Procedural City Generation

Luke Tolchard
Luke2.Tolchard@live.uwe.ac.uk
Supervisor: Sean Butler

Department of Computer Science and Creative Technology
University of the West of England
Coldharbour Lane
Bristol BS16 1QY

Fig 1: An example output from this procedural city generator

Abstract

This report outlines the implementation of a procedural city generator within the Unity engine, with the ideal outcome facilitating the automatic reading of a given height map and terrain for detailed generation in a user-defined environment. In practice, the project encountered difficulties that prevented it being realised to this standard, however it still has the power to create a highly customizable and dynamic environment to be used as a stencil for further development in the future.

Keywords: Procedural Generation, Perlin Noise, Terrain, Software Design, Urban Development, Architecture, Population Map, Water Map, Intersection, Mesh, Vertex, UI, Menu

Biography

This project has been an exploration into the world of procedurally generated content, and how it can be used to assist the software design process. I am also extremely interested in envrionmental design so this project perfectly encompasses my interests and it is my hope that projects like this will help me increase my understanding of this aspect of software development and help me pursue a relevant role within the industry.

How to access the project

Showcase video - https://www.youtube.com/watch?v=H1LDzN-9Fhc&feature=youtu.be

1. Introduction

There are many barriers to creating fresh and innovative video games. One of these barriers is that developers simply don't have enough time to create unique and impactful assets to use within their environment, which detracts from the experience. A quick fix to this problem is to outsource to artists, or to purchase generic assets online, but this is often an unpopular solution as a game constructed with assets avaialable to anyone could risk feeling unoriginal or not perfectly suited for their purpose (Tanya Short and Tarn Adams, 2017).

Procedurally content generation (PCG) is therefore a near-perfect solution by allowing developers to create unique assets in a fraction of the time it would take to make them by hand. Particularly with rouge-like games such as 'Rouge' (Michael Toy, 1980) use PCG as a unique selling point to give a game replayability(Khalifa et al, 2016). Variety is an excellent thing for players, and many players enjoy the sensation of stumbling across something new and using their intelligence and skills to figure their way through it, which is a huge part of someone playing a game or experiencing something for the first time. By using procedural generation, the player can have that new game experience every time they start up a game they've already been playing for hours, days, or months.

In the summary, the project objectives were as follows:

- To create an easy to use tool to facilitate procedural city generation
- To allow the generator settings to be fully customizable to support a developers needs
- To make the tool easy to import into existing projects, and also make the output of the generator easily to export as .obj
- To have the generator automatically read height maps to allow custom terrains
- The impose barriers to generation and ways to circumnavigate them as a cost/benefit analysis

In the context of this project, the ability to generate entire 3D environments in a matter of minutes such as with a procedural city generator yields a number of uses, particularly for simulator games, as well as a whole range of non-gaming related industries such as urban environment planning and military training. This project set out to create a tool that offered a developer a customisable and flexible way of generating a city according to their needs, or at the very least provide a stencil for manual design. There are several design features within the tool to help it realise this goal, and to be as user friendly as possible.

2. Practice

2.1. Perlin population and water maps

To begin generating a city, the area in which the generator could spread to needed to be defined and suitably restricted. The chosen method of generation would be to use a Perlin noise map as a population defining map, due to the idea that clumps of high noise and low noise would translate well into creating definable districts within the city style. Areas of high noise could be used to generate larger buildings and skyscrapers, and areas of low noise could be sparsely populated with smaller suburban houses which would return a more realistic looking generation. As well as referring to a population map, it was decided that the generator could also look at another texture that represents areas of water, or other untraversable terrain such as mountain ranges. In order to use this feature, the user would have to input a heightmap style texture where black space represented areas the generator could not go. With the generator checking the relevant pixel on both of these maps before each generation could be suitably constrained by its environment (Fig 2).

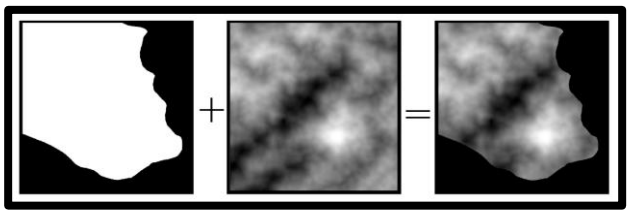

Figure 2: A water map (left) and Perlin map (middle) combine to control the generation

2.2. UI and Customisation

Next, it was decided that the way that cities would be generated would be through placing markers on the terrain that would act as an 'epicentre' for the generator to propagate from. These markers would place prefabs of the road generation scripts in up to 4 different directions, and the user could place as many of these on their terrain as they wanted, each with their own specified direction and generator settings. It was decided that the direction that these prefabs were pointing was not obvious enough to the average user, thus a large arrow was modelled in Autodesk Maya that would be used in the construction of a 'Gizmo' system. These gizmos would represent a start to the generator, and provide an obvious indication to the user what type of start point they had selected, where it was, and in what direction it would go in at run time. Figure 3 demonstrates how they appear in the editor, and the easy to use context menu that appears when the start point is selected in the hierarchy.

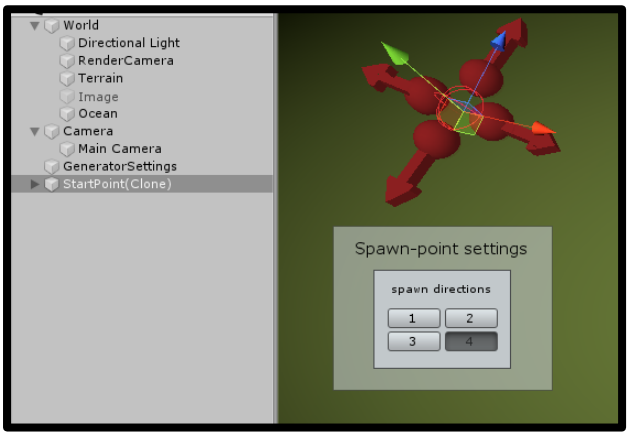

Figure 3: The gizmo for the city start point, and direction menu

A key objective of this project was to have everything easily accessible from the editor so that the user didn't have to open scripts and rewrite them to test new effects. The solution to this was to make a comprehensive menu that encapsulates all of the relevant generation settings, as shown in Figure 4. This menu could be used to create spawn points for the roads, set the population and water maps, edit their constraining values and change their colours. In an effort to keep the menu user friendly, flags were built in to the settings to warn the user if they entered a value for a variable that was likely to break the generator or give them an outcome that would likely provide them with an unwanted outcome.

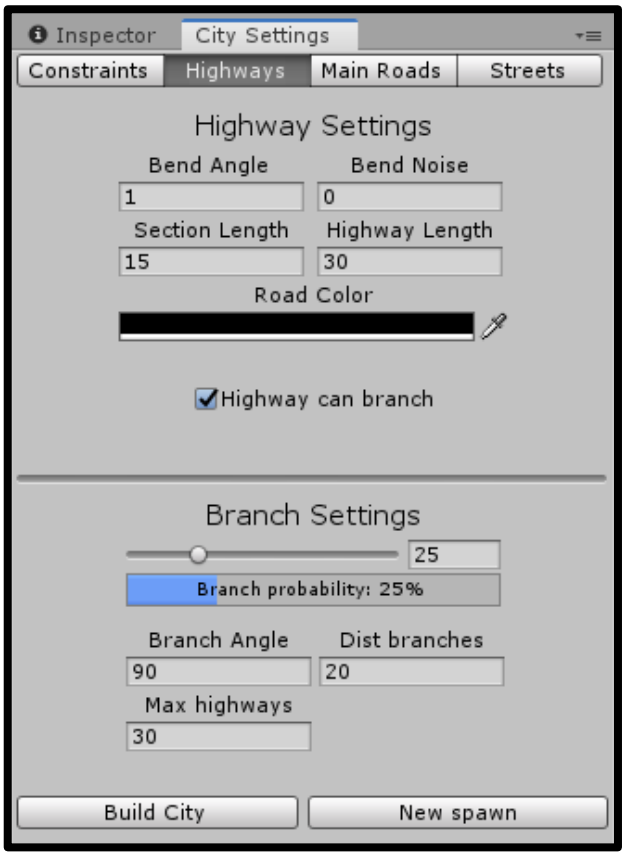

Figure 4: An example of a tab in the generator settings menu

Parish & Müller (2001) highlighted the fact that real cities have a particular and unique evolutionary growth pattern depending on how old they are and what access they afforded to nearby vital resources. This means that the road structure and urban planning of cities differs heavily around the world, and therefore there would be some requirement for the generator to be able to produce a wide range of city styles, from 'Manhattan' style blocks to a concentric style in Paris (Figure 5). A block/raster system could be achieved using the generator by setting 'bend angle' and 'bend noise' closer to 0 for each type of road, so that branches came off at perfect right angles to their parent road.

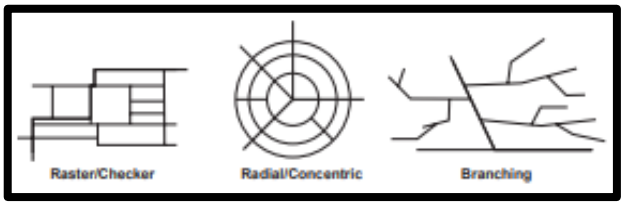

Figure 5: Examples of different city road patterns (Parish & Müller, 2001)

2.3. Quads and Junction Management

The roads in the system needed to be able to be aware of each other's position, and redraw themselves if necessary to form junctions and crossroads without it looking like they just happened to cross over each other by accident. The way this implemented was to divide the terrain up into quads, and have them store information on the contents of that quad. The quad functionality (Figure 6) was designed so that if a roads position needed to be checked against other objects in the world, it only needed to check the chunk of the terrain it was in for other potential obstacles which would save on computational resources.

```
private void InitQuads() {
    GameObject quadParent = new GameObject("Quads");
    quadParent.transform.position = new Vector3(0, 0, 0);

    int width = (populationMap.width / 10);
    int height = (populationMap.height / 10);

    int w = width;
    int h = height;

    for (int i = 0; i < 20; i++) {
        CreateQuad(w, h, quadParent);
        for (int x = 0; x < 19; x++) {
            w = w + width;
            CreateQuad(w, h, quadParent);
        }
        w = width;
        h = h + height;
    }
}

private void CreateQuad(int w, int h, GameObject parent) {
    GameObject q = new GameObject("Quad" + w + " " + h);
    q.transform.parent = parent.transform;
    q.transform.position = new Vector3(w, 0, h);
    Quad qw = q.AddComponent<Quad>();
    qw.quadPosition = new Vector2(w, h);
    quads.Add(q.GetComponent<Quad>());
}
```

Figure 6: The quad function that allows for faster processing of road and building intersections

There are 3 types of road within the generator: Highways, Main Roads, and Streets. These have unique settings in their own scripts, but they all derive shared functions from a `BaseRoad` script, where the rules on the population maps and intersections are stored, as these apply to all roads. The `RoadCrossing()` function handles 2 roads crossing over each other, and creates natural looking junctions and crossroads. When a new road position is called, it calls `RoadCrossing()` and loops through its current quad to check the position of everything else within that quad, and if the new suggested position comes into contact with anything that already exists. In that case, the road generator on that branch is destroyed and the final segment of road is redrawn so that it lines up with the closest point of intersection with the target road (Figure 7).

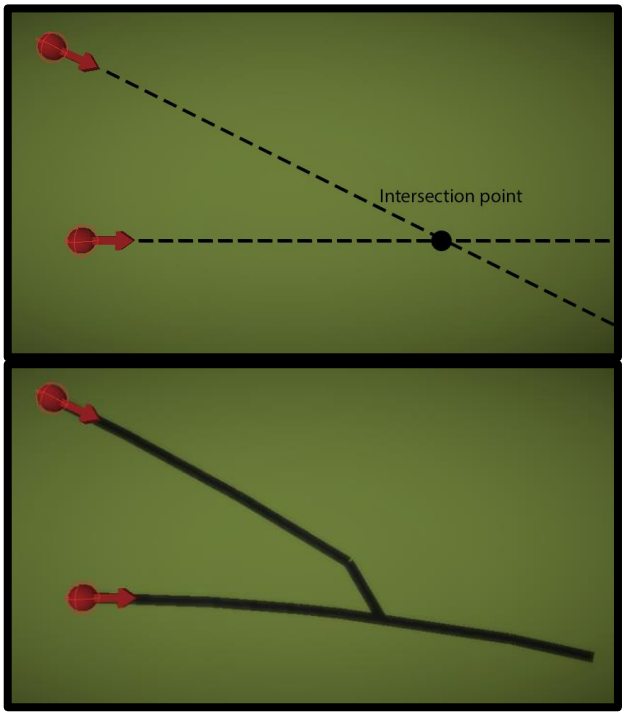

Figure 7: The intersection process to handle colliding road paths, visualised

2.5. Procedural Building Generation

The final implementation in this project was the buildings themselves. The decision was made to create these buildings at run time rather than instantiate prefabs as this allows for a much more random assortment of building shapes and sizes, and as there will be many buildings being generated at the same time, the issue of complicated meshes and textures being instantiated at the same time would increase the draw calls substantially and would result in a lower performance or a longer generation. The buildings can only generate on the sides of the street road type, which means that highways can't have houses running alongside them unrealistically, but the user can choose to disable the buildings from generating altogether if they wish from the settings menu.

Firstly, I defined 3 types of building that I wanted to generate: Small suburban houses, large tower blocks, and tower blocks with pointed roofs to indicate significant wealth in a built up area, or perhaps a place of worship in the smaller towns (Figure 8). Empty game objects are generated and positioned at the called location for a building, and then for each building type a set of vertices, normals, triangles, and uv's are defined. In an effort to save on function calls and lessen the stress on the GPU, each building object contains the meshes for 2 buildings, with an offset in between them for the road to pass down. This system means that the generator only has half the

work to do, which helps increase its generation speed.

There are scenarios where this double generation method would cause issues however; if 2 roads were in parallel too close together they would end up generating buildings on top of each other, or perhaps at a junction one house would not offset correctly and end up over the road. This is rectified by allowing the building generator to access `settings.streetLength` and calculate how much space there is to generate buildings on that particular stretch of street. This space is then divided up and fed back into the building size variable so that it is impossible for buildings to be too big for the street they are on.

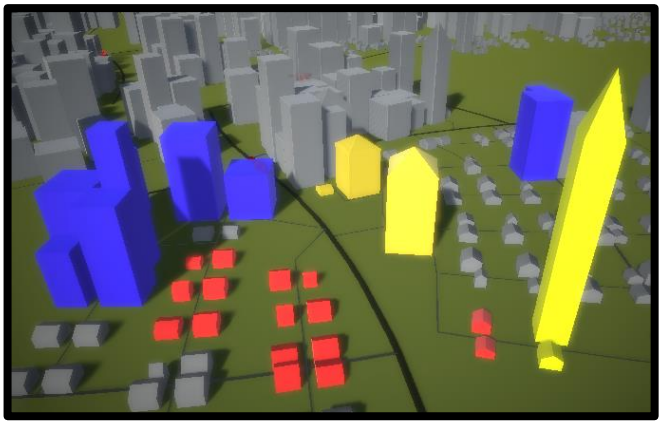

Figure 8: Houses (Red) Towers (Blue) and Point Towers (Yellow) all being generated in a scene

Another example of implemented optimisation in the generation of the buildings is the function `SingleMesh()`, which uses the inbuilt CombineInstance function to put all generated mesh faces on to one mesh and puts them in the parent mesh filter. This now makes the renderer consider the building a complete object rather than a collection of vertices and faces, which is easier to compute according to Prehn (2017), "Fewer meshes can mean fewer draw calls, even if the triangle count is the same".

In order to confirm this, I set up a simple experiment using the statistics panel built in to the Unity play mode window. I then tested building the meshes as individual face components, as a unified mesh after compression, and then generating 2 buildings per call as detailed above. The results are listed in Table 1, and show a clear optimisation in favour of combined meshes rather than individual.

Building Generation Mesh Method	Avg. FPS	Avg. CPU Batches	Avg. Tris
Individual – 1 building	24.61	9845	356k
Combined – 1 building	62.32	1327	352k
Combined – 2 building	71.12	972	259k

Table 1: Results of optimisation tests on building generation.

Reflection

From the earlier research carried out, it became apparent that there were many different methods of procedurally generating content and therefore a lot of avenues to explore when trying to build my own system. The main rivals to the Perlin style method I have used in this implementation was Wave-Function Collapse and L-Systems. Both presented interesting ways of creating a City, but they seemed too complex for the scope of my project. From my research I concluded that those systems would return only raster/block style cities and would not give me the option for the customisation I have with the Perlin noise map and settings menu. The systems for road intersections and building generation also tie in nicely to the Perlin system, and there seemed to be more choice for relevant research material online. Each step in the development was a clear transition from the last, with each system laying the foundation for the next one to be constructed.

Evaluation

During development, several issues arose with the plan of the generator and these had to be mitigated or removed entirely in order to proceed with development. The quad system was originally implemented as a fix to low frame rate during generation, but ended up being embedded in the system when handling intersections and impassable terrain. This is not a perfect solution however as a road just crossing the corner of a chunk and immediately entering a new one would not have access to that chunks occupied list until it was already inside and potentially colliding with something. However, for the most part this system was suitable for the scope of this project.

Another problem with the generator was that I had decided early on to generate roads by using segments of line renders. The initial decision was made with the idea that they were easier to compute and manipulate procedurally than a spline road would have been, and any problems with them looking not as good wouldn't be noticed from afar, which is where the city would probably be viewed from in its video game context. As suspected, the illusion of a curved road from lots of straight lines worked well from afar, however this would have proved to be a major and obvious pitfall had the generation worked on a terrain. I began playing with the idea of each road spawning 2 children game objects at each end of the road that represented the start and end of the segment, and have them only be able to link to the opposite member, like magnets. But with a terrain that might have its surface undulate quickly, the road would potentially clip through at either end if the segment length was too long.

Also, calculating the surface normal of the terrain at any given point would have potentially thrown up inaccurate readings for the same reasons.

3. Discussion of outcomes

A major problem with the development was incorporating verticality to the generator. I decided I was going to write the generation for the roads and buildings with only 2 dimensions in mind, thinking that the transition later on would only require the addition of a few lines of code and it wouldn't be too much of an issue. However, by the time I realised the extent of the upgrade required to make the generator capable of reading the height of the terrain, it was no longer feasible to redesign the generator from scratch. I tested 2 workarounds to try and alter the height of generated objects after they were placed.

The first solution was to use a looping ray cast that would go over each pixel in the terrain and shoot straight down, measuring the distance that that part of the terrain was away from y=0. This information would then be stored in a list and then when a road was generated its `(x,z)` value would be inputted into a formula to find the appropriate reference in the list, and therefore the hit result with it. This method was overcomplicated however, and cause serious performance issues with terrains larger than 100x100 that would make it totally impractical to implement. The alternative was just to use a `GetPixel()` on the heightmap to return the same value, but this couldn't be applied to the roads as the roads are instantiated at a local space of `(0,0)` which meant the generator didn't know what pixel to look up. There wasn't time to redesign the generative functions and therefore I wasn't able to implement verticality in this version of the generator.

At the conceptualisation phase of this project, the scope with which I set for this project was far greater than what has actually been achieved. A great inspiration for the design of this project was from the procedural road generation prototype for the proposed game 'Subversion' from Introversion Software (2007), and the main challenge I set for the development was to make the generation work on any heightmap, and the generator would be intelligent enough to mould itself to its geographical features.

Even though the end result of the project didn't live up to the initial vision, I still feel there are many aspects of its design that can be taken as a positive aspect. In a wider context, some of the systems developed to integrate together and solve problems with the development seem to be a unique solve amongst the other examples of city generation that I've seen, an example of this being the quad/intersection system. Something I've personally never attempted before within Unity is the vertex declaration method of creating object primitives; I've always instantiated new cube game objects at great expense to the computational resources available for the project. Using line renderers and vertex declarations to create the geometry within the project may be an imperfect solution but I believe within the scope of the project they were appropriate methods to use.

There is a whole range of projects that could build on this initial foray into the world of PCG. As well as adapting the generator to work with a vertical component, the cities could have an integrated traffic flow system using AI algorithms such as A*, that could add some dynamism to the world and make it less static after generation. Other possibilities include incorporating city districts into the generation like commercial, industrial and residential districts from games such as SimCity (Electronic Arts, 2013) and Cities: Skylines (Colossal Order, 2015). The buildings could have more complicated designs and procedural meshes, as well as the possibility for having landmarks.

4. Conclusion and recommendations

The field of PCG is rapidly expanding with the emergence of new technologies and their popularity as a software development tool will only continue to grow. Throughout the creation of this project, many new ideas and methods for problem solving have been introduced to me along the way, and many seemingly insurmountable obstacles have been nullified. The final outcome of the project was not exactly what I had set out to accomplish but the foundation has been laid for future potential side projects to improve it and make it something truly useful in the context of a games development. The logical next step for this project is to allow it to generate with some verticality.

Whilst the idea of a procedurally generated environment is not a new one, it is fair to consider that with a near countless amount of combinations to achieve this style of generation my project is unique to its own design and provides a learning opportunity for other developers seeking to reproduce something similar, in both its positive and negative aspects. The creativity that is facilitated by working within a field so broad really allows one's individuality to shine through and discover what kind of developer we are, and where our strengths lie.

5. References

Parish, Y. and Müller, Procedural Modelling of Cities. Available at: https://cgl.ethz.ch/Downloads/Publications/Papers/2001/p_Par01.pdf [Accessed 24 July 2020].

Prehn, R. (2017). Unity Tip: Combine Meshes for Performance and Organization. [Online]. Available from: https://medium.com/acrossthegalaxy/unity-tip-combine-meshes-for-performance-and-organization-c3515c844fdb [Accessed 24 July 2020].

Short, T. & Adams, T., 2017. Procedural generation in game design, Boca Raton: Taylor & Francis, CRC Press.

Khalifa, A., Perez-Liebana, D., Simon, M. and Togelius, L.&J. (2016) General Video Game Level Generation. *In Proceedings of the Genetic and Evolutionary Computation Conference 2016* [online]. [Accessed 27 July 2020].

wikipedia.org. *Rogue (Video Game)*. Available at: <https://en.wikipedia.org/wiki/Rogue_(video_game)> [online]. [Accessed 29 July 2020].

wikipedia.org. *SimCity*. Available at: <https://en.wikipedia.org/wiki/ SimCity> [online]. [Accessed 28 July 2020].

wikipedia.org. *Cities:_Skylines*. Available at: <https://en.wikipedia.org/wiki/Rogue_ Cities:_Skylines> [online]. [Accessed 29 July 2020].

Bibliography

Møller, Tobias & Billeskov, Jonas. (2019). Expanding Wave Function Collapse with Growing Grids for Procedural Content Generation.. 10.13140/RG.2.2.23494.01607.

Arnaud Emilien, Adrien Bernhardt, Adrien Peytavie, Marie-Paule Cani, Eric Galin. Procedural Generation of Villages on Arbitrary Terrains. Visual Computer, Springer Verlag, 2012

Olsen, J., 2004. Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. Realtime Procedural Terrain Generation. Available at: https://web.mit.edu/cesium/Public/terrain.pdf

Melbourne School Zones. Melbourne School Zones. Available at: http://melbourneschoolzones.com/.

Van Pelt, J. 2014. Urban Planning. Urban Planning - Research in Flanders, 2, 3 Available at: http://www.researchinflanders.be/en/thematic-papers/urban-planning/.

Appendix A: Project Log

Luke Tolchard	Procedural City Generator - Project Log		
	Tasks set	Brief Summary of Outcome	Tasks to be taken forward
14/10/19	• Begin collecting research papers for report • Build an understanding as to how to begin prototyping	Set up the project space within Unity and found some relevant papers on procedural generation algorithms, still need to find more to make a conclusive decision	More research needs to be done to consolidate ideas of how to begin prototyping
21/10/19	• Discover some methods of procedural generation to try • Make a basic prototype of a system	Continued on with research this week, found some interesting papers that discuss Wave Function Collapse, and L-Systems. Still not in complete understanding of their role to play but it seems to be good way of trying to tackle this problem. I have also managed to get a simple camera switch script working, which will allow a user to view the plane from a range of different camera angles, which are fully customisable and easily extendable when the time comes to do that. https://gyazo.com/6a550d5b21f4cfce6d15f483ba76529e	Try and make a basic implementation of WFC or L-Systems to understand how they work
28/10/19	• Begin work on a prototype • Continue with PCG method research	Following a meeting with Sean I have turned my research attention to how cities are actually designed in real life to give an idea of good functionality for the project	Look into saving objects from play mode in Untiy
4/11/19	• Research saving game objects in play mode	As the city would be built in play mode, as soon as I close out of it the city would be lost. I've been looking into trying to find a function that allows for the grouping and saving assets but found nothing that I can replicate in a test project yet	Keep looking for ways of saving objects, and look at how to manipulate unity terrain via code
11/11/19	• Hold a focus group to ask my cohort what they would want to see in a project like this	As this will be a development tool, I thought it a good idea to ask my cohort for their opinions on what would be important to them for a Unity tool	Publish results in a blog post and begin to rethink research focus based on response
18/11/19	Did not complete any work due to illness	Did not complete any work due to illness	Did not complete any work due to illness
25/11/19	• Research into Perlin noise	I've seen some work that suggested Perlin noise is a good way of clumping similar values together, which could be used as a way of making districts of similar buildings in a city	Further explore validity to Perlin noise, try and prototype something with it
02/12/19	• Research into UI and user experience • Consolidate Perlin Noise research	Perlin noise seems like a very simple way of giving the effect I'm after, a simple prototype using a GetPixel function to visualise the noise on a texture looked great	Hold another focus group to ask about UI, and begin work on the research report
09/12/19	• Complete another focus group and record results in a blog post • Continue work on the research report	I have completed another focus group with the same set of students as before. This was mainly aimed at trying to show them the implemented changes from before based off their feedback. The minutes can be found here: gamedevelopment676677755.wordpress.com /2019/12/05/minutes-for-focus-group-3-12-	Publish the results on a blog post and continue with report
16/12/19	• Continue with report	This week I nearly feel ready to try and make a proper prototype city generator, I need to think about what settings I want to be in this	Start to create a settings menu using the window functionality and begin to

	• Think about what settings could define a road • Research best way of generating roads	menu, and how roads will interact with each other.	construct a base road class
23/12/19	Week of due to Christmas	Week of due to Christmas	Week of due to Christmas
30/12/19	• Create a settings menu • Construct a base road class	The menu is available in the project now but the Unity GUI code is confusing and difficult to work with so it's taking some time setting it up how I want it. I guess it's more important to make sure its functional first and pretty later so I'll focus on importing some settings	Research road generation and make some settings for the menu
06/01/20	• Work on report • Research Road generation	The method of road generation seems to be a bit of a crossroad... I could choose to create models of road segments and have them procedurally selected from a wave function collapse algorithm but risk the city looking a bit rigid in how it appears or I can do something simpler like lots of line renders overlapping and only view from afar so the stitches aren't visible, but have a more fluid looking city	Decide a road generation method and write settings for it
12/01/20	• Finish research report • Apply settings to road generation	I've decided to use the line renderers as the January demo is fast approaching and I don't want to experiment with WFC this soon. I have finished the research report and have started applying some settings to the roads, I need a way of defining where the generation starts from instead of always at (0,0)	Create a start point function that lets the user choose where the city starts.
20/01/20	• Refine scope after Demo feedback • Complete additional research into building generation that's easier on the GPU	The demo went well, the roads generated okay but randomly selecting building prefabs and instantiating them was very costly and cause a bit of lag, need to see if there's another way of doing that. I also need to start looking at ways of moving the generation to a 3D terrain. City start location is nearly implemented, and I want to make it a physical object you move around the scene with a giant gizmo model like a flag or something that gives a clear indication where it is	Research cheaper ways of creating lots of primitives and create flag gizmo for start location implementation
27/01/20	• Test creating buildings using vertex declarations • Rethink Gizmo design	The flag gizmo looks great and makes it much easier to see where the start is, however it isn't really helping in regards to what direction the generation is going to start in so I may need to redesign it to a big floating arrow or something. Also found a unity doc about rendering boxes using vertex definitions to edit a game objects mesh filter. Might be cheaper on a GPU to create a new empty game object and then edit its mesh filter to show a box. Should be relatively simple to make the min and max size for each building change procedurally as well, so that kills 2 birds with 1 stone	Test different road types deriving from base class, think about how roads need to interact with each other.
03/02/20	• Develop a system for road intersections	If roads branch out over the top of each other it looks really messy and horrible, I need to develop a function that kills the road generator if its going to come into contact with another road. But I don't want that to happen all the time as then no roads will ever cross and there'll be no junctions. Maybe a % chance to intersect, or stop just before	Experiment further with road intersections
10/02/20	• Split road type into 3 different roads	It's occurred to me that there are many different types of roads that have different	Develop menu further to make the GUI look better

	with individual properties • Update settings menu to reflect the change in road structure • Develop a system for road intersections	functionality in the real world. I don't want houses spawning next to motorways, and I don't want streets stretching on for miles and miles before stopping. Easiest way to stop this is to make sub classes for each type of road, and make more tabs on the menu to contain them all (Makes the menu look more important as well)	and carry on working on intersections
17/02/20	• Make the heightmap editable on the menu • Implement intersections	Added a texture map slot on the menu for the heightmap so the user doesn't have to use the inspector. I've found a way round the intersection problem by making the branch generator check the terrain for something it can intersect with, and if its going to hit something it deletes the last road segment when it collides, then it redraws to the road. The calculation is costly when every branch is doing the same though	Optimise the intersection calculation
24/02/20	• Redesign road generation classes to allow for 3D terrain implementation • Optimise intersections	I may have made a mistake by leaving the upgrade to 3D generation so late, the code is now difficult to rewrite so that it works vertically. I did manage to optimise intersections though by dividing the terrain into quads so a road only has to check for intersections within its quad rather than the whole map.	Keep working on 3D implementation
02/03/20	Didn't complete any work due to coronavirus related issues	Didn't complete any work due to coronavirus related issues	Didn't complete any work due to coronavirus related issues
09/03/20	Didn't complete any work due to coronavirus related issues	Didn't complete any work due to coronavirus related issues	Didn't complete any work due to coronavirus related issues
16/03/20	Didn't complete any work due to coronavirus related issues	Didn't complete any work due to coronavirus related issues	Didn't complete any work due to coronavirus related issues
23/03/20	Did not complete any work due to mental health issues	Did not complete any work due to mental health issues	Did not complete any work due to mental health issues
30/03/20	• Push for 3D implementation	As I've been sick and unable to work for several weeks I am now very behind schedule on most modules so do not have a lot of time to dedicate to this problem right now. Have acted on the contingency stated in my proposal and reduced all hours at my part time job to help me catch up and figure this implementation out	Keep working on 3D implementation
06/04/20	Did not complete any work due to mental health issues	Did not complete any work due to mental health issues	Did not complete any work due to mental health issues
13/04/20	• Keep working on 3D implementation	I don't have time to rewrite the road generation code, but I feel that I may have to in order to convert this project over to 3D terrains rather than 2D planes. Might have to draw up compromises on the final implementation In order to meet the deadline	Keep working on 3D implementation
20/04/20	• Prepare for resubmission of CTP	I no longer have time to finish this in time for the deadline due to my health issues, I do have a chance to finish my other modules however, so I will be shelving this project until the resubmission period begins where I shall go again and work hard to finish this as well as I can	-

RESIT PERIOD			
22/06/20	• Redesign camera system • Refresh memory on project and code	It has been a long time since I've opened this project, so I'm going to spend a week to refamiliarise myself with how it works and what I need to do to convert it to 3D. The camera system is currently annoying to use so I shall break myself back in gently by redesigning it to be more intuitive.	Begin working on a solution to the issue
29/06/20	• Develop a ray cast system for detecting terrain height	Its apparent to me that attempts to have the roads created at the correct height have all ended in failure, which has given me the idea to try and edit their height after all the generation is finished. That way I no longer have to try and back pedal on the code that is already in place, I just have to develop a new system that plugs in to a new road's x,z position and then adjusts its y.	Develop a formula for looping over a terrain and storing values in a list for retrieval
06/07/20	• Implement excel implementation to code	I've made a game object that loops over every pixel of the terrain before generation begins that shoots a ray cast straight down and records the distance to the terrain in a list. When a road created at that pixel needs to know the height it needs to be bumped to, it needs to access the right value in the list and adjust accordingly	Optimise that function in C#
13/07/20	• Bug fix the height tester	The height tester sort of works, but nowhere near well enough for use in the project. With a terrain of 1000x1000, unity freezes for approx. 10 seconds at the start of each generation as it stores the height values in the list. The values are also lacking precision, most are out of the true value by about +/- 10%. Research online suggests that Unity makes ray cast results imprecise under heavy duress in an effort to save on computation, so I suspect that dumping this workload on the system is causing it to immediately break. To intentionally slow it down even more so that it is more precise makes the system not worth using. It's back to the drawing board.	Contemplate more ways the 3D terrain system might work.
20/07/20	• Try and read a GetPixel value off the heightmap	I was trying to make it so much harder than it needed to be, and have resorted to using GetPixel to do the same job as the ray caster, just at a fraction of the computational effort. I do not know how to assign it to the roads however as the roads themselves are created at 0,0 of their 'parent' which is a generator prefab that is immediately deleted after its child is created.	Will have to begin the final report write up
27/07/20	• Work on writing the final report • Create showcase video • Create control readme	I have begun writing the final writeup report, it's a shame I didn't manage to get the project to where I wanted it to be but there is a lot in there that I was able to overcome and develop so I am proud of the implementation I will be presenting	Submit!
03/07/20	SUBMISSION DATE	SUBMISSION DATE	SUBMISSION DATE

Appendix B: Project Timeline