Comprehensive Creative Technologies Project: Procedural Generation of Virtual Cityscapes Using WaveFunctionCollapse Algorithm

Alexander Hillman
Alexander3.Hillman@live.uwe.ac.uk
Supervisor: Bethany Mackey

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

Abstract

The WaveFunctionCollapse (WFC) algorithm is an example driven generation algorithm created by Maxim Gumin (2016) which takes in an input image that demonstrates what tiles can be used and how they connect to each other. With this information the algorithm then attempts to recreate a similar output image using the provided data. This project aims to explore how the WFC algorithm could be used to procedurally generate 3D cityscapes within the Unity game engine. It attempts to recreate the core functionality from the WFC algorithm and adapt it to create a designer tool to aid in generating customizable virtual cities. The result from this project is a system which does not use the input image component of the WFC algorithm and instead combines user-defined tile sets and ruleset as an alternative. This project details the reasoning behind why the WFC algorithm was chosen, its main features and functionality, and the product that was created as a result of this research as well as an evaluation of the final product.

Keywords: Procedural Content Generation, WaveFunctionCollapse (WFC), Socket, Cell, Tile, Superposition, Entropy.

Biography

This project was chosen due to prior work conducted with the streaming of an open world computer game level. The level created for the previous project was created using heightmaps and as a result some form of procedural generation would help to improve the project by adding some variation, this project was such an attempt. Carrying out such a task allowed for the development of skills in areas such as system tool creation, it also provided a chance to test a variety of skills in various areas of game development that had yet to be experienced. This project was further created from a failed prior project of a different research area, as such procedural content generation was chosen as a different research topic, and this project was created. Further work created by this author can be found at: https://ahillman2000.github.io/

Accessing the project

The Projects source code can be accessed via: https://github.com/Ahillman2000/Procedural-Generation under the main project branch. Unity 2021.1.21 is required to run any non-executable versions of this project.

The video for this project can be found at: https://youtu.be/tduuEkUc3Hk

1. Introduction

Procedural content generation (PCG) is the process of using algorithmic logic to automatically create content with limited or indirect user input, as described by Togelius and Shaker (2016). Within the video games sector, such generated content can range from assets such as textures, meshes or procedural effects to story content, entire puzzles, levels, or worlds. These can be generated in such a way that users may never encounter the same identical content twice (Smith, 2015).

Games such as Eldrich (Minor Key Games, 2013) or Minecraft (Mojang, 2011) for example use procedural content generation for the purpose of level design. Minecraft for example utilises Perlin noise to generate the game world in a way that is both natural feeling and theoretically infinite (Robertson). Since each world is procedurally generated players do not know anything about how the world is setup before they start playing, they are therefore required to explore the world for themselves to progress and anything they discover will be unique and new to that specific playthrough. By using Perlin noise, the level design better reflects real world environments to greater immerse the player in surrounding world. The features created by procedural generation play into Minecraft's core goal of discovery and creativity since players are given near infinite inspiration.

Image 1: Procedural Generation in games. (Mojang, 2011, Minor Key Games, 2013)

Short and Adams (2017) identify multiple reasons why procedural content generation may be used in game design.

- It can save time on creating large amounts of content compared to a more manual approach.
- It adds replayability since one generator can produce many similar but varied

instances of content, which in turn allows designers to offer players different experiences that feel unique to them.
- It can produce content to a scale that can be difficult to achieve especially within small development teams.

These reasons give quantifiable evidence for why procedural generation is necessary within the modern-day games industry and how using it can be a benefit to developers and studios alike.

The project will explore generating the layout and content for a city-like environment so that the final product will be a system that can procedurally generate entire city environments with relative ease. Procedural content generation was chosen for this project specifically to automate the process of creating these environments so that they can be generated either before or at runtime and to partially reduce the workload on designers whilst still providing some level of control.

The main objectives are:

- To automate content generation by implementing and adapting the WaveFunctionCollapse algorithm
- To increase customization and designer control over how procedural content is generated

And the key deliverables for this project are:

- A system that can procedurally generate cities with varying sizes, layouts, and content
- A user interface that either allows for a designer to specify certain conditions or to place objects that affect how content is generated

2. Literature review

Kelly and McCabe (2006) have detailed several procedural techniques for generating content, these methods include using Fractals, Perlin noise, Voronoi Diagrams and L systems. By using this paper as a starting point, many other methods were found that could be used for procedural content generation, however for this project three main approaches were identified that had the most potential.

2.1 L-systems

CityEngine by Parish and Muller (2001) utilizes an L-system based approach to model cities through generating buildings as well as the road networks needed to support them. The system takes image maps as input to generate highways and streets, it then creates lots of land from the generated road networks and then fills these lots in with geometry. The L-system operates by recursively rewriting mechanisms based off a set of production rules which often leads to the creation of a fractal like structure (Santel, 2019). An L-system approach was chosen for this system because of their branching nature that allowed for a main road to be generated which could then diverge off into smaller, more local sub-road networks in a natural way.

The CityEngine system then uses a population density image to determine the distribution of the population to create highways from one population dense area to another, from here streets are then grown following a street pattern until an area with no population is reached. From the generated road network, the system has divided the area into areas called blocks which can then be subdivided into lots through recursive division of the longest edges of the block.

Image 3: CityEngine's approach to block and lot division. (Parish and Muller, 2001)

This implementation by Parish and Muller (2001) seems to be able to generate different city layouts using different patterns allowing for variation even within a single generation. The resulting layout appears to be very realistic in nature since the system can use real-world geographical data as input, with a variation in building height and appearance which are also created through procedural generation.

2.2 Voronoi Diagrams

Another potential method would be the use of Voronoi diagrams, these can be used to partition a given space into cells by randomly placing site points on a Euclidean plane and then generating cells around each of these sites so that the region of the plane is closer to that site point than any others (University of Bristol).

This method is used by Sun et al. (2002), most notably to produce a population-based template model for generating non-structured virtual cities. A Voronoi diagram was used since the it better reflects the relationship between population dense areas and the road networks needed to support them. Sites are generated from a population density map and the spaces

between them are partitioned to generate cells. These cells then become districts of land to support buildings and infrastructure and the edges between each of the generated districts become sections of the overall road network. Larger generated cells result in longer roads to bisect them from others, reflecting the need for a stronger road to support larger areas with a greater populous.

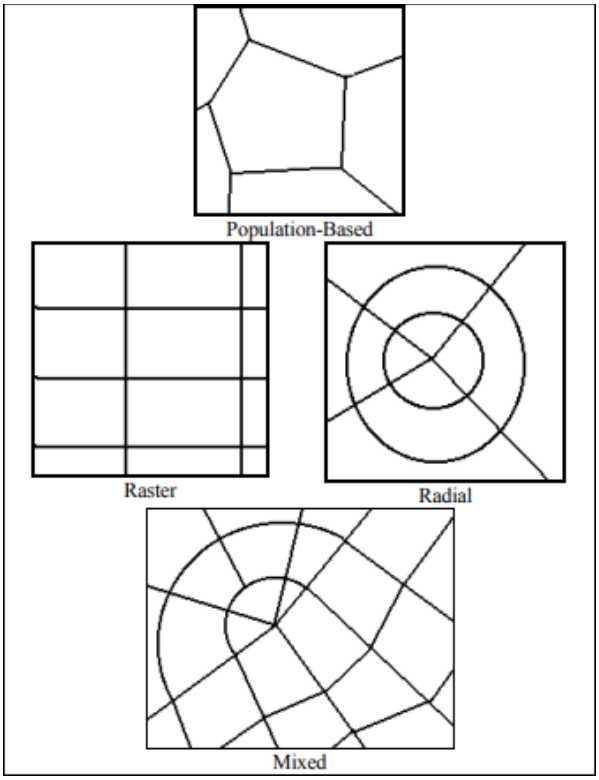

Image 4: Frequently used road patterns as identified by Sun et al. (2002), the most notable being the population-based pattern that produces a non-structured pattern.

This paper presented a reasonable methodology to create non-raster layouts so that parcels of land could be non-uniformly sized, it also allows for curved roads to be generated using the Minkowski distance as shown by Galindo-Torres (2010). It also appears to have some level of designer interaction with the ability to colour the input image to alter aspects such as land-sea boundaries or population density. However, the paper has a heavy focus on road networks with little information on how the generated parcels of land are populated.

2.3 WaveFunctionCollapse (WFC)

The WFC algorithm (2016) is an example-driven generation algorithm that produces outputs which are locally similar to a given sample input image. Most Notably, the algorithm has been implemented in games such as Bad North (Raw Fury), TownScaper (Raw Fury) and Caves of Qud (Freehold Games, LLC) to name a few.

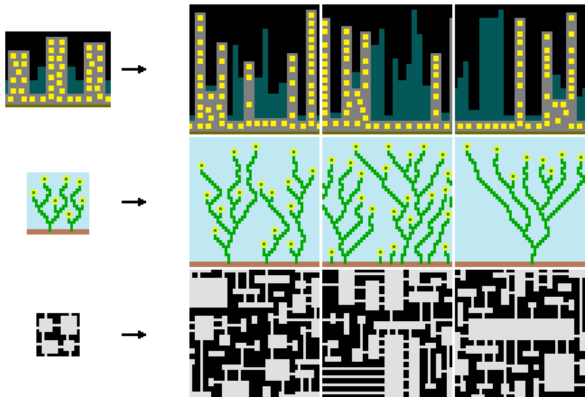

Image 5: Examples of the WFC algorithm (2016). The algorithm takes in an example image (left) to generate a similar output image (right).

Karth and Smith (2017) have summarized the algorithm down to four key stages:

1. To extract local patterns from the input image
2. processes the extracted patterns into an index
3. incrementally generate an output by eliminating the possible states of neighbours
4. Generate the final output from the total assignment

Stage 1 from the above steps can be omitted as identified by Scholz (2019). Their implementation instead explicitly identifies the relationship between tiles through designer-specified relations rather than inferring them through an example input image. Scholz's work attempts to implement the WFC algorithm to procedurally generate a 3-dimensional terrain using blocks of geometry. The system takes a tile set as input and the matching information is calculated based on the geometry on each of the sides. This data is then compared to each of the neighbouring tiles for the horizonal faces and the grid is propagated.

Image 6: Model Synthesis in action. A set of input tiles (a) is used to generate an output model (b) (Scholz, 2019).

Procedural generation using this form of the WFC algorithm seemed to provide a great deal of designer interaction with Stålberg's wave implementation allowing for users to choose specific tiles to be placed in spaces. This methodology also does not need input data unlike the other two methods researched, allowing for the designers to interact with the system itself to alter it.

3. Research questions

- How effective is the WFC algorithm at creating procedurally generated content?
- Assess the extent to which the WFC algorithm allows for parameterisation for designer control.
- How effective is WFC at producing organic-looking cities?

The main question for this project will aim to answer how well the chosen algorithm can create procedural content. This will be evaluated against Kelly and McCabe's (2006) criteria to determine aesthetic factors such as how realistic the generation is to real-life scenarios, the level of variation that the system can produce as well as more technical considerations such as how computationally complex the system is or how long it takes to produce an output.

The secondary research question will aim to build upon the implementation of the algorithm so that it can be used for more interesting functionality that it wasn't originally conceived, such as allowing for designers to place buildings within the map before it is solved in a similar way to that of Stålberg's Wave implementation. The reasoning behind this decision is that if this tool is used by designers, then they will not be limited to purely random generation and instead will be able to influence it in a way that suits their needs or creative ideas.

As well as the two main research questions, a potential third question may be introduced as a stretch goal. This question could help to fill in the gap surrounding the type of content that PCG is producing, especially concerning procedurally generated cityscapes. The research conducted shows that a considerable proportion of the content being generated is grid based, resembling a raster pattern (sun et al). This question would therefore like to investigate if something more organic could be created using this algorithm.

4. Research methods

To answer the research questions identified above, primary quantitative study was conducted. Conducting this form of research was chosen over qualitative research due to the technical nature of the project. The focus of this project was to assess the effectiveness of a methodology rather than the impact of the final product on users, which would have required descriptive and open-ended responses. The research conducted also allowed for the project to be evaluated on a technical level and determine a quantifiable response to how successful the project was at answering the proposed research questions.

Kelly and McCabe's (2006) key criteria to evaluate generation systems offered a good range of factors to determine the overall effectiveness of the final product:

1. Realism – Does the generated city look like a real city?

2. Scale – Is the urban landscape at the scale of a city?

3. Variation – Can the city generation system recreate the variation of road networks and buildings found in real cities or is the output homogeneous?

4. Input – What is the minimal input data required to generate basic output and what input data is required for the best output?

5. Efficiency – How long does it take to create the examples shown and on what hardware are they generated? How computationally efficient is the algorithm?

6. Control – Can the user influence city generation and receive immediate feedback on their actions? Is there a tactile intuitive method of control available or is the control restricted? To what degree can the user influence the generation results?

7. Real-time – Can the generated city be viewed in real-time? Are there any rendering optimisation techniques applied to enable real-time exploration?

These criteria offer a chance to analyse the project from both a technical standpoint (how well the implementation preforms) as well as a more subjective one (whether it resembles real

life cityscapes). These questions will therefore require some level of numerical analysis such as the framerate or memory usage whilst the project is running as well as more personal opinions such as how well the final product looks aesthetically.

5. Ethical and professional principles

The project did not need to worry about any external ethical considerations concerning data collection or storage methods due to there being no outside participants. Since the project was more focussed on the implementation of a system rather than its effect on a userbase, no external testing input was required as this was outside the projects scope.

Where ethical and professional principles had to be applied however were during the initial stages of the project when research and development took place. The main considerations were related to intellectual property and copyright issues concerning the work of others. When conducting research, it was important to cite the contribution of others when referring to their work. Furthermore, any code, algorithms or functions that were not directly implemented was also cited within the source code. This ensured that proper credit was given to avoid plagiarism or infringement of intellectual property.

6. Research findings

The research conducted identified numerous possible methods to generate procedural cityscapes. The work of parish and Muller (2001) and Sun et al. (2002) seemed to rely on multiple image maps for input. These were used to shape the way in which cityscapes were generated by providing the system with data such as the overall shape of the city using land-water boundary maps, where structures roughly need to be using population density images. These maps allow for designers to be able to alter the way in which content is generated externally from the rest of the system, instead opting to be done by either scanning in pre-existing 2D images or by creating them with drawing tools. This method of designer interaction lacks any form of personal interaction with the system, users do not see how their actions will impact the final product since the way in which this is done has been abstracted from the generating system. With the lack of designer intimacy with these systems in mind, the project decided to better engage designers by integrating them with the process more and allowing them to immediately see the results of their design choices. This was furthered by Stålberg's Wave implementation (Stålberg, O) which allows users to not only select the tiles that they want to use but also to specify the tile of a chosen cell. As such, the use of input images to control how content was generated was abandoned for an alternative method.

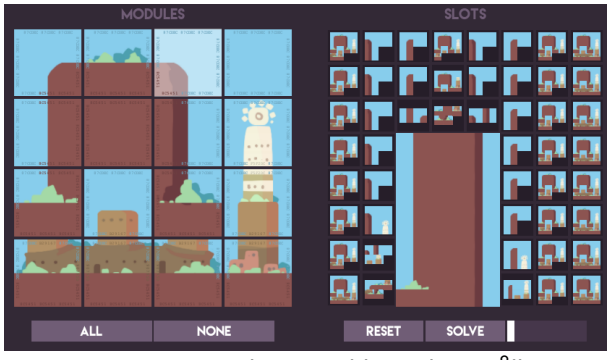

Image 7: WFC implemented by Oskar Stålberg.

Out of the possible methodologies discovered, the use of Voronoi diagrams and the WFC algorithm showed the most merit. These did not rely on external data to be included and seemed the most interesting to implement. One benefit of using Voronoi Diagrams would be that the final product would be less structured in appearance due to the formation of irregular parcels of land which in turn would create a shape that looks very organic. More research revealed that using different distance calculations would result in the regions being divided differently. The Drawbacks of using this method however were discovered after attempting to implement a very simple implementation of a Voronoi diagram. This highlighted problems with how edges were produced, with the edges often looking unnatural and not quite what the project was looking for, furthermore since the research discovered surrounding this method was rather lacking and vague, there was no clear method of how this could be used by itself to produce the desired results. It was not clear how this method could do more than just produce a 2D texture and how it could be used to extract out the necessary information required to identify and generate either the roads or parcels of land required.

[Voronoi implementation]

Using the WaveFunctionCollpase algorithm therefore showed the most promise for this project after eliminating the other researched methods. WFC seemed the most fitting methodology for this task for numerous reasons. It is a fairly recent implementation and as such is very current. Because of this, it has a lot of interest from within the field on top of it being very versatile and adaptable. Numerous

examples were discovered of others that have attempted to implement this algorithm, either by attempting it in their own way or by altering it to suit their needs like the work of Marian42 (2019) or Donald (2020). It therefore seemed likely that using this algorithm would not only result in an interesting product but also be both technically challenging and generally interesting to implement. Scholz's (2019) work was a good starting example; it demonstrated the possibility of implementing the algorithm within a 3D environment without the need for any input image and could be furthered with designer customisation.

7. Practice

The project started with a premade 2D tile set that rigidly followed the original implementation of the algorithm, it utilised an input example image and used this to generate an output (Image 8). It primarily followed a video tutorial for the code and structure, the result of this was a better understanding of what the algorithm was and how it worked. The initial attempt confirmed that the project needed to move away from the use of an example input image due to the overhead of information required to produce it. The majority of the code was removed since a different approach was going to be taken but some structures such as the Direction class remained as these could be used universally despite the shift in approach.

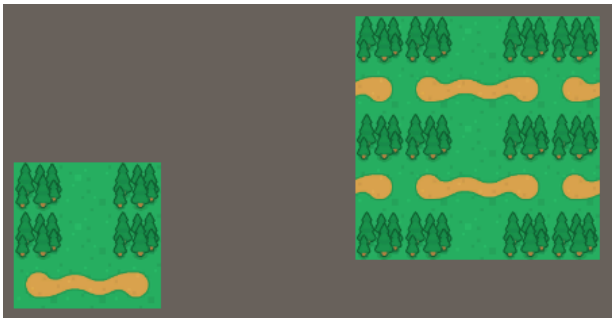

Image 8: The first iteration of the WFC system using an example input image.

The next attempt was closer to the final product, it again used a much more simplistic tile set. The aim was to focus on getting the core functionality of the algorithm working, thus the small set of tiles that were not complex in design. A tile set of grass and path sections was chosen, each tile edge could be one of three possibilities, an edge could either be grass, a horizontal pathway or a vertical pathway. Connections such as pathways could later be simplified to just a single pathway value if tile prototyping was used, but as a starting point these were kept separate for ease of use.

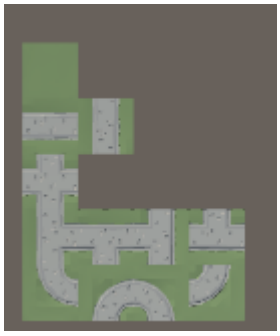

Image 9: The final tile set used for the second iteration of the system, including the additional tile pieces.

The initial resulting system did not produce the correct connections being made. The problem was caused by the solver propagating out until there was no impact on adjacent tiles as well as the limited tile set. The problem was solved by only doing a single propagation iteration on the immediate neighbours of the collapsed cell as well as by increasing the tile set to include corners and T-junctions on top of the existing grass, straight pathways and four-way intersections (Image 9). Making these changes seemed to solve the problem with incorrectly solving the grid (Image 10).

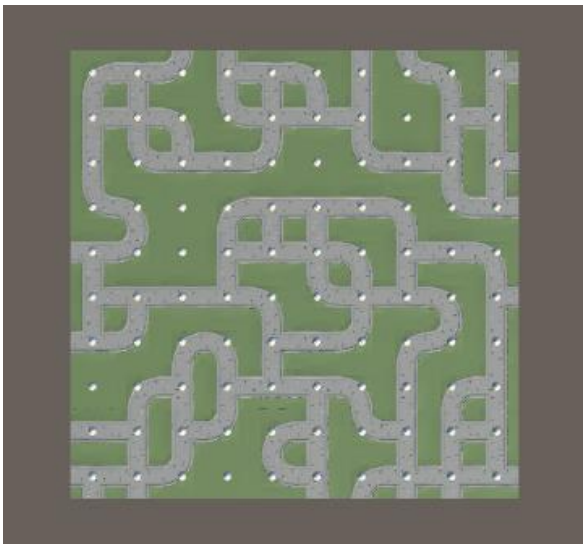

Image 10: A fully collapsed grid using a grass and pathway-based tile set.

This then resulted in the code structure that the final implementation iterated upon to be created. The final system implemented its own version of the key WFC stages which can be broken down into two main parts.

7.1 Editor Component

The first part to the system is the editor aspect, this is where designers create the tiles to be

included in the final product and specify key details about how the algorithm should operate.

7.1.1 Tile set

The core component of the WFC algorithm are the tiles which are joined together to form the overall cityscape. Each tile is converted into one of Unity's prefab objects. The prefabs are constructed from assets such as road or sidewalk tiles as well as buildings that are then placed onto the sidewalk assets (Image 12). Each prefab then has a custom script attached to it that specifies information about said tile (Image 11).

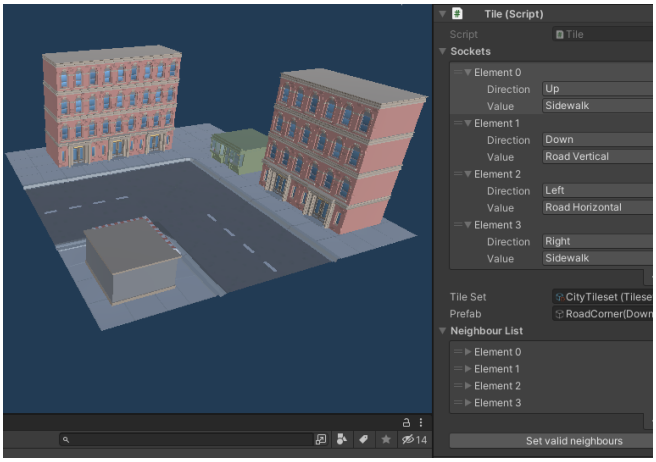

Image 11: The second iteration of the tile set used by the system to better reflect the cityscape goal of the project.

This information includes detail about each of the tile's sockets (edge connectors) as well as determining which of the created prefabs can connect to each of these sockets. For the current setup the sockets can be assigned as either sidewalk, road vertical or road horizontal, however if more connections are needed then they must be added to the enumeration that stores each of these values.

Image 12: The second iteration tile set used by the system to better reflect the cityscape goal of the project.

The tile set used is identical to that used by the initial pathway implementation, it covers almost all necessary scenarios except dead ends (Image 12). These then form the basis of system by which to be placed next to each other to form the overall cityscape. Each tile prefab is then added to a scriptable object that stores a list all the prefabs used so that they can be easily accessed and altered if needed.

7.1.2 Tile connections

Attached to the Tile script is an editor script button that checks each of the tiles against the others in the tile set. When the button is pressed it ensures that any changes made to the prefab are saved to the project so that these changes are not lost between sessions, sets the valid neighbours for the tiles and then records that a modification has taken place so that it can be undone if necessary (Image 13).

```
[CustomEditor(typeof(Tile))]
Unity Script | 0 references
public class TileInspector : Editor
{
    0 references
    public override void OnInspectorGUI()
    {
        DrawDefaultInspector();
        Tile tile = (Tile)target;

        if (GUILayout.Button("Set valid neighbours"))
        {
            EditorUtility.SetDirty(tile);
            tile.SetValidTiles();
            PrefabUtility.
                RecordPrefabInstancePropertyModifications(tile);
        }
    }
}
```

Image 13: Setting the valid neighbours of a tile prefab.

The script sets the valid neighbouring tiles checking the value of each of the four directional sockets against the opposite socket of each of the tiles in the scriptable object. If the two values are identical then the tile gets added to a list of valid neighbours in that direction. The result is four directional lists of tiles that can slot next to this tile in the given direction. By doing this to each of the tiles in the set, it is possible to determine all the possible connections for the given tile set.

7.2 Executable Component

The second part of the system occurs when the program is run. This is where the majority of the WFC algorithm is implemented and executed and is where the finalised cityscape layout and mesh is generated.

7.2.1 Grid Generation

The Grid Generator script handles the initialisation of the environment to be solved. It generates a grid of values to position each of the cells in a square layout.

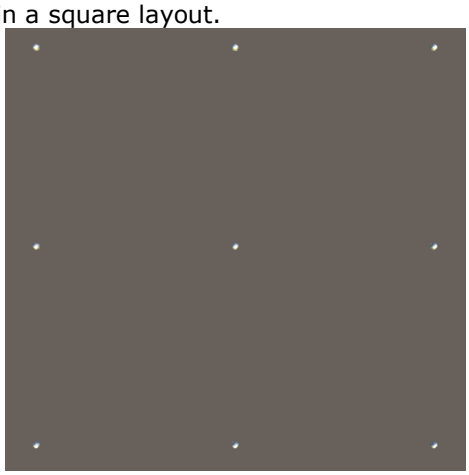

Image 14: Each cell's position indicated by a sphere.

Each Cell then stores information pertaining to a position within a grid, this includes information such as the cells index within the grid, its world positioning as well as the list of possible tiles that the cell could be, whether the cell has been collapsed (had its possible tile set reduced down to a single tile), and when it gets collapsed, the tile that the cell has been set to (Image 15).

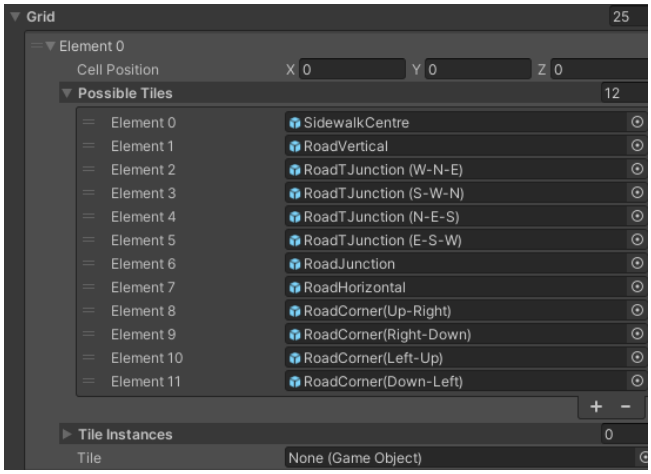

Image 15: An element in the Grid list denoting a cell.

When each Cell is created, it is parented to a main game object which acts as a container for everything created by the WFC system. The cell is then instantiated with the related information and is added to a list of Cells which then forms the grid. The attached Grid Generator script contains the function to find which cell within the grid has the lowest entropy (number of possible tiles).

7.2.2 Solver

The role of the Solver is to assign a tile to each of the cells in the grid. This is done by either selecting the cell with the lowest entropy, or if one cannot be found then a cell is chosen at random. The chosen cell is collapsed to a single tile by randomly removing each of the possible tiles until only one remains. Once the cell is fully collapsed the cell instantiates the tile as a game object and it notifies the solver that a cell has collapsed. The solver keeps track of the total number of cells that have been fully collapsed so that it only iterates whilst there are still non-collapsed cells remaining, once this is no longer the case the solver can be assured that it has fully solved the grid and that it no longer needs to continue.

The next step for the solver is to propagate out these changes to the neighbouring tiles. Since the cells are stored in list, the first step is to check which of the directions are valid and contain a cell since checking a cell located on the

edge in each of the four directions can result in a null reference since it checks for cell with an index value outside the range of the list of cells. The check returns a list of valid neighbours, containing information such as the neighbouring cell and which direction it neighbours the given cell.

With the neighbouring cells found, the solver then iterates through each of these neighbours. It iterates through each of possible tiles for the neighbour cell, checks these against the valid neighbour list for this cell. Ff the possible tile cannot be found within the list of valid neighbours then it is removed since it will not correctly match this tile (Image 16).

```
foreach (ValidNeighbour neighbour in GetValidNeighbours(cellToPropagate))
{
    var possibleNeighbours = cellToPropagate.GetTile().
        neighbourList[(int)neighbour.conectionDirection].neighbours;
    var otherPossibleTiles = neighbour.cell.possibleTiles;

    List<GameObject> removals = new List<GameObject>();

    foreach (var otherTile in otherPossibleTiles)
    {
        if (!possibleNeighbours.Contains(otherTile))
        {
            removals.Add(otherTile);
        }
    }

    foreach (var otherTile in removals)
    {
        neighbour.cell.RemovePossibleTile(otherTile);
    }

    neighbour.cell.ShowPossibleTileInstancesinCell();
}
```

Image 16: An element in the Grid list denoting a cell.

Once the cell has propagated it changes out to its neighbours, the solver then selects a new tile to collapse, and the process repeats until all the cells in the grid have fully collapsed.

7.2.3 WFC

The WFC object incorporates the above stages into a single entity, this then allows for the necessary functions to be centrally located, these are split up so that the generator and solver can be run independently from each other or one after the other from one function call.

7.3 Designer Interaction

During runtime, the user can alter key features of the system to change how content is generated. The first is the size of the grid, this ranges from grid sizes of 2x2 up to 20x20 so that different sized cities can be generated. This also means that bigger grid sizes result in more complex cityscapes as they are composed of a greater number of pieces and therefore have a greater chance to vary in pieces as well as layout.

Taking inspiration from Stålberg's implementation, the algorithm depicts the possible tile set of each of the cells which can either be used for a visual aid to demonstrate what happens within each iteration as well the impact this has on a cell's neighbouring tiles. The system also allows for users to select a tile for all the cells within the grid so that each cell can be specified. This is done through the use of a tile selection script. When the user clicks on a tile the script projects a raycast towards it, it then converts the world coordinates of the hit point, converts the position to a cell index to determine the targeted cell and then assigns the hit cell to the selected tile. The result is that users can actively alter the way in which the grid is solved by selecting the chosen tiles into the grid and then allowing the system to solve the remaining cells with consideration of the users input (Image 17).

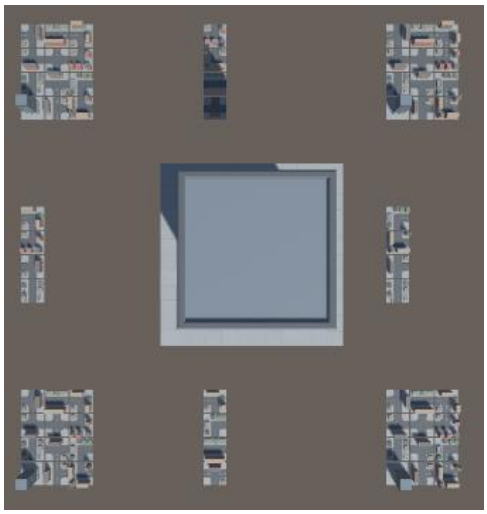

Image 17: The centre cell collapsed by the user and the impact of this decision on the neighbouring cells' possible tile sets.

Demonstrating each cell's possible tile set is combined with the use of a coroutine to execute the solver component of the WFC algorithm with a variable delay, so that users can speed up or slow down the rate at which the solver solves the grid. This provides users with the chance to get a better idea of the steps taken to produce the final cityscape.

7.4 Results

Image 18: A procedurally generated cityscape created with this project's variation of the WaveFunctionCollapse algorithm.

The final resulting system, implemented within the Unity game engine, is one that procedurally generates a 3-dimensional cityscape by using a set of tile prefabs. Users can alter the system in a variety of ways that affect how the final output of the system in ways such as altering the size of the resulting cityscape or by explicitly assigning tiles to cells.

This system can however be utilised for a variety of different circumstances besides that of city generation due to its tile-based approach. It is therefore possible for a designer to change the tile set that is being used to anything they desire. The system initially started out with a grass and path-based tile set with which was then swapped for the final city-based tile set, to better reflect the main goal for this project. Because of this flexibility, this system could theoretically produce vastly different content for a variety of situations, based off the given input tile set. One hinderance to this however would be the need to access some of the scripts as some functionality such as the values of sockets are hard coded into the system.

8. Discussion of outcomes

Unlike Karth and Smith's (2017) findings, this project did not treat the WFC algorithm as a metaphorical black box and instead attempted to fully understand it's workings so that it could be altered and implemented in the project's own way. The result is a system that, whilst taking heavy inspiration from the original algorithm, is not implemented in an identical manner to the work of Gumin (2016). The project instead approaches the algorithm in a similar manner to the work conducted by Donald (2020) and Mariam42 (2019) in which numerous tiles are created with the user specifying how they should connect rather than using an input image to determine connectivity.

8.1 Evaluating the overall effectiveness of the project

With Kelly and McCabe's (2006) criteria in mind, the overall system is fairly effective at producing procedurally generated content.

The realism can be somewhat questionable with the generated cityscapes at times due to unrealistic road networks being created. Since the system chooses valid tiles at random, it does not consider the value of neighbouring tiles and as such can generate sections such as in Image 19 where junction tiles have been densely placed in a local space. One method to solve this lack of realism could be to introduce a weighting to each of the tiles to alter the chance of each of the tiles being selected or by adding more varied tile types.

Image 19: A cityscape containing a dense area of junctions

The scale can be altered meaning that the size can be suited to the need for any sized city needed, however increasing the grid size can cause framerate issues due to the processing power needed to generate the larger structures and to render all the meshes for each of the tiles. This problem could be mitigated by taking a similar approach to that of Sunset Overdrive (Insomniac Games, 2014). Elan Ruskin (2015) presents the way in which the game optimised the world for streaming. Each chunk could be streamed in so that the system is not rendering the entire map at once or alternatively a level of

detail system could be used to reduce the vertex count of meshes that are less important, both of which would help to reduce the number of draw calls the system must make.

The system can generate a variety of layouts for the cityscape which can be further increased by adding to the possible tile set, the level of variation of the system can therefore be chosen by the designer with the number of tiles that they provide the system. The minimum input data required to maintain the system is a tile set and the ruleset of how these tiles connect. The system can generate smaller grids with relative ease and speed with larger grids taking longer to generate due to the increase in tiles needed to be solved and drawn to screen.

Designer control was a core focus through the project's development since this was the best way to test the extent to which the algorithm could be altered as well as generally creating a system that gives a lot of control over how it operates to designers to be tailored to their needs. As such, the system allows for multiple ways in which to influence the content generated. Tile sets can be swapped out produce different aesthetics and users are able to manually place tiles that will still adhere to connectivity logic. Alongside being able to alter the scale, these were the main features that the implementation intended to have throughout the project that allows for customisation and alteration.

The generated city can be viewed in real-time, but this is limited to the entire thing being generated at once, unlike Scholz's (2019) or Mariam42's (2019) implementation, the system does not allow for infinite generation. This could have been achieved by using a chunk-based system that considers the edges of the connecting tiles, the initial problem with attempting this is that WFC is aimed at constraint solving which would be hindered by infinite generation. The same optimisations that would benefit the variable scale would also help to solve the issues to allow for real-time infinite generation,

After considering these points, the overall WFC system is effective at producing procedurally generated content but has room for improvement, it successfully meets the design criteria for a successful system but only just, if the suggested improvements were made then the system would better meet the criteria and could be made more effective.

The project was unable to implement tile prototyping, and as such the way tiles are created could have been simplified and more efficiently handled. The lack of prototyping resulted in overhead data being needed. Tiles that were conceptually the same but with a different rotation required a completely new tile being created, the horizontal and vertical road pieces for example are identical if appearance and logic but have a rotation difference of 90-degrees. If the project were able to implement tile prototyping, then any duplicated tiles could be grouped into a single core tile with some variable difference (I.e., different rotations). Prototyping would involves capturing the core data of the tile needed to be able to recreate it such as its rotation, the data from each of its sockets and the mesh data. Since each tile was composed of more than one single asset, a problem arose with trying to replicate the tiles mesh data since there was no one single mesh file to load from. As such, prototyping was not implemented in the final project.

8.2 How well does the project respond to the research questions?

The product created as a result of this project demonstrates that the WFC algorithm is effective at producing procedurally generated content. Throughout production the project implemented two versions of the algorithm, once with an example input image and one without, both of which were able to produce distinctively different content in terms of appearance and layout. Analysed against a set of criteria, this form of procedural generation excels in some areas such as in variation or control but is lacking in other areas such as its limitations with real-time generation or the scale of the generated content, this may however be a result of the implementation rather than the actual algorithm since the project does not recreate the algorithm perfectly.

This implementation also demonstrates that whilst WFC can allow for some form of parameterisation, this is limited and as such the project did not really become the tool that it was intended to become, which is only furthered by the fact that most of the system runs at runtime due to the need for project to be an executable build for demonstration purposes. The algorithm alone does not allow for much customisation, or at least not in the way that this project implemented it, as such the system would benefit from further development to provide additional system on top of those already created that would, when put together, allow for in depth customisation for a range of customisation. Additional tools such as being able to classify regions with a specific label such as commercial,

industrial or residential would allow designers to further specify the type of tiles that must be placed within a certain area and provide a more realistic portrayal of building diversity and distribution. Another such potential improvement may be to focus on the 3-dimension aspect more since this project only really covered a 2-dimensional grid, by introducing a 3d grid the system could make use of sloped tiles to allow for changes in height to resemble non-level ground such as that found in San Fransisco, America which can be heavily sloped in some areas.

Whilst research conducted has shown that it is possible, this project failed to answer whether the algorithm can produce organic-looking cityscapes since the final content generated was more structured. This was due to the tiles used by the system as these were square tiles that used straight pieces, these could be made to look more organic by either adding more curves to the content of the tiles to give the appearance of a more organic feel or this could have been achieved by changing the shape of the tiles. By using tiles of a hexagonal design would have allowed for a larger number of connections and thus more curves. This was not implemented due to the system using a square grid pattern that only used the four cardinal directions, by transitioning over to hexagonal tiles the system would require more connection directions which in turn would require both a new method to place cells within the grid layout as well as an alteration to how neighbouring tiles are calculated. This would require two more neighbours to be found and the logic of finding the neighbours would need to be changed to check in a diagonal direction.

9. Conclusion and recommendations

Hopefully, this project has demonstrated the power of the WaveFunctionCollapse algorithm and provided one such possible method to implementing the algorithm to those looking for a place to start.

From here, this project would make several recommendations for future work. One such recommendation would be to simplify and/or improve upon the codebase of the project and to generally improve upon how the core features of the algorithm are implemented. For example, since the Tile script is a MonoBehaviour it must be attached to a game object within the scene. This caused significant problems during the production stage when attempting to get the tile component of a neighbouring cell that had yet to be instantiated. The project was also unable to determine a possible method for allowing a user to create their own list of socket values for the enumeration which results in them having to access scripts which could be troublesome. Improving upon the areas in which the project was unable to meet would improve the tool to be more user friendly for designers and overall allow for a wider range of implementations. By finding a more efficient method of implementing the core features of this system would also lead to more features being integrated as an overall benefit.

Overall, the project spent more time than it should have in the initial practical stages since the research conducted failed to identify the necessary steps needed to be taken to implement the algorithm. However, this system could potentially be used by others as a basic implementation in achieving a procedural layout of tiles or act as a rough guide to those looking for the steps needed to be taken to achieve a similar outcome to that of this project.

10. References

Donald, M., (2020) *Superpositions, Sudoku, the Wave Function Collapse algorithm*. Available from: https://www.youtube.com/watch?v=2SuvO4Gi7uY

Freehold Games, LLC (2015) *Caves of Qud*. [Video Game] Freehold Games, LLC. Available from: https://www.cavesofqud.com/

Galindo-Torres, s., and Muñoz, J. (2010) *Minkowski-Voronoi diagrams as a method to generate random packings of spheropolygons for the simulation of soils.* Physical Review [online] 82 (5). Available from: https://journals.aps.org/pre/abstract/10.1103/PhysRevE.82.056713#fulltext

Gumin, M. (2016) *Wave Function Collapse Algorithm*. Available from: https://github.com/mxgmn/WaveFunctionCollapse

Insomniac Games (2014) *Sunset Overdrive*. [Video Game] Xbox Game Studios. Available from: https://insomniac.games/game/sunset-overdrive/

Karth, I. and Smith, A.M. (2017) *Wavefunctioncollapse Is Constraint Solving in the Wild*. FDG '17: Proceedings of the 12th International Conference on the Foundations of Digital Games [online]. Available from: https://adamsmith.as/papers/wfc_is_constraint_solving_in_the_wild.pdf

Kelly, G. and McCabe, H. (2006) *A Survey of Procedural Techniques for City Generation*. The

ITB Journal [online]. 7 (2) Available from: https://arrow.tudublin.ie/itbj/vol7/iss2/5

Marian42 (2019) *Infinite procedurally generated city with the Wave Function Collapse algorithm*. Available from: https://marian42.de/article/wfc/

Minor Key Games (2013) *Eldritch*. [Video Game]. Minor Key Games Available from: https://eldritchgame.com/

Mojang studios (2011) *Minecraft*. [Video Game]. Mojang studios. Available from: https://www.minecraft.net/en-us

Parish, Y.I.H. and Müller, P. (2001) *Procedural Modelling of Cities*. SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques [online]. Available from: https://dl.acm.org/doi/pdf/10.1145/383259.383292

Raw Fury (2018) *Bad North*. [Video Game] Raw Fury. Available from: https://www.badnorth.com/

Robertson, H., (2018) *Procedural Regeneration: Matching the World to the Player*. Game Developers Conference 2018. [online] Available from: https://www.youtube.com/watch?v=rf4VaRldwOY

Ruskin, E., (2015) *Streaming in Sunset Overdrive's Open World*. Game Developers Conference 2015. [online] Available from: https://www.gdcvault.com/play/1022268/Streaming-in-Sunset-Overdrive-s

Santell, J (2019) *L-systems*. Available from: https://jsantell.com/l-systems/

Scholz, D (2019) *Tile-Based Procedural Terrain Generation*. Bachelor of Science, Vienna University of Technology. Available from: https://www.cg.tuwien.ac.at/research/publications/2019/scholz_2017_bac/scholz_2017_bac-thesis.pdf

Short, T. and Adams, T. (2017) *Procedural Generation in Game Design* [online]. Taylor & Francis Group, LLC. (pgs. 9-12)

Smith, G. (2015) *An Analog History of Procedural Content Generation*. Foundations of Digital Games. 2015. Available from: http://www.fdg2015.org/papers/fdg2015_paper_19.pdf

Stålberg, O. https://oskarstalberg.com/game/wave/wave.html

Stålberg, O., (2020) *Townscaper*. [Video Game] Raw Fury. Available from: https://www.townscapergame.com/

Sun, J., Yu, X. and Baciu, G. (2002) *Template-based Generation of Road Networks for Virtual City Modelling*. VRST '02: Proceedings of the ACM Symposium on Virtual Reality Software and Technology [online]. Available from: https://dl.acm.org/doi/pdf/10.1145/585740.585747

Togelius, J. and Shaker, N. (2016) *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems [online]. Springer, Cham. Available from: https://link-springer-com.ezproxy.uwe.ac.uk/chapter/10.1007/978-3-319-42716-4_1

University of Bristol, School of Mathematics. *What is a Voronoi diagram?* Available from: https://www.bristol.ac.uk/maths/fry-building/public-art-strategy/what-is-a-voronoi-diagram/

11. Bibliography

Bucklew, B., (2019) Math for Game Developers: Tile-Based Map Generation using Wave Function Collapse in 'Caves of Qud'. Game Developers Conference 2019 [online]. Available from: https://www.gdcvault.com/play/1026263/Math-for-Game-Developers-Tile

DigiDigger. (2020) How does procedural generation work? | Bitwise. Available from: https://www.youtube.com/watch?v=-POwgollFeY

Emilien, A., Bernhardt, A., Peytavie, A., Cani, AP., Galine, E. (2012) Procedural generation of villages on arbitrary terrains. The Visual Computer. 28. [online] Available from: https://link.springer.com/article/10.1007/s00371-012-0699-7

Freiknecht, J., Effelsberg, W., (2017) A Survey on the Procedural Generation of Virtual Worlds. Multimodal Technology. 1(4). [online] Available from: https://www.mdpi.com/2414-4088/1/4/27/html

Gaisbauer, W., Raffe, W.L., Garcia, J.A. and Hlavacs, H. (2019) Procedural Generation of Video Game Cities for Specific Video Game Genres Using WaveFunctionCollapse (WFC). CHI PLAY '19 Extended Abstracts: Extended Abstracts of the Annual Symposium on Computer-human

Interaction in Play Companion Extended Abstracts [online]. p. 397–404. Available from: https://dl-acm-org.ezproxy.uwe.ac.uk/doi/10.1145/3341215.3356255

Hendrikx, M., Meijer, S., Van Der Velden, J., losup, A. Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications. 9(1). Available from: https://dl.acm.org/doi/pdf/10.1145/2422956.2422957

Pittman, D. Procedural Level Design in Eldritch. Game Developers Conference 2015. [online] available from: https://www.youtube.com/watch?v=BYN0PJOdvzs

Sharma, R. Procedural City Generator. 2016 International Conference System Modeling & Advancement in Research Trends (SMART). https://ieeexplore.ieee.org/abstract/document/7894522

Smith, G. (2014) Understanding procedural content generation: a design-centric analysis of the role of PCG in games. CHI '14: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. [online] Available from: https://dl.acm.org/doi/abs/10.1145/2556288.2557341

Appendix A: Assets used in the Project

POLYGON City - Low Poly 3D Art by Synty – Available from: https://assetstore.unity.com/packages/3d/environments/urban/polygon-city-low-poly-3d-art-by-synty-95214#description