TECH DEMO FOR PROCEDURAL GENERATION IN VIDEO GAMES

CALUM MCADAM

H00271918

CM140@HW.AC.UK

BSC (HONS) COMPUTER SCIENCE

FINAL YEAR DISSERTATION

SUPERVISED BY: MURDOCH GABBAY

SECOND READER: ANDREW IRELAND

HERIOT-WATT UNIVERSITY

SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

OCTOBER 2020

Abstract

Modern game development is both costly and time consuming. With games becoming larger and consumer expectations becoming greater, there is an ever-increasing demand on video game developers. Many studios are now utilising Procedural Generation to quickly create useable assets.

This purpose of this project is to Demonstrate how procedural generation can not only quickly generate environments but can also create them to be functional while not being identical to each other. This will be done through a tech demo that will use procedural generation to generate an environment or level.

Declaration

I, Calum McAdam confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed Calum McAdam

Date 10/12/2020

TABLE OF CONTENTS

INTRODUCTION 4

BACKGROUND 5

- **WHAT IS PCG** 5
- **ADVANCEMENTS OF PCG** 5
- **MODERN EXAMPLES OF PCG** 7
  - **MINECRAFT** 7
  - **DEEP ROCK GALACTIC** 7
  - **SPORE** 8
- **INDIE GAMES** 9
- **BEYOND GAMES** 10
  - **PLANNING** 10
  - **FILMS** 11
  - **MUSIC** 11
- **FAILINGS OF PCG** 12
  - **UNIQUENESS** 12
  - **MASS PRODUCED VS HANDCRAFTED** 12
- **METHODS OF PCG** 13
  - **NOISEMAPS** 13
  - **BINARY SPACE PARTITIONS** 13
- **PATH FINDING** 14
  - **A*** 14
  - **DEPTH-FIRST & BREADTH-FIRST SEARCH** 15
  - **DIJKSTRA'S** 15
- **SUMMARY** 16

REQUIREMENTS 17

PROJECT MANAGEMENT PLAN 19

- **LEGAL, ETHICAL AND SOCIAL ISSUES** 19
- **RISK ANALYSIS** 19

EVALUATING 20

DESIGN 20

- **OVERVIEW OF WHATS PLANNED** 20
- **ENGINE CHOICE** 21

Implementation 21
Biome Pre-set 22
Calculating the Biome 23
Enemy and player 25
Evaluating and testing 25
Requirements and results 25
Hardware Testing 28
Effectiveness of project plan 28
Identified Risks 28
Unidentified Risks 29
Conclusion 29
Success of Procedural Generation 29
Future Developments 30
Improved combat 30
Interactable environments 30
Infinite world 31
REFERENCES 31
APPENDIX 32

INTRODUCTION

The purpose of this project is to produce a tech demo for procedural generation in video games. The demo will show a 2d environment created through procedural generation, instead of everything being created by hand. The finished project will be presented as a playable 2d environment.

Procedural Generation (PG) or Procedural Content Generation (PCG) refers to the creation of content using algorithm rather than having it be created manually. This allows content to be created automatically which allows for work to be greatly reduced. PCG can be used to create a variety of content, such as animals, planets, music. To name a few.

We will consider the use of Procedural Generation in video games from its first use within games up to how it is being utilised in modern gaming. I aim to show how PG was used in the creation of their games as well as to show why they were being used. While doing this I will look at how the use of PG has changed throughout the years as well as how it may evolve in the future.

In this report I will be discussing the background and current use of PG as well as the benefits, advantages, disadvantages, and implementations.

BACKGROUND

WHAT IS PCG

PCG or procedural content generation is the method of algorithmically producing content either before a level start or continuously during runtime. One of the earliest recorded uses was in the rogue like game developed by quality software called “Beneath Apple Manor”(1978)[2], The game used PCG to dynamically create the levels, this was done by using ascii or regular tile based systems to define the room as well as the monster hallways and treasure.

This helped immensely back in the 70’s as with the significantly older technology meant that the creators suffered from lack of memory when creating games. however, using PCG meant there were no quality checks to what was created, often meaning that generated room were confusing and hard to navigate. Two years the release of BAM, and another game release was set to continue the trend of using PCG in rouge likes.

Rogue (also known as ***Rogue: Exploring the Dungeons of Doom***) was released in 1990[3], however this game was released with technology that was not related to BAM. With rouge being released as freeware it became immensely popular, showing that Procedural Content Generation can be effective tool to create levels.

FIGURE1: BENEATH APPLE MANOR

ADVANCEMENTS OF PCG

Finding PCG in rouge like games became normalised after Bam and rouge, this trend continued on with games such as Moria(1988)[4] , which was a Lord of the Rings rogue-like game which added the generation of towns with PCG , these towns would contain a number of defined buildings such as a store, armoury, weaponsmith, etc. the trend of PCG rogue like games still continues with Hades(2020)[5]. Hades uses rouge like element and PCG to generate the rooms that the player can move between, like how it is done in Rouge

After the success of theses games many more developers were willing to create more games using PCG, Games such as The Elder Scrolls II: Daggerfall(1996)[6] was realised using PCG to generate an ambitious world with a playable area of 161,000km squared[7] .

To put that in perspective that is over half the size of the United Kingdom, unlike Bam or Rogue which use a tile-based system to define the environment, Daggerfall chose to use pre-set blocks to generate towns and dungeons, also unlike the prior games these were generate while the game was

running instead of during the loading of the level. However where this game truly stands out when using PCG is how it generates not only the environment but also that enemies, NPC'S, Towns and weapons were all having PCG used to generate them.

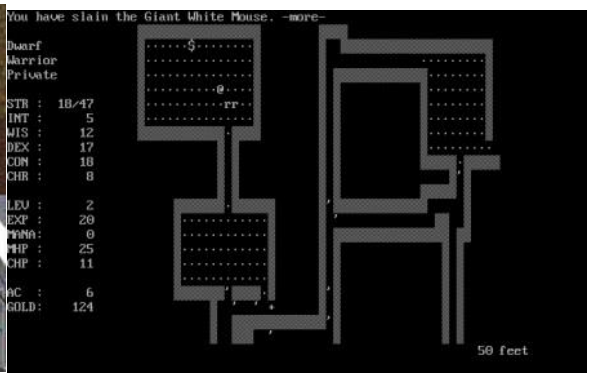


Figure2: Elder Scrolls Daggerfall Figure3 : Moira

Another system that was used in the late 80's was the use of Pseudo random number generators, the were used with seeds that were predefined to generate vast worlds that appeared premade. The Sentinels(1987)[8] was a game that used this to create up to 10000 levels that were store in up to 64 kilobytes .

A Larger example of the pseudo random generator would be Elite (1984)[9] which is a space exploration simulator which creates a realistic 1:1 scale of the Milky Way, however to store a 1:1 scale of the milky way would require a monumental amount of storage, that is why the developers turned to Procedural generation to easily generate each of the systems without it needing an extraordinary amount of space, much like sentinels Elite used a seen Number which was then run with a pseudo random number generator which would produce the co-ordinates of every planet.

Similar technology was used again more recently with the Kickstarter game Elite Dangerous (2014)[10]. This game was a successor to the 1984 title Elite, which also intended to recreate the milky way with up to 400 billion systems.

Figure4 : Elite (1984)

MODERN EXAMPLES OF PCG

MINECRAFT

Minecraft (2011)[11] is one on the most well-known uses of PCG in video games, I has captured the hearts of many with its endless terrains and plethora of biomes. Allowing users the chance to explore an endless world and delve to the darkest depths.

Minecraft uses Perlin Noise calculation. It start out by just painting the topographical map and the it uses noise maps to fine tune the terrain by adding details such as lakes shrubbery and animals[12].it generates the map chunk by chunk (16*16*256) with noise maps on each face of the chunk. The game also needed to store multiple biomes and generate the different terrain types for each of the biomes. Aswell as mapping out the caves below. Minecraft is a vast game with PCG being utilised to create a near endless world with almost no handcrafted elements.


Figure 5: Minecraft

DEEP ROCK GALACTIC

Deep Rock Galactic (2018)[13] is a first person co-op “looter shooter “where the players delve into a expansive Procedurally generated cave system filled with enemies and resources.

These caves are procedurally generated to give the sense of exploration and discovery, they are made using simple primitive shapes that their procedural system “dress” with rocky surfaces and debris as well as procedurally places resources and other geological details. Then theses caves are randomly chosen and connected via a series of tunnels that can range from tight gaps to enormous chasms [14]

Figure 6 : Deep rock galactic

SPORE

Spore (2008)[15] is a game that follows and control the evolution of a species from its beginning as a microscopic organism all the way to a space faring civilisation, this is broking down into 5 stages: Cell stage, creature stage, tribal stage, civilisation stage, and space stage.

Spore central mechanic is the procedural generation of creatures at every stage from how they look as a cell to how they appear in their spaceship, they use a combination of procedural generation as well as procedural animation. The game also creates its own music using procedural generation depending on how the player creates their species, if its more aggressive their will be an aggressive ton, if mor social then a social tone[16](note that there is no lyrical composition within the music in spore)

Figure 7 : Spore

INDIE GAMES

PCG have been used very advantageously in triple A studios. However , procedural generation is not just exclusive to the wealthy developer. PCG is vital to many low budget games as a way of bypassing hiring additional staff and the continuous rise in costs to make a game

One of the most well known indie games that utilises Procedural generation is FTL(2012) which uses ftl to define the sectors that your ship can fly to, how each sector connect to one another and the start and end point of each sector

Figure 14: FTL

Another use of PCG is in KKrieger(2004) which is on of the best examples of PCG, developed as part of a 96K game development competition , KKrieger uses PCG techniques such as txture generation to create a fully procedural game with procedural sound track and procedural enemies

Figure 15:KKrieger

BEYOND GAMES

PLANNING

PCG can be used for more than just games, there is a extensive field of utilising PCG in areas such as urban planning, films and music. A planner can create the most vibrant and safe city but there is no real way to test how people will interact with the city and each other, PCG is used to bridge this gap by generating large quantities of pedestrians and vehicles to see how the flow of traffic is able to move around the city and to find key areas of congestion[17]

Figure 8: Crowd Simulation

FILMS

Similarly, to how PCG can be used to simulate group of people in towns and cities it can also be done for background characters movies in movies, this was used to affect on movies such as “lord of the rings” were large groups of orcs need to be depicted on screen[19]. PCG can also be used to to create large cityscapes for films to reduce production cost and time[17]

Figure 9 : procedural generation in LOTR

MUSIC

We have already discussed how music was procedurally generated within Spore (2008)[15] and how that is able to take something that the user did and create music from that[16]however it is possible to not only generate music outside of games unsupervised but also to create a lyrical composition that can try and imitate a song as if made by a human[18]

FAILINGS OF PCG

UNIQUENESS

One of the most notorious uses of PCG Is No Mans sky (2016)[19], the games tagline was an endless, universe to explore that was entirely created using PCG, And games like this have done similar things and were perceived well, such as Elite. However, on release what was meant to be its greatest feature became its main downfall. The content that was created lacked any uniqueness, every planet felt the same, every animal acted and looked similar. Their was very little that brought the game to life.[18]

Figure 10: No man's Sky

MASS PRODUCED VS HANDCRAFTED

Borderlands 3(2019)[20] has one of the largest number of guns a player can find within any game, with over 1 billion guns that have been Generated procedurally, however many of these weapons have its own purpose and are just copies of another, in contrast there are 21 handcrafted weapons that the player can use In Apex Legends (2019)[21] each with different mechanics and purpose within the game, This can also be seen with procedurally generated levels. Where they may have been generated In an unexpected manor where as the hand crafted level is always going to be at that consistent standard.

METHODS OF PCG

PCG have been used within gaming for almost 40 years , within that time there have been a variety of algorithms that could be used, however to narrow it down I'm only going to be looking at two.

NOISEMAPS

Noise maps are an excellent way to procedurally generate terrain. An example of a noise map is perlin Noise, it was originally used within the Tron movies by ken Perlin. Noise maps are still used to generate content such as in Mincraft[11] which uses the maps to generate the terrain and cave systems. Many games use multiple noise maps, such as Minecraft which uses multiple maps to create the terrain, animals, resource[12].There are now many different game engines which creates these maps for the developers, making It much simpler for the creation of PCG within games[22].All the developers will need to do is entering the parameter that they want and layering them above one another. Noise maps are very suited for the generation of terrain as they can be easily configurable and easy to set up. in contrast, manually constructing terrain can be time consuming for the developers which in turn the advantages of using PCG seem greater.

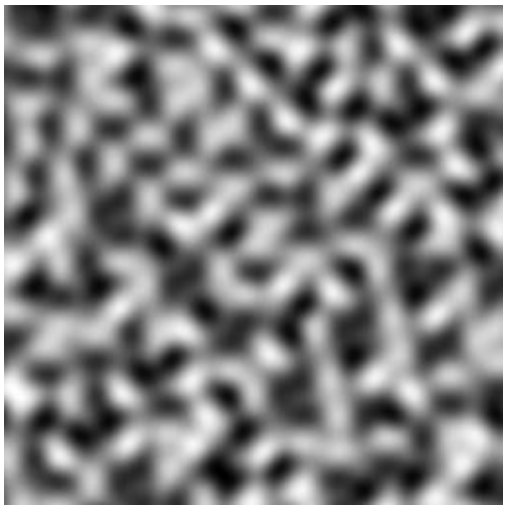


Figure 11: Noise Maps

BINARY SPACE PARTITIONS

Binary Space Partitions(BSP) is another form of PCG, generally used when storage is a concern's generates a level by setting up the maximum size for an area and then recursively divides the area into a pair of cell, this method creates a more random level however it Is not great at creating a more organic shape [23]

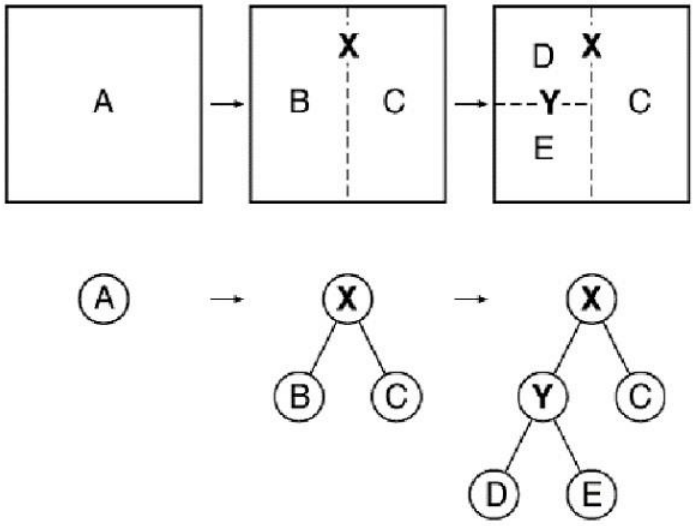


Figure12: Binary Space Partitions

PATH FINDING

Sometime when using PCG the level may not be to a correct standard so we need a wayto judge the certain paths of a level.one way to do that would be if the level could be stored within a graph and then running a Path finding algorithm on it. These will take in the path and return the number of paths that can be achieved from it. I will be looking at three path finding algorithms:

A*

A* is the most popular search algorithm being used within modern games. The search algorithm that repeatedly examines the most promising unexplored location seen, once a location is explored it takes note of all its neighbours for further exploration. The algorithm finishes once its goal node has been explored. A* has certain properties that can be very useful, first is that it is guaranteed to find a path from the start node to the goal if there is a path to be found. It also make the most use out of the heuristic [24]

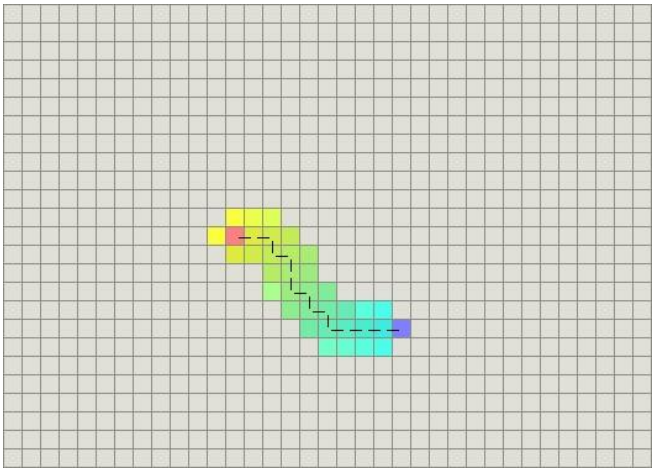


Figure 13: A* in use

DEPTH-FIRST & BREADTH-FIRST SEARCH

Depth first search travels through the graph a layer at a time, for each node its visits it will then visit it children and then it will visit the children's children until it has reached every node possible25

Depth first is the opposite, instead of visiting each of the children it will take a direct path until there is no more children to visit. Then it will back track to each node and continue a single path until there is no more children. It will continue this process until all nods have been visited[25]

DIJKSTRA'S

Dijkstra's algorithm finds the shortest path by traversing a graph and giving weights to each edge, it will the find a path from the start to the goal with the lowest accumulative weights. This is the most common pathing algorithm used in game development and is always guaranteed to fin the shortest path, if there is one[26]

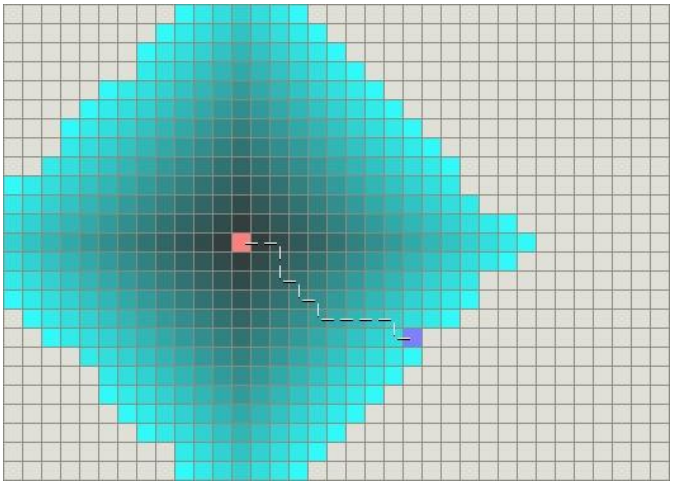


Figure 14: Djikstra's algorithm in use

SUMMARY

In the 40 years since its first use with video games PCG has been pushing the boundaries of how much can be achieved within games, It has been used to develop several games that would have not been possible without Procedural Generation and will continue to go on to create even more ground-breaking results. However, PCG should not be relied on too heavily, as show it has several key flaws and that making a game with PCG does not guarantee that the game will be successful

REQUIREMENTS

I have created this table to show the system requirements, the requirements were set using the MoSCoW method. This will be used as the core of the project as well as a guideline to what must be done and any additional that can be made

	Requirements	Priority	Requirement Type	Status	Justification
1.	The project must be a fully playable tech demo	Must	Functional	N/A	The project must be playable to meet the main aim of this project
2.	The project must generate a full traversable 2d map	Must	Functional	N/A	To allow the player to explore the map and see the procedural generation
3.	The 2D map must be made using Procedural Generation	Must	Non-Functional	N/A	This project is being made to sho procedural generation
4.	The project must not take too long to load	Must	Non-Functional	N/A	To create a better experience
5.	The project must not perform poorly	Must	Non-Functional	N/A	To allow for adequate experience will using
6.	The project must not contain any graphical errors	Must	Non-Functional	N/A	To create a better experience
7.	The project must contain a mini-map	Must	Functional	N/A	This will allow the user to travesre the map more easily
8.	The project must be stable , with minimal crashing	Must	Non-Functional	N/A	To ensure a good experience for the user
9.	The project must feature a variety of biomes	Must	functional	N/A	To show variety and generation of different terrain types
10.	The project must implement a noise map in the generation of terrain	Must	functional	N/A	The main focus for the PG will be on the noise map

11	The project must show the user as an avatar in the tech demo	must	Non-Functional	N/A	To help the user travers the map
12.	the project should generate some form of inaccessible land such as water	should	Non-Functional	N/A	To allow for more elaborate pathing
13.	The project should generate resources such as tree, etc	Should	Functional	N/A	To allow for additional noise maps to generate more
14.	The project should have some form of enemy	Should	Functional	N/A	To PG the enemy locations within the map
15.	The project should generate cave levels	Should	Functional	N/A	To add another level of PG
16.	The cave levels should be generated with procedural generation	Should	Non-Functional	N/A	Main reason of the prooject
17.	the project should be able to feature multiple sizes of maps	Should	Functional	N/A	To analyse how the PG handles the variety of sizes
18.	The project should have a start and end point	Should	Functional	N/A	To give a clear objective
19	the project should have some form of npc	Should	Functional	N/A	An extension on enemies
20.	Resources within the project could be gatherable	could	Functional	N/A	Show case traditional gaming concepts
21.	The project could have some form of inventory	Could	Non-functional	N/A	An extension of gathering resources
22.	The project could have shops procedural generated	Could	Functional	N/A	This will need more PG to create
23.	If there is shops the project could have a town	Could	Non-Functional	N/A	An extension of the shops
24.	If there is a town then the project could have npcs that are local to that town	Could	Non-Functional	N/A	An extension of the shops

25.	The mini map could contain markers	Could	Functional	N/A	To aid in traveling through the map

PROJECT MANAGEMENT PLAN

To open the Gantt chart for the project, click on the icon below.

Gantt Chart.pdf

LEGAL, ETHICAL AND SOCIAL ISSUES

There will be no ethical issue during the development of the project the demo will be evaluated algorithmically. This will be done to evaluate the configurations of the software, when testing this environment, I will be looking to test the functionality and I have chosen to do this with algorithms

Procedural content generation does not relate to any social or ethical issues as all it does is allow developer to create content algorithmically.

RISK ANALYSIS

In production of this project there is many potential risks and it is important to recognise these risks and take the necessary steps to prevent or minimalize their effect o the project

In the following table I have listed the potential risks to the project and the steps I will take to avoid or mitigate the effect

Risk	Probability	Severity	Effect	Steps taken
Bugs in the development of tech demo	High	High	Longer develop times, possibility of incomplete task	Create a foundation within code and test regularly
Unable to finish art assets	Low	High	Incomplete project	The use of grey box assets for demonstration

Poor Hardware Performance	Low	Low	Longer loading times	Use methods to improve performance
Personal issues of developer causing loss of time	Medium	Medium	Time to develop will be reduced	Focus on the main objective , don't start should or could until met
Hardware Failure	Low	high	Loss of work	Through the use of cloud storage as well as making regular backups

EVALUATING

The tech demo will be evaluated on how well it meets the system requirements that were laid out in the requirements section, I will also be evaluating how well each map's graph using a pathing algorithm,

Once I have gathered the results on such a graph I will average all the results for the maps and evaluate it using a fitness function and return a score indicative of the performance, since this is a tech demo I also intend to preform hardware testing to see if the tech demo will be performing acceptable in game.

DESIGN

OVERVIEW OF WHATS PLANNED

When planning the system that will produce the maps, the previous methods listed were considering. The core of the tech demo will be to generate a 2d environment from predefined arguments.

I will be using noise maps as the core for the algorithm, the initial noise map will be used to define the terrain, then I will be applying several more layers on t that to define the Resource, Buildings, Biomes, caves etc.

I will then be running a pathing algorithm marking out each of the areas within the map, such as cave entrances, start point, end point.

ENGINE CHOICE

The first decision when planning out this project was what engine the tech demo was going to be made in; the two main engines I have focused on are listed below.

Engine	Unreal engine	Unity
Language	C++	C#
Difficulty	Easy	Easy
Advanced Graphics Support	Yes	Yes
Memory Management	Manual	Automatic
Cost	Free	Free

The main factor that leads me to choose these two out of the reast was the ease of use as well as them being free of charge to use.

When deciding between the two I opted for Unity over Unreal, this is mostly due to me being more familiar with unity and C# over unreal and C++.I chose familiarity over unreal as Time constraints were mention above in risk management as being a possible major effect on the project.

The automatic memory management that unity offer will also be crucial when generating the maps, as the data will be dynamically created and varied, without the automatic memory allocation the risk of errors occurring within the project increase

Implementation

The demo has been implemented within c# and with an object orientated programming style. The was done due to the need to dynamically generate objects.

Noise Maps

the noise map algorithm returns a number between 0 and 1 based on an input, the input is defined as a 2D plane “Noise Map”, which then displays a number from 0 to 1 with 0 being black and 1 being

white. The noise maps correspond to an aspect of the map, these are Altitude, temperature and humidity.

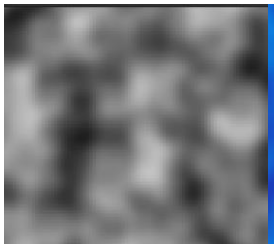


Figure 15.

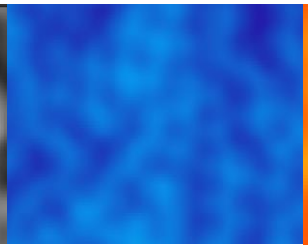


Figure 16.

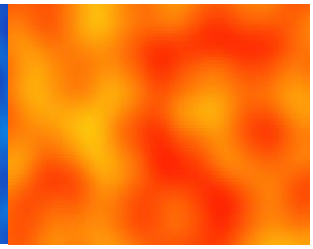


Figure 17.

As you can see from Figures 15, 16, and 17 these are Noise maps for the height moisture and heat, Respectively. Each of these maps generate different noise,they are then attached to a raw image within a canvas which can flip through each of them through a dropdown box, finally each map has been given a suitable colour to make them distinctive and the difference more apparent , however this is mostly an aesthetic feature .

The noise map is generated by storing the width and height of the map and then iteratively going through each a and y co-ordinate. At ever co-ordinate the unity function mathf.perlinnoise is called , this will generate a number between 0.0 and 1.0, this will then be stored within a 2d array.

However, this will generate no cohesion between each of the noise, to fix this a new Wave class was made, this would have the properties of seed, frequency and amplitude. Frequency will determine how far a part the wave is or the distance between white and dark areas, whereas amplitude determines the intensity of the waves or how smooth or jagged the transition from white to black is

This can be seen in figure 17 where there is a high amount of frequency and figure 19 where there is high amounts of amplitude

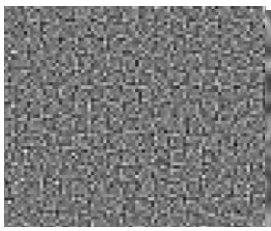


Figure 18

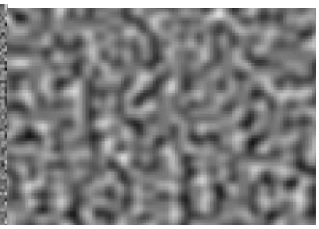


Figure 19

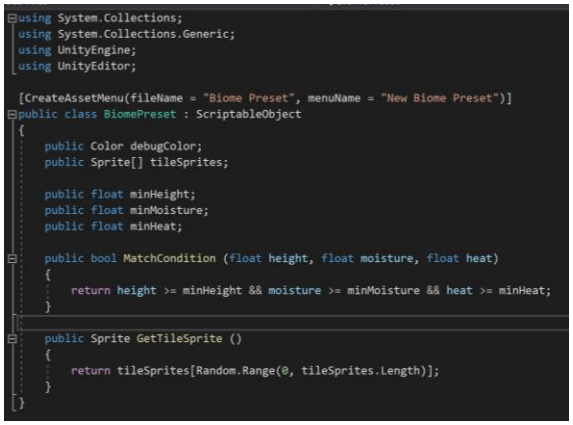
Biome Pre-set

The idea of having different biomes generated within the map was one of the main feature that was going to be implemented, however to do this, there was a need to have a pre-set where different biomes can be generated from. This was done by creating a new Scriptable Object which allowed me to create ne biomes that already have specific properties that could be assigned to them, without them being attached to any game object.

The properties that could be given to each biome were the minimum Height, minimum moisture and minimum heat, the idea behind this was to ensure these requirements were to be met before the tile can be generated there,

As you can see in figure 15, the get tile sprite method this will be used to randomly select a predefined tile that has been selected for the specific biome.

The predefined tile are store within the tile Sprites array



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;

[CreateAssetMenu(fileName = "Biome Preset", menuName = "New Biome Preset")]
public class BiomePreset : ScriptableObject
{
    public Color debugColor;
    public Sprite[] tileSprites;

    public float minHeight;
    public float minMoisture;
    public float minHeat;

    public bool MatchCondition (float height, float moisture, float heat)
    {
        return height >= minHeight && moisture >= minMoisture && heat >= minHeat;
    }

    public Sprite GetTileSprite ()
    {
        return tileSprites[Random.Range(0, tileSprites.Length)];
    }
}
```

Figure 20

Calculating the Biome

To calculate the biome a new method was created within the map script, this will return a specific biome determine by the three different noise maps this will be done by looping through each of the biomes, where its going to check to see if Match Condition(from biome pre-set) is set to true. If it is true then it will add the biome to a list of temporary biomes that have all met the condition

It will then compare each of them to see which one is the closest to the original value

```
}

// will get the biome that is closest to what the noise maps gives
BiomePreset GetBiome (float height, float moisture, float heat)
{
    BiomePreset biomeToReturn = null;
    List<BiomePreset> tempBiomes = new List<BiomePreset>();
    // will loop through each of the biomes if it  matches the requirements it will be added to tempbiomes
    foreach(BiomePreset biome in biomes)
    {
        if(biome.MatchCondition(height, moisture, heat))
        {
            tempBiomes.Add(biome);
        }
    }
    float curVal = 0.0f;
    // loop through each of the biomes that meet the  requirements
    // find the one closes to the original height, moisture and heat values
    foreach(BiomePreset biome in tempBiomes)
    {
        float diffVal = (height - biome.minHeight) + (moisture - biome.minMoisture) + (heat - biome.minHeat);
        if(biomeToReturn == null)
        {
            biomeToReturn = biome;
            curVal = diffVal;
        }
        else if(diffVal < curVal)
        {
            biomeToReturn = biome;
            curVal = diffVal;
        }
    }
    // if no biome is found - return the first one in the biomes array
    if(biomeToReturn == null)
        return biomes[0];
    return biomeToReturn;
}
No issues found
```

Figure 21

The map is then generated using a tile prefab to instantiate the tiles, this utilizes the GetTileSprite method from the biome pre-set script, which will return a random sprite that has been stored within the tiles sprites array for that biome.

```
void CreateMap ()
{
    // Create the height map
    heightMap = NoiseGenerator.Generate(width, height, scale, offset, heightWaves);
    // Create the moisture map
    moistureMap = NoiseGenerator.Generate(width, height, scale, offset, moistureWaves);
    // Create the heat map
    heatMap = NoiseGenerator.Generate(width, height, scale, offset, heatWaves);
    //creates an array for each of the pixels in the texture
    Color[] pixels = new Color[width * height];
    int i = 0;

    // loop through each pixel
    for(int x = 0; x < width; x++)
    {
        for(int y = 0; y < height; y++)
        {
            // how do we want to display the debug map?
            switch (displayType)
            {
                case MapDisplay.Height:
                    pixels[i] = heightDebugColors.Evaluate(heightMap[x, y]);
                    break;
                case MapDisplay.Moisture:
                    pixels[i] = moistureDebugColors.Evaluate(moistureMap[x, y]);
                    break;
                case MapDisplay.Heat:
                    pixels[i] = heatDebugColors.Evaluate(heatMap[x, y]);
                    break;
                case MapDisplay.Biome:
                {
                    BiomePreset biome = GetBiome(heightMap[x, y], moistureMap[x, y], heatMap[x, y]);
                    pixels[i] = biome.debugColor;

                    if(Application.isPlaying)
                    {
                        GameObject tile = Instantiate(tilePrefab, new Vector3(x, y, 0), Quaternion.identity);
                        tile.GetComponent<SpriteRenderer>().sprite = biome.GetTileSprite();

                    }
                    break;
                }
```

Figure 22

Enemy and player

These are placed after the map has been created using random placement, it is more to show that after PG is done that more object can be added to it later.

Evaluating and testing

Requirements and results

	Requirements	Priority	Requirement Type	Status	Comment
1.	The project must be a fully playable tech demo	Must	Functional	Finished	The tech demo is playable
2.	The project must generate a full traversable 2d map	Must	Functional	Finished	A 2d map is generated at runtime
3.	The 2D map must be made using Procedural Generation	Must	Non-Functional	Finished	The map uses multiple noise maps to generate the terrain

4.	The project must not take too long to load	Must	Non-Functional	Finished	The game is generated consistently quickly
5.	The project must not perform poorly	Must	Non-Functional	Finished	There is no performance issue while running the game
6.	The project must not contain any graphical errors	Must	Non-Functional	Finished	There is no graphical errors
7.	The project must contain a mini-map	Must	Functional	unfinished	Due to time constraints while working onother functionalities I was not able to start the creation of a mini-map
8.	The project must be stable, with minimal crashing	Must	Non-Functional	Finished	There was no crashing during testing
9.	The project must feature a variety of biomes	Must	functional	Finished	There are four biomes with upwasrds of 10 that can be made with minimal effort
10.	The project must implement a noise map in the generation of terrain	Must	functional	Finished	The main focus for the PG will be on the noise map
11	The project must show the user as an avatar in the tech demo	must	Non-Functional	Finished	The user can control a 2d sprite that can traverse the world
12.	the project should generate some form of inaccessible land such as water	should	Non-Functional	unfinished	Was unable to add any collision to the generated tile maps
13.	The project should generate resources such as tree, etc	Should	Functional	unfinished	This could be made however was not attempted due to time constraints
14.	The project should have some form of enemy	Should	Functional	Finished	Minimal enemy feature was implemented

15.	The project should generate cave levels	Should	Functional	unfinished	To add another level of PG
16.	The cave levels should be generated with procedural generation	Should	Non-Functional	unfinished	Main reason of the project
17.	the project should be able to feature multiple sizes of maps	Should	Functional	partially	Ther is no ui that allows this however it is possible to create different map sizes
18.	The project should have a start and end point	Should	Functional	partially	It has a start but no end
19	the project should have some form of npc	Should	Functional	partially	Enemies exits but no non-hostile npc
20.	Resources within the project could be gatherable	could	Functional	unfinished	
21.	The project could have some form of inventory	Could	Non-functional	unfinished	Not attempted
22.	The project could have shops procedural generated	Could	Functional	unfinished	Not attempted
23.	If there is shops the project could have a town	Could	Non-Functional	unfinished	
24.	If there is a town then the project could have npcs that are local to that town	Could	Non-Functional	unfinished	
25.	The mini map could contain markers	Could	Functional	unfinished	

In evaluating the project from what I set out to achieve it is clear the project has fallen short of what was aimed, a series of setback and technical difficulties has led to this incomplete state. Ideally I would have finished all of the must and should requirements, for some of them I had plans on how to create them whereas other I was not able to focus on it

Most of the cave feature are an extension of what is already capable with the map generation. The way it was planned was to have one or two object spawn on the map that would act as doors, these doors would lead to separate maps that would act as the caverns and would move the player within

The idea for proceeding with the towns, shops and npc was to evaluate an area within the map to see if it was free, then instantiate a prefab that was predefined onto the free area, the main

problem when creating this was due to the biomes, trying to find a way to instantiate in one and only one biome while making sure it was note spawned on top of the ocean.

Hardware Testing

The most important requirement that needed to be tested within hardware testing was

- To have a playable tech demo
- Must create the map within a reasonable time

To test the tech demo I tested the map generation with multiple map sizes, the standard size in which the demo was created for was 100x100, in turn I changed the map sizes to fit around this dimension, the other dimensions that were tested was 50x50 and 200x200 and an extreme case of 500x500, all of which took less than a second except the 200x200 which took 3

From these test you can see that the demo is not hardware intensive , however it should be stated that these test are not done extensively.

Effectiveness of project plan

I initially drafted a project plan containing what I assumed was a good foundation and a steady risk analysis, this turned out not to be the case as I have failed to meet most of the requirements that I set as well as not being able to recognise some of the major risks such as well as avoid when capable.

I initially employed an agile approach to the creation of the demo, this was partly due to me being focused upon other course work, however once I hit my first major problem, the agile methodology dissolved and the organisation of the project began to spiral.

Identified Risks

Risk	Encounterd	Severity	Steps taken	Comments
Bugs in the development of tech demo	Yes	High	Create a foundation within code and test regularly	I encountered a few bugs when creating the noise maps

Unable to finish art assets	No	High	The use of grey box assets for demonstration	Did not occur

Unidentified Risks

Risk	Encounterd	Severity	Steps taken	Comments
Domestic Problems	Yes	Low	N/A	N/A
Problems Within other CW	No	High	N/A	Problems with other participants within group work
Hardware Problems	Yes	High	N/A	Had a problem where laptop needed to go into the shop for repair, lost a couple of days work
Inability to finish project to standard proposed	Yes	High		There were sever “should” that were not completed and almost all “could”

Conclusion

In conclusion, the project suffered a series of setbacks that resulted in it not being finished to the standard that was wanted within the requirement specification. However the core of the project , which was to show how Procedural Generated content can rapidly create environments that differ from each other has been shown to some extent with the implementation of the noise map generations and then converting that into a 2d tilemap world with multiple bioms

Link to GitHub Repository

https://github.com/CalumMcAdam99/Cm140-Procedural-Tech-Demo

Success of Procedural Generation

The main aims of the project were to show the procedural generation can quickly and effectively produce environments that are functional as well as being unique, I believe that I was able to partially prove this

The current implementation that I have allows for the generation of a unique environment that is created quickly.

- This allows it to create vast quantities of levels extremely faster than what it would take to manually create them
- It allows itself to be quickly changed and customized through the biomes, where you can make it mostly desert or just ocean and grassland'

However there are drawback from using PGC

- The most obvious one to me would be the complexity of creating and fixing the project

Future Developments

The most prominent development to be made would be to continue to strive to meet the original requirements, this would include.

- Procedurally generated cave system that connects to the map
- Collisions with tiles such as the ocean
- Adding resources , trees
- Inventory system
- Adding npc and prefab housing

Improved combat

Modifying the enemies so that they can patrol and chase the player when the get into close enough proximity, as well as giving them a health system and multiple forms of attack (such as melee and range) this would require that there be more than just one enemy prefab. This could also allow the player character to have a failure state as well as a life system.

Interaction with npc

It was stated within the requirements to have shop, but it was never stated to have npc interaction, this can be something simple such as a simple greeting when the player is within the proximity of an npc, it could also have a simple moral system where the player has the option to attack the npc's and in turn they attack them or just bar them from the shops.

Interactable environments

There is potential to improve on the physics within the game by having the player or npc's collide with and move objects. There is also the possibility of destroying or gathering objects such as trees or stone.

Infinite world

Games such as Minecraft or Elite dangerous have already created world that seem like the go on forever, in truth this would seem like a final improvement upon the game where you have improved everywhere else, and the final step is to make it bigger

REFERENCES

1. Blatz, Michael, and Oliver Korn. "A Very Short History of Dynamic and Procedural Content Generation." *Game Dynamics*. Springer, Cham, 2017. 1-13.
2. Beneath Apple Manor. (1978).Chatsworth: Quality Software.
3. 3-Rogue. (1980). Berkeley: Epyx/BSD.
4. Moria. (1983). Oklahoma: R.Koeneke.v
5. Hades. (2020). San Francisco: Supergiant Games
6. The Elder Scrolls II: Daggerfall. (1996) Rockville: Bethesda Softworks.
7. Giant Bomb (2016). The Elder Scrolls II: Daggerfall. [online]Giant

Bomb.Availableat:https://www.giantbomb.com/the-elder-scrolls-ii-daggerfall/3030-1129/[Accessed 17th November 2017]
8. The Sentinals (1987).Firebird
9. Elite.(1984).Cambridge:Acornsoft
10. Elite Dangerous.(2014).Cambridge:Frontier Developments.
11. Minecraft. (2011). Stockholm:Mojang.
12. Persson,M (2011).Terrain generation, Part 1. [online] The World of Notch.
13. Deep Rock Galactic. 2018. Online: Ghost Ship Games
14. Procedural Level Generation. 2017. Online: Ghost ship games
15. Spore.2008. California: Maxis
16. Creating Creatures. 200,online:MIT technology Review
17. Watson, Benjamin, et al. "Procedural urban modeling in practice." *IEEE Computer Graphics and Applications* 28.3 (2008): 18-26.
18. Scirea, Marco, et al. "SMUG: Scientific Music Generator." *ICCC*. 2015.
19. A look into procedural generation. 2017.online:medium
20. Borderlands 3. 2019. California: 2k Games
21. Apex Legends. 2019. California: Respawn Entertainment 22. Unity (2017). Mathf.PerlinNoise [Online] Unity Documentation.
23. Binary Space Partitions. 2020. Online:GeeksforGeeks
24. Cui, Xiao, and Hao Shi. "A*-based pathfinding in modern computer games." *International Journal of Computer Science and Network Security* 11.1 (2011): 125130.
25. Morin, P. OpenData Structures (in Java). [online] OpenDataStructures.org.
26. Djikstra's Shortest Path Algorithm. 2020.online:brilliant.org

APPENDIX

1.Beneath Apple Manor - https://dosgames.com/game/beneath-apple-manor/

2.Elder Scrolls: Daggerfall - https://en.wikipedia.org/wiki/The_Elder_Scrolls_II:_Daggerfall

3. Moria - https://www.retrogamer.net/retro_games80/moria/

4.Elite (1984) - https: //en.wikipedia.org/wiki/Elite_(video_game)

5. Minecraft - https://www.digitaltrends.com/gaming/how-to-get-minecraft-for-free/

6. deep rock galactic - http://starnews.ca/article/5549/review-deep-rock-galactic/

7.Spore - https://store.steampowered.com/app/17390/SPORE/

8. crowd simulation - https://www.researchgate.net/figure/Procedural-city-shown-from-groundlevel-showing-the-crowd-that-is-alongside-the_fig41_327142029

9. procedural generation in films - https://medium.com/@homicidalnacho/a-look-into-proceduralgeneration-dfa62fe7536d

10. No mans sky - https://www.gamesradar.com/uk/no-mans-sky-gets-a-free-ps5xbox-series-xupgrade-on-day-one/

11. Noise Maps - https://www.gamesradar.com/uk/no-mans-sky-gets-a-free-ps5xbox-series-xupgrade-on-day-one/

12. djiktras algorithm being used - http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

13. A* algorithm being used - http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

14: ftl - https://store.steampowered.com/app/212680/FTL_Faster_Than_Light/ 15:

KKrieger - https://en.wikipedia.org/wiki/.kkrieger

15: Height Noise Map

16: Moisture Noise Map

17: Heat Noise Map

18: High Frequency Noise Map

19: High Amplitude Noise Map

20: Biome Pre-set Code

21: Get Biome Code

22: Create Map Code