L-systems and procedural generation as a whole and some of the pitfalls that come with the method, as well as the importance of proper planning throughout the lifecycle of a project.

A Third-Party Code and Libraries

A.1 Libraries

Unity Scripting API - allows use of all unity features used in this project, for this project Unity version 2021.3.33f1 Personal edition was used

B Ethics submission

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
rhj44@aber.ac.uk

Full Name
Rhys Johnson

Please enter the name of the person responsible for reviewing your assessment
Fred Labrosse

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
ffl@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module Code
CS39440

Proposed Study Title
Procedural Generation of Cityscapes

Proposed Start Date
January 2024

Proposed Completion Date
May 2024

Are you conducting a quantitative or qualitative research project?
Mixed methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants
No

Institute
IMPACS

Please provide a brief summary of your project (150 words max)
A system that allows users to procedurally generate believable city scapes with an emphasis on user control over generation parameters

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Not applicable

Will the appropriate measures be put in place for the secure and confidential storage of data
Not applicable

Does this research pose more than minimal and predictable risk to the researcher
No

References

[1] https://www.massivesoftware.com/index.html - The massive engine originally developed for Peter Jackson's The Lord Of The Rings films is a tool for the procedural generation of crowds used widely across the film and entertainment industry.

[2] Petrisor LE, Petrisor AI (2001), Why Are American and European Cities Different? A Legislative Approach, The Student Diplomat 7(2):6-9 - This paper explores the difference in city shapes and road layout between Bucharest and Columbia, South Carolina and how differences in legislation can impact cities.

[3] James A. Ferwerda, Three Varieties of Realism in Computer Graphics. - this paper discusses types of realism in computer graphics what they are and the criteria to achieve them.

[4] Great Britain. Ministry of War Transport. Committee on Design and Layout of Roads in Built-up Areas, Pages 22-25 - these pages discuss the various road layouts of towns/cities across the world, while the paper focuses on traffic implications of various road layouts and improvements it is useful in understanding the different road patterns that exist.

[5] Yoav I H Parish, Pascal Müller, Procedural Modeling of Cities - One of the definitive papers on city generation describing not just road generation but adaption to various maps and the modeling of buildings and texturing of those buildings.

[6] Francis Henry Wollaston Sheppard, London: A History, Pages 8 - 11 - these pages describe the origin of the city of London, the Roman arrival in the region and the potential of pre-existing iron age occupation of the region.

[7] Maarten Bosker, Regional Science and Urban Economics - this paper talks about factors affecting the development of cities in detail as well as why locations are picked for cities both for long term developments and planned cities.

[8] Jonathan Clarke, 20th-Century Coal and Oil-Fired Electric Power Generation - this paper explores the development and history of power generation in the UK and the movement from coal and oil to other technologies

[9] The Wildlife Trusts, What's The Damage? - this report explores the environmental impact of the proposed High speed rail route in the UK, the sites that would be affected and what could be lost.

[10] https://www.blender.org/about/ - website for Blender a 3D modelling tool for information about features and showcases of previous projects

[11] https://blendermarket.com/ - website marketplace hosting thousands of user made addons for Blender

[12] https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/introduction.html - introduction to the geometry node system in Blender

[13] https://unity.com/case-study - a huge list of projects that use unity in a variety of industries

[14] https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview - overview of the CityEngine a tool for the procedural generation of cities

[15] https://www.sidefx.com/products/houdini/ - website for Houdini a 3D modelling tool, contains information about features of the program

[16] https://www.world-machine.com/features.php - website for WorldMachine a tool for generating procedural terrain for an array of uses.

[17] Martin Jormedal, Procedural generation of road networks using L-Systems - this paper explores the generation of road systems for cities exploring problems to be overcome and results from the algorithm.

[18] Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, Radomír Měch, L-Systems: From the theory to visual models of plants - this paper explores the use of L-systems for the modelling of plant growth and how various such as insects and pruning affect their development

[19] Werner Heisenberg - The actual content of Quantum Theoretical kinematics and mechanics - this paper talks about the position and velocity of electrons particles and how observation of a electrons positions means less certainty in its velocity and vice-versa.

[20] https://www.gridbugs.org/wave-function-collapse/ - webpage offering a an introduction to what a wave function collapse algorithm does and how it can be implemented on a basic level.

[21] Bo Lin, Wassim Jabi, Rongdan Diao, Urban Space Simulation Based on Wave Function Collapse and Convolutional Neural Network - this paper explores road generation with a wave function collapse algorithm

[22] https://marian42.de/article/wfc/ , Article on the creation of a procedural infinite city generation utilising wave function collapse by Marian exploring common problems and solutions.

[23] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, Eugene Zhang, Interactive Procedural Street Modelling - this paper explores the use of tensor fields for generating road systems on a large scale with a focus on allowing users to tweak results to their liking

[24] Ken Perlin, An Image Synthesizer - This paper explores the development of natural looking textures for use in films and games

[25] Ondřej Šťava, Bedřich Beneš, Matthew Brisbin, Jaroslav Křivánek, Interactive Terrain Modeling Using Hydraulic Erosion - this paper explores the implementation of simulating rainfall and other flowing water to create realistic terrain

[26] https://www.redblobgames.com/maps/terrain-from-noise/ - this article contains not only the basics of terrain generation using perlin noise but a variety of extensions from the basic algorithm that can be used to get differing terrain.

[27] https://heredragonsabound.blogspot.com/2019/02/perlin-noise-procedural-content.html - blog post exploring the flaws or perlin noise for procedural generation and the need to dive deeper into what makes a space interesting and coherent to create a procedural system that makes interesting results.

[28] https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html - introduction to animation curves in unity how they can be used and the various values that can be affected

[29] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, Luc Van Gool, Procedural Modeling of Buildings - A comprehensive look at the procedural generation of buildings using L-systems

[30] Xing Mei, Philippe Decaudin, Bao-Gang Hu, Fast Hydraulic Erosion Simulation and Visualization on GPU - This paper explores the implementation of simulation of rainfall and water flow over terrain and the erosion it causes.