Procedurally Generated Rogue-like Game

spheRe

By [redacted] [redacted] [redacted]

BSc Computer Games Technology

Supervisor – Huw Lloyd

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work.

Signed: 

Abstract

This is a Computer Games Technology project looking at procedural generation in regards to the Roguelike genre. The Roguelike genre is an almost 3 decade old genre hearkening back to the game that inspired the genre, Rogue. Rogue used many features that are a mainstay of the genre now, such as procedural generation, permadeath and a turn-based game loop. This project attempts to use these ideas, and to look at relevant work on similar games, to create a vertical slice of a procedurally generated roguelike game. This project had a strong randomly generated world, as evidenced by the fact that 100% of users stated that the game felt fairly to very random.

Table of Contents

Chapter 1 – Introduction

1.1 - Project Background

Since the original Rogue back in 1980, many games have employed the strategies that this game used for its core development. Many "clones" of Rogue have been developed over the years, with varying degrees of similarity. Games started to pick and choose elements of Rogue as they went on, taking thing such as perma-death, random generation and turn-based combat.

Games that only follow this premise loosely are referred to as "Rogue-lites", while games that take a large number of these concepts are considered "Rogue-likes", named for the popularity of Rogue.

To create a Rogue-like, the game generally has to hit the same elements as the original Rogue. Features such as the ASCII art however are not necessary to be considered a Rogue-like.

The game generally requires:

- Random tile-based levels.
- Perma-death.
- Turn-based combat.
- Emergent Gameplay.
- Fantasy Themes.
- Inventory System.

The original Rogue started a huge trend of procedural generation in video games. Procedural generation is the act of creating random levels using a variety of different methods. It uses an algorithm to create levels as opposed to having a human design them manually.

There are many ways of generating random tile-based levels, which have been used in various different Rogue-like games. Such examples include:

- The search based approach
- Binary Space Partitioning
- Agent based growing
- Cellular Automata
- Fractal and noise based generation.

Rogue itself used binary space partitioning to create its random dungeons, however many Rogue-likes have not followed this and have used these various other ways of procedural generation.

Each way of procedural generation has its own pros and cons, which have to be considered when designing the game, as each method creates very different levels which all need to be approached with a different design in mind.

With the dawn of easy self-publishing due to the rise of high-speed internet, many independent developers with very small or even one man teams can now create and sell their games, while still reaching a wide audience. With such a small team often comes the problem of creating enough content to make their games worthwhile and keep the player engaged.

Procedural generation has been very helpful in this regard with many games choosing to procedurally generate their game content, and focus solely their design on mechanics, enemies and challenges.

1.2 - Related Work

There have been many games that have used procedural generation to generate game content, with the majority being independently developed like Minecraft and various roguelikes such

as The Binding of Isaac or Nuclear Throne, and some large publishers such as Nintendo with the Pokémon Mystery Dungeon series.

Cellular Automata has been used in many games and is a popular method of content generation. A piece of work evaluating a method of Cellular Automata has been created by the IT University of Copenhagen [Johnson, Yannakakis, Togelius. (2010).] The Delft University of Technology also created a piece surveying the various uses for Procedural Content Generation [Hendrikx, et al. (2013).]

1.3 - Aim

A1- To gain a better understanding of the various ways to procedurally generate game content.

A2- To practise skills in coding game logic.

A3- Create a strong algorithm for generation of procedural game content that creates an enjoyable experience.

A4- To create a game that is structured so that new levels and enemies can be added with little difficulty beyond their design.

1.4 - Objectives

O1- Study the various methods of procedural generation.

O2- Decide upon and practise with a given method.

O3- Develop a suitable Algorithm/Pseudocode for the generation of content using this method.

O4- Select a Game Engine to create game content using this method.

O5- Design content that could fit around this engine and method.

O6- Gain feedback on the randomness of the game using play testing.

O7- Test the usability of the game via play testing.

O8- Having created a working game, review findings with feedback and evaluate the development process.

1.5 - Problems

There are many potential problems in the development cycle of a videogame. The code could be difficult to implement within the engine, too large a scope at the design stage of the game could run into problems with time management and the ability to deliver the original concept, leading to cuts or maybe rush unbalanced game elements.

For a Rogue-like specifically, if the algorithms for procedural generation are not polished to the right amount, it could create levels that are not very fun to play, or even worse simply unbeatable. Parts of the level could be inaccessible if the code is not written properly, and this could lead to exits being unreachable, or certain elements being out of reach.

With the idea of random chance, and the turn based combat system, if many enemies spawn at once, certain levels might become completely unbeatable. With a roguelike this is expected to happen every now and again, but too often and it ruins the games replayability.

Due to the inherent random nature of the game, content will not only need to be designed very specifically to avoid balance issues, but it will also need to be implemented in very specific ways. The game will need to adhere to a steadily rising difficulty setting, and the balancing of the game will need to adhere to this while remaining interesting and fun.

1.6 - Solutions

In this project the Rogue-like game will be created with the Cellular Automata method of procedural generation in mind. This solves the industry problem of independent developers not having enough time and resources to develop all of the game content by hand, and thus helps them create large amounts of content for their game.

1.7 -Required Resources

The game will require a game engine, a scripting language, and some photo editing software for the creation of the in-game assets. An Audio-editing tool will also be necessary for the creation of audio cues and sounds.

The review and evaluation of the game will require willing participants to test the game.

Chapter 2 - Literature Survey

2.1 - Background

Procedural generation is a technique that has been used in videogames for almost 30 years. Since Rogue in 1980, many games have chosen to use the route of procedural generation, with regards to the levels or even content itself. Rogue was created by Michael Toy and Glenn Wichman, with Ken Arnold contributing later on. Rogue was released for UNIX, but was later ported to non-Unix systems.

Due to the popularity of Rogue at the time, there were many copycat games created, leading to the coining of the term “Roguelike”. Roguelike simply means a game created with the same principles as Rogue, such as Procedural generation, permadeath (when the character dies the game restarts), a fantasy theme and many more.

Rogue created its own dungeon generation algorithm, which created levels which were just rooms connected by corridors. Upon creating of the randomly placed rooms, the algorithm also randomly placed loot for the player to collect, and monsters for the player to fight.

While Rogue used the method of placing rooms and then essentially joining up the dots, there are many different ways of procedural generation, although some are better suited than others. One method is known as Binary Space Partitioning which uses a binary tree structure to generate rooms and then connects them to their parent nodes. [Shaker, et al. 2016]

Another way to randomly generate dungeons is the use of an “agent”. The agent digs corridors, with a small chance of generating a room. It moves through the grid turning every now and again, generating rooms as it goes till it has created a full dungeon. [Shaker, et al. 2016]

Cellular Automata is where the code iterates through a grid, determining if the tiles are on or off depending on a given percentage. The grid is then iterated through again, switching the status of the tiles depending on how many tiles of a certain type surround it. Tiles with more walls than floor surrounding it become walls, and tiles with more floor surrounding it than walls become floor. This is then done a few more times to create smoothing looking formations, which are then joined up. [Shaker, et al. 2016] The most famous instance of this is John Conway's Game of Life.

Procedural Generation itself has been covered by many academic studies, such as the generation of virtual cities [Greuter, et al. 2003] or even the procedural generation of roads between cities [Galin, et al. 2010].

It is quite apparent the similarities between these two studies and Rogues map generation Algorithm. Generating a virtual city and then connecting with generated roads is no different than generating virtual rooms and connecting them with generated corridors.

2.2 - Terminology

This report uses various computer science terminology, so this section will cover anything not apparent to the layman.

An Algorithm is essentially an idea for a set of rules that the code will follow. In the example of procedural generation it's the rules that define the levels generation.

A Binary Search Tree is a method of storing date by attaching two new pieces of data to the starting data, and then two new pieces of data to those pieces. This continues until there is no data left.

The Game of Life is an algorithm created by John Conway, which is contrary to the name more of a simulation than a game. It uses a very strict set of rules and cellular automata to create patterns as it is ran.

2.3 - Algorithms Justification

The project itself is using Cellular Automata. Cellular Automata generally creates a lot more visually appealing levels than the other methods. The long winding cave system that the algorithm generations is a lot more interesting and varied than the standard box connected to another box with a line. It creates many small areas of the map where special items could be hidden away and makes exploring the level a lot more fun and engaging.

It's also very easy to iterate with, so once it is done generating the level, the code can iterate through the level again, adding enemies, loot, traps and so on. This will create a much more compelling and random feeling game for the player to play through and should be different enough each generation to keep interest.

The pathfinding for this game is going to be done primarily through the use of the A* Algorithm. The A* Algorithm is one of the most popular algorithms for pathfinding in games, primarily due to is usage of Heuristics. These Heuristics allow the user to tailor the algorithm to find paths that prioritize different movement schemes. [Xiao, Hao. 2011]

2.4 - Tools

The code for the game is written in Visual Studio, using C# scripts, which are compiled together using the Unity Engine. As this is mainly a code based project, most assets will be acquired online from royalty free archives. Any visual assets created for the game itself will be done in GIMP, and any audio editing done in Audacity.

2.5 - Related Work

There have been many instances of games using the roguelike formula, going back to Hack and Moria in ’92 and ’93 respectively. There are many more recent examples of Roguelike games, and even games that only very loosely take from Rogue, these being known as Rogue-lites.

One particular example of a Rogue-lite is a game called ‘Rogue Legacy’. Rogue Legacy employs some of the features of Rogue. It has Permadeath, Random Generation and Random Skills. However it employs 2D graphics and a side on action based combat system. Also even with the Permadeath system, the player gains rewards that progress through different lives, that can be used to power up their character overall. It is almost required that the player collects these rewards to defeat the game, although player have beaten it on one life by memorizing attack patterns. [Cellar Door Games. 2013]

Rogue Legacy was very well received, receiving an 85 aggregate score on Metacritic, and a 7.9 user score. Reviews of the game praise it based on how well it implements its Roguelike features. [Shea, C. 2013]

Another very popular Rogue-lite is known as Spelunky, created by Derek Yu. Originally created as an 8bit game online, Spelunky was “Remastered” for the Xbox Live arcade and Steam. Like Rogue Legacy, Spelunky employs 2D graphics with a side on puzzle platforming action based system. There is a progression system outside of the main gameplay loop, with the player being able to create shortcuts to levels further into the game, but the game actively discourages this by making the player unable to reach the true end game without starting from the beginning each time, thus making the Permadeath system in this game a little closer to a true roguelike. The levels are randomly generated, and the player can pick up items to give them new abilities, such as the climbing gloves to stick to surfaces or the jet pack to fly.

The player has no inherent skills outside of these, other than basic movement, an attack, and the ability to jump on certain enemies similar to Mario. [Yu, D. 2009]

Spelunky had great critical success, hitting a 90 on metacritic, however only received a 7.2 on user score. Spelunky gained most of its praise based on the various gameplay systems it used, as opposed to the Roguelike elements themselves. The way the gameplay systems interact, creating very memorable moments of emergent gameplay, is what made the game so appealing. Every decision the player made had the potential to change the course of a run entirely. [Bramwell, T. 2013]

A far more classical example of a Roguelike is the game Dungeons of Dredmor. Dungeons of Dredmor is a top down, turn based Roguelike, operating under very similar rules to the original rogue, regardless of the 2d graphics as opposed to Rogue's classic text based graphics. Every run of Dungeons of Dredmor is randomly generated, and each run a new character is generated. There is the option for this character to be random, although the player does have some leeway over the characters skills and abilities if they so desire. The game has pickups and a level up system per run, and the likelihood of the player reaching the end game is based purely on their knowledge and skill, with a small amount of random chance. However there is no system in place to allow the player to have better in-game skills on a run to run basis. [Gaslight Games. 2011]

Dungeons of Dredmor was fairly well received, garnering a 79 metacritic score, and a 7.8 user score. Dungeons was praised highly on its hardcore difficulty, and character creation system, with very honourable mentions of the writing, be it item and weapon text or even just the lore hidden within the game. [Meunier, N. 2011]

A very refreshing take on the roguelike genre was FTL. FTL is not a standard roguelike in the sense that you control more than one character and it deviates substantially from the fantasy

theme that is common across almost every Roguelike. The game itself doesn't really contain dungeons, but its galaxy generation is something in a similar vein, creating a galaxy map that randomly contains different types of system, such as a friendly system, an enemy system, or a neutral nebula. These are essentially the dungeons of FTL, with each node in a system offering a random chance of encounter. Each run of the game starts with a pre-set ship you can choose from, but the player can upgrade this as they move through each run. The random nature of this game and the sheer amount of permutations of ship upgrades, new characters and character levels make every run of this game a very different beast. [Subset Games. 2011]

FTL's metacritic was very well received, getting the same score from Critics and Consumers alike. 84 from critics and 8.4 from users. FTL was praised for its incredibly brutal difficulty, with every encounter being a possible failstate, making every action the player takes incredibly important. The randomness of the game, and the way every run is different has been lauded as one of the most important features of this game, and this counts for almost every roguelike. [Dilks. 2012]

One of the bestselling and well known indie Roguelikes is the Binding of Isaac. Since its release in 2011, the original version of the game has sold roughly 1.6 million copies. It has been remade as well, by another company for the major consoles and pc as well. The Binding of Isaac randomly generates a level of interconnecting randomly chosen premade rooms. The main draw of the Binding of Isaac is the randomly chosen items and the interactions between them. Each floor has the chance for at least one new item, but generally a few more. Each item can interact with the others, creating strange new ways of attacking that the player might not even have considered. The game isn't turn based, using a top down almost bullet hell system, but there is 'permadeath', with progression outside of each run, with character and item unlocks. [McMillen. 2011]

The Binding of Isaac: Rebirth (the remake) received critical acclaim, with an 88 critic score, and a 7.7 user score on metacritic. The game is praised for its very unique design, a twisted macabre representation of the Christian religion. The Item connectivity and control scheme were also highly praised. [Parkin. 2014]

A lesser known, but incredibly loyal to the original concept of a roguelike is a game known as powder. Powder is a top down, turn based Roguelike, with very simple graphics. While it is not the ASCII that Rogue is known for, it is still very basic and relays the same information. The levels are all randomly generated with permadeath, and even the potion effects change between runs, so a red potion might heal on one run, and will poison on the next. The game does have a finite end, with the player descending through multiple dungeons to find and defeat Baezl'bub. [Lait. 2009]

Procedural Generation does not need to be contained to just level generation and content placement. It's possible to generated Textures and Sounds as well, to create content in a quick, efficient manner. Textures can be generated using noise generation and even genetic algorithms too. Urban environments are very commonly created using procedural generation, with networks of roads connecting buildings that themselves are generated procedurally.

Even puzzle games can be created with procedural generation. Puzzles are created randomly according to a set of rules, and then the algorithm runs through every possible permutation of these puzzles, discounting them if they are impossible or can be broken through player input. [Hendrikx, et al. 2013]

Dungeons have been procedurally generated in games for over thirty years, originally proving the benefit of procedural generation in games. Dungeon generation was typically just a series of rooms connected by hallways but various different algorithms have covered different types of geometry and topology. The control over the generation of dungeons is a huge element in

why procedural generation is so well used. The Designer of the generation algorithm can tweak it to create many different looking dungeons and areas.

Procedural content has moved far beyond simple connections of rooms and has started to use various high-end techniques like cellular automata, generative grammars and various genetic algorithms, all of which are used to create much more interesting looking areas for players to interact with. [van der Linden, et al. 2013]

2.6 - Evaluation

The Project is going to be made in Unity using C# scripting. After an exhaustive review of the various types of procedural generation, the level generation and content placement is going to be done with Cellular Automata. The game is going to include the main features of a Roguelike, including permadeath, a turn based movement and battle system and basic 2d graphics. For the purpose of the Project, the game is going to be a vertical slice, being a maximum of a ten to fifteen minute run through a randomly generated dungeon.

There will be various usable items dotted around the level, such as potions, weapons and armour. Enemies will also be scattered throughout the map, and will be combated in a tactical fashion.

The game will lend many of the critically acclaimed features of other Roguelikes, with stackable items, a fairly hard difficulty meaning the player has to think out their actions and a hopefully humorous tone within the lore of the game.

Procedural Generation is a huge boon to Indie Development, allowing rapid development of varied content, within a small time frame. The random nature of procedural generation also creates a sort of replayability, giving the player more time with the game.

The diversity of content within a game that could be generated is massive. Virtually every aspect of the game from sound design to level design and even narrative can be done procedurally, meaning that a lot of development can be done quickly and efficiently.

Chapter 3 - Design Statement

3.1 - Level Design

The game is going to be created with an algorithm that creates large sprawling cave systems. The best way to achieve this after researching various Procedural Generation methods, would be the usage of cellular automata, to create these. The smoothing technique employed within the cellular automata creates the nice curved edges of the areas, something that is not generally possible with other methods of procedural generation, at least in a way that applies to roguelikes.

While the levels layout will be generated by Cellular Automata, the graphics of the level is going to be tile based low pixel graphics. As the Cellular Automata operates within a grid system, simply applying a graphic to a tile is the easiest method of adding graphics to the game.

This isn't just a complexity issue however, In keeping with the nature of Roguelikes, the graphics are intentionally simple in an attempt to replicate and stick to Rogue's ideologies.

3.2 - Character Design and Abilities

The Character can move in the 4 major directions, Up, Down, Left and Right.

The character can:

- Pickup Objects
- Equip Weapons
- Equip Armour
- Drink Potions
- Cast Spells from scrolls

- Attack with its weapon
- Throw certain objects

The Character is going to have a certain amount of hit points, but its ability to cast spells will be decided purely by the amount of scrolls it has.

3.3 - Enemy Design and Abilities

There are going to be two enemies over the course of this vertical slice of the game. In the finished game, enemies will vary level by level, and will be designed according to the theme.

The first enemy type is just a basic enemy. As the enemy is spawned into the game, it will arbitrarily be given a level between 1 and 3. This scales up the enemies health and damage. These enemies follow the player when it is within a certain range of the character, and their line of sight is blocked only by walls. Enemies can attack the enemy when adjacent to the player. When the enemy's line of sight is blocked, and the player is not visible, they simply stand waiting for a visual of the player.

The second enemy type is a boss that is very similar to the regular enemy. It however has twice the damage and health of the strongest enemy in the level, and is quite a challenge because of this. The boss hides by the exit, ready to jump out at the player when they seek freedom.

3.4 - Item Design

<u>Potions</u>

There are various potions in the game:

- A health potion which heals a random amount within a certain range.
- A strength potion which increases the player's physical damage.

Weapons

- Sword
- Club
- Axe
- Bow
- Rock (throwable)

These weapons increase the player's damage. The Sword is considered upper tier, having the highest damage. The Axe lower, and the club lowest.

The bow is slightly worse, but it has the added bonus of being a ranged weapon, allowing the player to deal damage from afar. The rock has the same ranged ability, but is used up on attack. Its damage is very low, but can swing combat in the players favour.

Armour

- Head
- Body
- Legs

The player can find pieces of armour for each of these body parts. These armour fall into 2 categories in this stage. Leather or chain. Leather has lower defence.

Scrolls

The player can pick up scrolls containing magic spells. These are destroyed upon use, but can perform special actions that help the player. These scrolls are:

- Scroll of fireball. This cast a ball of fire, doing 4 damage to a block of 9 squares.
- Scroll of push. This pushes the enemy back 3 tiles, away from the player.

- Scroll of teleport. This randomly teleports the player to a point in the level.

3.5 - Win State

To beat the vertical slice the player must simply find the exit, defeat a boss enemy and go “down” to the next level. In the main game the player must progress through several levels, and defeat a big boss.

3.6 - UI

The UI for the game is designed to be inherently simple. It designed to show the bare minimum that the player needs to complete the game. The UI will show the player’s health and current damage they deal. It will show all of the armour and weapons they have equipped, upon their discovery in the world.

The Inventory will only appear when the player has picked up an object in the world. As objects are collected and used, items in the inventory will appear and disappear.

Enemies will have their name and number over their representations in the world, so that the player can know which enemy they are attacking if there are multiple in close combat. They will also have their health points displayed alongside this. Enemy damage will not be displayed, although discerning players will realise that enemies with higher starting health will have higher damage.

Chapter 4 - Implementation

4.1 - Cellular Automata Level Creation

The level of the game has been created using Cellular Automata to create a large branching cave for the player to explore. The Cellular Automata works by first generating the map itself. It does that by creating an array of 2 integers, which work as the width and height of the map. This array stores a single integer value for the coordinates. This value indicates if the tile is on, or off. A tile that is off is a floor tile, and is represented by the integer zero. A tile that is on is a wall tile.

These tiles are randomly assigned to either be on and off, creating a grid of wall and floor tiles with a completely random orientation. These are then "smoothed" to become a natural looking cave system.

```
void randomFillMap()
{
    seed = (UnityEngine.Random.Range(0f, 100f));
    System.Random pseudoRandom = new System.Random(seed.GetHashCode());

    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (x == 0 || x == width - 1 || y == 0 || y == height - 1)
            {
                map[x, y] = 1;
            }
            else
            {
                map[x, y] = (pseudoRandom.Next(0, 100) < isWallPercent) ? 1 : 0;
            }
        }
    }
}
```

Figure 1 - Random Tile Allocation

The smoothing works by getting the state of each tiles neighbour. If the tile has more than four neighbours that are walls, the tile becomes a wall. If the tile has less than four neighbours that are walls, that tile will become a floor tile. Any tile that has exactly four neighbours that are walls will remain the same state it was in before the smoothing.

This smoothing is repeated multiple times until the required smooth cavern feel has been reached. This requires the level to be smoothed six times. The code sometimes creates multiple cavern systems, and if this is the case, all but the largest cavern system is filled with walls, to create one seamless dungeon.

```
void smoothMap()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            int neighbourWallTiles = getSurroundingWallCount(x, y);

            if (neighbourWallTiles > 4)
                map[x, y] = 1;
            else if (neighbourWallTiles < 4)
                map[x, y] = 0;

        }
    }
}
```

Figure 2- Tile Smoothing

After this simple cubes are instantiated, white for the floor (tiles at state 0) and black for the walls (tiles at state 1).

4.2 - Asset Placement

After the level has been generated, the player is then spawned. A prefab of the player with all necessary scripts attached is randomly placed in the bottom left quadrant of the level.

```
void spawnPC()
{
    seed = (UnityEngine.Random.Range(0f, 64f));
    System.Random pseudoRandom = new System.Random(seed.GetHashCode());

    int x = pseudoRandom.Next(0, 64);
    int y = pseudoRandom.Next(0, 64);

    if (map[x,y] != 1)
    {
        players = GameObject.FindGameObjectsWithTag("Player");
        foreach (GameObject player in players)
        {
            player.transform.localPosition = new Vector3(x, 1, y);
        }
    }
    else
    {
        spawnPC();
    }
}
```

Figure 3- Player Spawning

Following this, all the enemies are spawned as well. These enemies are randomly placed around the map, and are assigned a level between 1 and 3 as they are spawned. This level acts as a multiplier for the health and damage. Generally between 25 and 50 enemies spawn, although sometimes there are greater or fewer numbers.

After all the enemies have been placed, items are added into the world. As the script adds items, there is a 1 in 3 chance for the item to be either a piece of armour, a weapon, or a consumable. When the script has decided what kind of item it will be, it is then further split into what item of this type it could be. There is no weighting to this system, so multiple items are often spawned.

```
void spawnItems()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            int seed = (UnityEngine.Random.Range(0, 1000));
            if (map[x, y] != 1)
            {
                if (seed < 2)
                {
                    int itemtype = (UnityEngine.Random.Range(0, 3));
                    if (itemtype == 2)
                    {
                        int armourtype = (UnityEngine.Random.Range(0, 6));
                        {
                            if (armourtype == 0)
                            {
                                GameObject newArmour = (GameObject)Instantiate(leatherHelm, new Vector3(x, 1, y), Quaternion.identity);
                                newArmour.name = leatherHelm.name;
                            }
                            if (armourtype == 1)
                            {
                                GameObject newArmour = (GameObject)Instantiate(leatherChest, new Vector3(x, 1, y), Quaternion.identity);
                                newArmour.name = leatherChest.name;
                            }
                            if (armourtype == 2)
                            {
```

Figure 4- Item Spawn Example

Finally after everything else in the level has been created the exit is instantiated. The exit is a simple red cube, allowing the player to beat the game by interacting with it. However the exit is guarded by a level 6 boss monster that is also spawned at the same time, providing the player with one last challenge.

4.3 - Interactions

The combat, like the rest of the game itself is turn based. The player gets to take the first move, and can attack by a few different methods. The player can attack with their melee weapon, which will be either a fist, a club, an axe or a sword. They can attack any enemy that is directly adjacent to them, for a set amount of damage per turn. They can also attack with a bow, if they have collected it, by clicking on the bow, then on the specific enemy they wish to attack, if it is in range.

```
else if (Input.GetMouseButtonDown(0) && pc.GetComponent<playerStats>().usingbow)
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;
    LayerMask layermasksphere = -1;
    if (Physics.Raycast(ray, out hit, 100f, layermasksphere, QueryTriggerInteraction.Collide))
    {
        UnityEngine.Debug.Log("You selected the " + hit.transform.name);
        if (Vector3.Distance(hit.transform.position, pc.transform.position) < 6)
        {
            foreach (GameObject enemy in enemies)
            {
                if (enemy.name == hit.transform.name)
                {
                    enemytodamagebow = enemy;
                    bowdamage = true;
                }
            }
        }
    }
}
```

Figure 5- Bow Raycast

There is also a linecast drawn between the player and the enemy. If the line passes through any wall, the enemy takes damage. If not this counts as a turn, and no damage is done.

```
if(bowdamage)
{
    Debug.DrawLine(enemytodamagebow.transform.position, pc.transform.position, Color.red, 10f);
    bool cansee = false;
    if (Physics.Linecast(enemytodamagebow.transform.position, pc.transform.position))
    {
        cansee = false;
    }
    else
    {
        cansee = true;
    }
    if (cansee)
    {
        enemytodamagebow.GetComponent<enemyStats>().health = enemytodamagebow.GetComponent<enemyStats>().health - 2;
        if (enemytodamagebow.GetComponent<enemyStats>().health <= 0)
        {
            Destroy(enemytodamagebow);
        }
    }
    bowdamage = false;
    enemytodamagebow = null;
    moved = true;
}
```

Figure 6- Bow Linecast and Damage

The player can also attack by using one of the various items that are spawned throughout the map. The Fireball scroll creates a collider on the tile that the player clicks, provided it is in range. This collider makes every enemy within it take damage, and is roughly a 9 tile square, with the centre tile being where the player clicked.

The rock operates exactly like the bow, but it is only able to be used once. While it has very low damage, it can still swing combat by lowering an enemy's health to an amount it can be killed by one melee blow, if the player has the right item. This also has a linecast checking there are no walls in the way.

The push scroll calculates which direction the enemy is from the player. It then pushes the enemy 3 squares back in this direction. If the enemy is pushed inside of a wall, it is then destroyed. This makes this item very useful for killing high level enemies.

The health potion simply increases the player's health points by 3, although it checks that the health points are not now higher than the max health points, and if they are, sets them to the max health allowed.

The strength potion is just a Boolean called "isStrong" that multiplies the player's damage by 1.5 for 3 turns. This is achieved by having a running counter that starts at 2 when the level is first loaded and keeps increasing by 1 per turn. When the strength potion is drunk, the counter is set to 0, and if the counter is less than 2 the Boolean becomes true. After the counter then becomes greater than true, the Boolean is reset back to false.

The Teleport Scroll randomly chooses a point in the map to teleport to, by randomly generating an x and z coordinate between 0 and 128. If this point is a wall, the player is not teleported, and the scroll destroyed. This counts as taking a turn. If the point is not a wall and the players position is updated to this.

4.4 - Pickups/Inventory

The characters inventory is split into two sections, equipable items and consumable items. The equipable items are simple Booleans with the general naming scheme of "hasArmourPart" or "hasWeaponPart". Items are interacted with by having the player's position and their position be the same, and then having the player press the 'X' key. If the player does not have this piece of armour, or does not have a higher tier piece of armour, the corresponding Boolean is ticked, and this item is destroyed. If the player has the armour or a higher tier piece of armour, the item will not be picked up.

Once the armour has been picked up it increases the player's health by a specific amount. When a weapon is picked up, this increases the player's damage by a specific amount. Upon pickup, the relevant UI texture is then drawn, notifying the player when it is in their inventory. When the bow is picked up, the bow Boolean is ticked, and the bow UI element is added. This creates a button under the bow UI element, which is for using the bow during combat.

The consumable item inventory works a little differently. As an item is picked up, it is destroyed in the world, and an integer corresponding to that item is added to a list in the player stats script. This is then converted into an integer array for drawing the UI elements to the screen.

Each element in the inventory array has a slightly different location to draw on the HUD. When all 6 slots are filled, each is directly next to the element before it, along the middle bottom of the screen. A series of IF statements based on the int value in the array decide which UI element to draw, corresponding to the item stored.

As the UI elements are drawn on screen, each creates a box behind it, allowing the player to use the item, similar to how the bow functions. As the item is used, that items integer is

removed from the inventory array, and the inventory list is updated, sliding all the elements down 1. The max size for the inventory is 6 integers, forcing the player to either waste items they do not want, or forgo picking up more items.

4.5 - Pathfinding

The original design statement for this game included the use of the A* pathfinding algorithm to calculate the path the enemy would take between the enemy and player. If the player was within a certain range of the enemy, and was in the enemy's line of sight, the enemy would move towards the player after each turn the player made, until they were close enough to attack the player.

For the A* algorithm to work, a script was created to turn the game world into a grid of nodes. Starting at the bottom left of the world, where the cellular automata script created the world made of cubes. It makes a grid out of all the walkable tiles, with nodes exactly the same size as each floor cube. The walkable tiles were calculated by getting the corresponding map coordinates from the cellular automata code, and checking if they were walls. If they were not walls they were considered walkable and added to the grid.

This code created an invisible grid of walkable tiles over the floor of the level, for the enemies to use in their pathfinding code. This grid was constructed of Nodes, which were set up with some cost weightings, and coordinates. The cost weightings were mostly irrelevant, as there were no high cost tiles, and the enemies can only move in 4 directions. However, this allowed weightings to be applied to specific tiles, if they were to be added later in development.

The A* algorithm took the starting location of both the enemy and the player, and set up an open set and a closed set for sorting nodes after they had been examined by the pathfinding script. The script checks the current node to see if it the target node, and if it is, it considers

the path found. If the path is not found it then examines all of the nodes neighbours, and works out all the movement costs to these neighbours. As all the costs are the same, it considers all of the neighbour tiles as long as they are walkable and they have not been considered before. It moves along all neighbours it can, basically spreading out in a circle from the enemies original point until it comes into contact with the player.

Once the player, or target node has been found this specific path that has discovered it is retraced back to the enemy, and each node it has passed through is added to a specific array. The enemy would then be moved through the points of this array towards the player.

However, as the game is turn-based, every time the player moves the path needs to be updated so that the enemy is moving towards the player's new position.

Upon implementation of the A* path-finding into the game, there was a bug that stopped this method from being able to be used. Each time the path was remade, the first point of the path would always be 1 tile directly down from the enemy, even if that tile was not part of the walkable grid. This would have been more or less fine in a none turn based setting, as the rest of the path would be fine, and the enemy could continue towards the player after the initial step, however as it was updated as the player moved, the enemy would continuously move down, into walls where they would then become stuck.

The reason for this problem could not be found, and therefore the A* algorithm could not be properly integrated into the game. A workaround was found, by applying constraints to Unity's own move towards function.

```
public Vector3[] FindPath(GameObject enemy, GameObject target)
{
    Vector3 startPos = enemy.transform.localPosition; Vector3 targetPos = target.transform.localPosition;
    Vector3[] waypoints = new Vector3[0];
    bool pathSuccess = false;

    Node startNode = grid.NodeFromWorldPoint(startPos);
    Node targetNode = grid.NodeFromWorldPoint(targetPos);

    Heap<Node> openSet = new Heap<Node>(grid.MaxSize);
    HashSet<Node> closedSet = new HashSet<Node>();
    openSet.Add(startNode);

    while (openSet.Count > 0)
    {
        Node currentNode = openSet.RemoveFirst();
        closedSet.Add(currentNode);

        if (currentNode == targetNode)
        {
            pathSuccess = true;
            break;
        }

        foreach (Node neighbour in grid.GetNeighbours(currentNode))
        {
            if (!neighbour.walkable || closedSet.Contains(neighbour))
            {
                continue;
            }
            int newMovementCostToNeighbour = currentNode.gCost + GetDistance(currentNode, neighbour);
            if (newMovementCostToNeighbour < neighbour.gCost || !openSet.Contains(neighbour))
            {
                neighbour.gCost = newMovementCostToNeighbour;
                neighbour.hCost = GetDistance(neighbour, targetNode);
                neighbour.parent = currentNode;

                if (!openSet.Contains(neighbour))
                {
                    openSet.Add(neighbour);
                }
                else
                {
                    openSet.UpdateItem(neighbour);
                }
            }
        }
    }
    if (pathSuccess)
    {
        waypoints = RetracePath(startNode, targetNode);
        return waypoints;
    }
    return null;
}
```

Figure 7- Bugged A Implementation*

4.6 - A* workaround

If the enemy could see the player, as proved by doing a linecast between the player's position and the enemy's position and checking if the line hit any walls, and the player was within a certain range of the enemy, the enemy pathfinding would begin. The greater difference between the x and the z coordinate of the enemy and the player was calculated, and the enemy was moved towards the player 1 step in that direction.

This constrained all the movement to a pseudo-grid, but having the enemies only move in steps the size of each floor tile. If the difference between x and z was the same, the enemies moved horizontally one step. Before moving, the code would check whether that position was currently occupied by any other enemy in the game. If it was, the enemy would not be able to move to that position.

Once the enemy was close enough to the player, this would activate the damage calculation method, and attack the player.

```
Debug.DrawLine(enemy.transform.position, pc.transform.position, Color.red, 10f);
bool cansee = false;
if (Physics.Linecast(enemy.transform.position, pc.transform.position))
{
    cansee = false;
}
else
{
    cansee = true;
}

if (cansee == true)
{
    float checkx = enemy.transform.position.x;
    float checkz = enemy.transform.position.z;

    enemy.transform.position = Vector3.MoveTowards(enemy.transform.position, pc.transform.position, 1);

    float checknewx = enemy.transform.position.x;
    float checknewz = enemy.transform.position.z;

    int x = Mathf.RoundToInt(enemy.transform.position.x);
    int y = Mathf.RoundToInt(enemy.transform.position.y);
    int z = Mathf.RoundToInt(enemy.transform.position.z);
    Vector3 roundedVector = new Vector3(checkx, y, checkz);

    float changeinx = checknewx - checkx;
    float changeinz = checknewz - checkz;

    changeinx = Mathf.Abs(changeinx);
    changeinz = Mathf.Abs(changeinz);

    if (changeinx > changeinz)
    {
        roundedVector = new Vector3(x, y, checkz);
        if (map[x, (int)checkz] == 1)
        {
            roundedVector = new Vector3(checkx, y, checkz);
        }
    }
    if (changeinx == changeinz)
    {
        roundedVector = new Vector3(x, y, checkz);
        if (map[x, (int)checkz] == 1)
        {
            roundedVector = new Vector3(checkx, y, checkz);
        }
    }
    if (changeinx < changeinz)
    {
        roundedVector = new Vector3(checkx, y, z);
        if (map[(int)checkx, z] == 1)
        {
            roundedVector = new Vector3(checkx, y, checkz);
        }
    }
```

Figure 8- New Enemy Movement Code Example

4.7 - Damage Calculation

The damage is calculated at the end of the player turn method, and before the start of the enemy turn. This allows the player to always move before the enemy. Damage is also recalculated if the enemy is next to the player, so that the enemy can give some damage.

The Damage Calculation code checks for a series of Booleans, and deals damage based on these Booleans. The enemy damage calculation looks for the player damage Boolean that is changed when an enemy is 1 tile away from an enemy. It then gets the enemy that is attacking, and finds the damage stat in its stats script. It then minuses the amount of damage from the players health.

It also checks for if the bow or rock has been used. As the bow and rock are used, these each activate a Boolean for their own damage, and the calculation takes a set damage from the health of the selected enemy.

For melee combat, the script checks for an enemy selection. In melee combat, all the enemies that are in range (directly up, down, left or right) of the player, and creates 4 buttons for these enemies. As these buttons are pressed, the enemy selected is passed into the script, where its health has the player's current melee damage away from it.

```
if (enemy1selected)
{
    if (pc.GetComponent<playerStats>().isstrong == true)
    {
        enemytodamage1.GetComponent<enemyStats>().health = enemytodamage1.GetComponent<enemyStats>().health - (int)(pc.GetComponent<playerStats>().dmg*1.5);
    }
    else
    {
        enemytodamage1.GetComponent<enemyStats>().health = enemytodamage1.GetComponent<enemyStats>().health - pc.GetComponent<playerStats>().dmg;
    }
    if (enemytodamage1.GetComponent<enemyStats>().health <= 0)
    {
        pc.GetComponent<playerStats>().health = pc.GetComponent<playerStats>().health + enemytodamage1.GetComponent<enemyStats>().dmg;
        Destroy(enemytodamage1);
        damagingenemy = null;
        enemyselect1 = null;
        enemyselect2 = null;
        enemyselect3 = null;
        enemyselect4 = null;
    }
    enemy1selected = false;
    enemytodamage1 = null;
    moved = true;
}
```

Figure 9- Melee Combat Damage Calculation Example

During the damage calculation, if the enemy's health ever falls below zero, the enemy is then deleted, and removed from the array of enemies in the level. If the player's health falls below zero, the player dies, and a scene is loaded where the player has a choice to restart the game.

Chapter 5 - Evaluation

5.1 - Questionnaire Design

A questionnaire was attached with the game that was filled out by play testers. The questionnaire was designed mainly with quantitative data in mind so that data could easily be understood in a visual graph format. The questions focused on the randomness of the game, the difficulty of the game, and if there were any problems with how the game has been designed in general.

The questions were designed to get a feel for how well the game had achieved it's likeness to rogue, by asking questions that asked the player to judge a specific part of the game.

There were also parts of the questionnaire that asked the player to give qualitative answers. These were less about giving data about how close the game got to Rogue, but more about providing feedback on how to make the game better in a general sense.

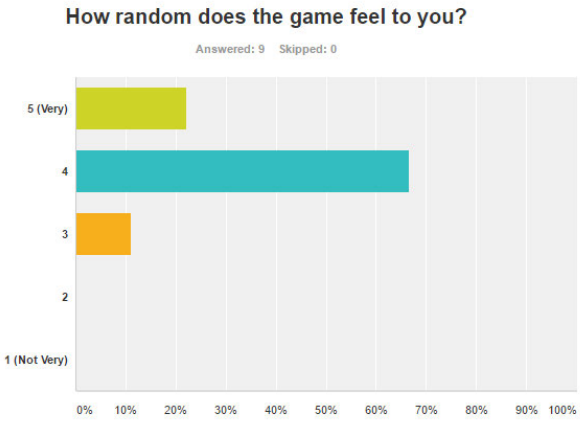
5.2 - Questionnaire Results

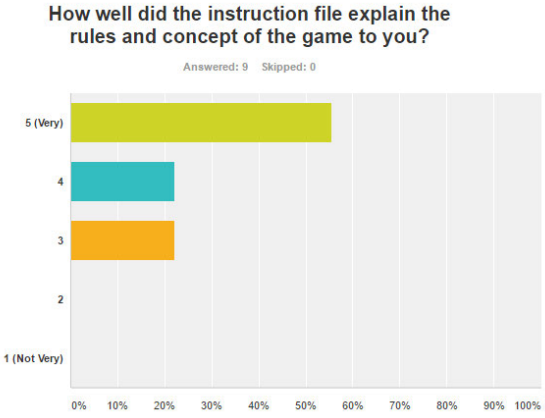
The questionnaire was mainly given to people who were familiar with videogames as whole. People who would class themselves as fans of games. These people were chosen as although roguelikes are part of a niche genre, they are considered quite a hardcore game, and therefore having people who are inexperienced or casual gamers would skew the results.

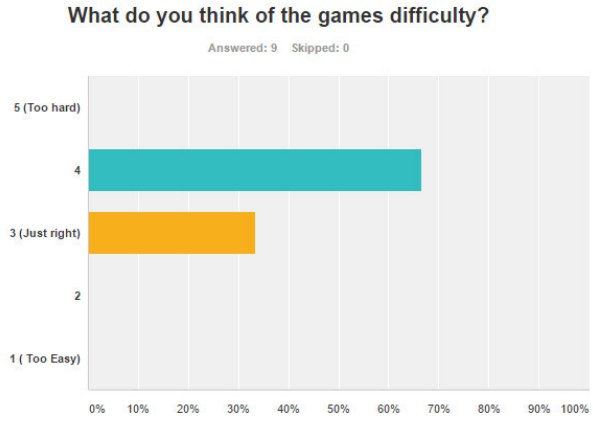
Only a few of the 9 people who completed the questionnaire were actually fans of roguelikes. The users tested the game by running an executable file, and given their own time to play the game. Users were requested to complete at least 3 runs of the game, allowing them to get a feel of the random generation, and the mechanics used. They were asked to play until a successful run if possible, but due to the inherently difficult nature of the game, and the lack of roguelike experience in the testing pool, only a few did.

They were asked to thoroughly read through a readme file before playing the game, as this explained the controls, the mechanics and a few tips.

After their 3 play-throughs (or play-throughs till success) the users were asked to fill in a questionnaire pertaining to their experience. Below are graphs visualising the answers.

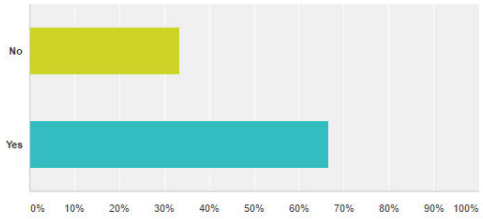






Did you encounter any bugs while playing the game?

Answered: 9 Skipped: 0



Answer Choices	Responses	
No	33.33%	3
Yes	66.67%	6
Total		9

Responses

Fullscreen crashed
4/21/2017 8:50 PM View respondent's answers

enemies would become stuck if the player past an object and the cpu was adjacent to the object, rendering them useless.
4/21/2017 10:54 AM View respondent's answers

Either a blue thing went inside something or disappeared
4/21/2017 1:29 AM View respondent's answers

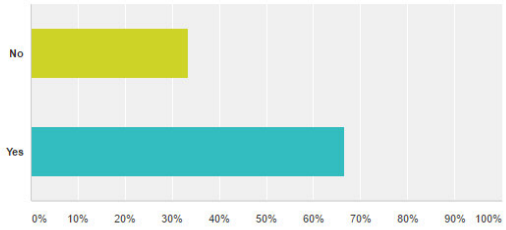
Health didn't do anything? maybe?
4/20/2017 10:43 PM View respondent's answers

item block enemy pathing and ability to shoot enemy
4/20/2017 9:57 PM View respondent's answers

On first try, the game froze, but numerous playthroughs after were perfectly fine
4/20/2017 8:41 PM View respondent's answers

Are there any obvious balancing issues to you?

Answered: 9 Skipped: 0



Bow is OP as fuck cos all enemies are meele
4/21/2017 8:50 PM View respondent's answers

Bow seems a little over powered
4/21/2017 1:29 AM View respondent's answers

Health potion didn't do much against the boss, couldn't outrun it to find other equipment
4/20/2017 11:31 PM View respondent's answers

The bow is strong, so is the push into wall
4/20/2017 9:57 PM View respondent's answers

Bow is godtier
4/20/2017 8:41 PM View respondent's answers

The bow, and enemies seem to spawn close 9/10 times
4/20/2017 8:38 PM View respondent's answers

What would you like to see included with further development?

Answered: 9 Skipped: 0

Responses (9) | Text Analysis | My Categories

PRO FEATURE
Use text analysis to search and categorize responses; see frequently-used words and phrases. To use Text Analysis, upgrade to a GOLD or PLATINUM plan.

Upgrade Learn more »

Categorize as... | Filter by Category | Search responses

howing 9 responses

Drops from killed enemies
4/21/2017 8:50 PM View respondent's answers

development on the graphics
4/21/2017 5:02 PM View respondent's answers

More weapons and some form of animation to be included.
4/21/2017 10:54 AM View respondent's answers

more varied opponents
4/21/2017 1:29 AM View respondent's answers

Something to add character to the game, e.g. sound, GUI
4/20/2017 11:31 PM View respondent's answers

my movement finger is dead, maybe make it less button mashing needed for long movement
4/20/2017 10:43 PM View respondent's answers

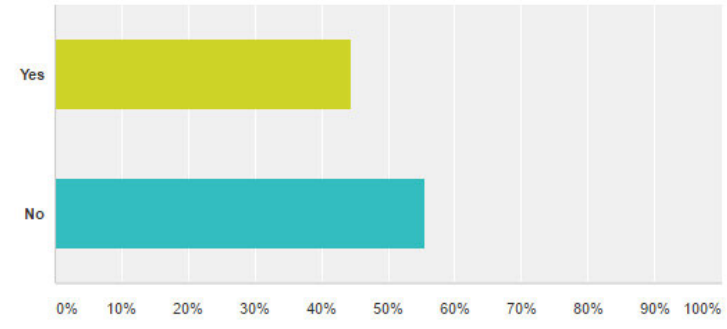
sound and graphics would add a lot more appeal to the game
4/20/2017 9:57 PM View respondent's answers

The items and characters replaced with meshs/models. Otherwise, super fun
4/20/2017 8:41 PM View respondent's answers

Enhanced graphics
4/20/2017 8:38 PM View respondent's answers

Did you manage to win during the runs?

Answered: 9 Skipped: 0



5.3 - Results Discussion

Randomness

The users thought that the game was quite random. Every user who answered the survey gave the game at least a 3 on the randomness scale. 88% of the users thought the game was a 4 or above, which shows that there is a strong randomness to the generation of the game, achieving an objective of having a procedural generated random map, that allows for different experiences every time.

The randomness of the spawning enemies however often caused the player to spawn surrounded by enemies, and have an early loss to a run. One player noted this specifically.

Difficulty

The majority of the users though that the game was slightly too hard. Only 33% of player stated that the game was just at the right difficulty. This lines up almost perfectly with the amount of players who beat the game in their runs for the survey.

Balancing

Two thirds of all players surveyed agreed that there were issues with the balancing of the game. The majority of the comments that came with the balancing discussion were about the bow being overpowered, and one with the health potion being too weak.

Bugs

There were a few bugs discovered whilst playing the game. A few of the users reported bugs that could not be acted upon, or the reasons for which could not be discovered, such as crashes on opening the game, even when the game ran fine for the next attempts.

Some of the responses were not very detailed and therefore could not be used to ascertain where there could be a problem. Some of the reports showed exactly where the problem was, and what would be needed to fix them.

Further Development

The majority of the features asked for in further development of the game were related to the graphics and sounds. While the original rogue had a very simplistic look and no sounds, it is apparent from these results that people of the modern age require much more fleshed out games. The other major thing that cropped up was the inclusion of more varied enemies and items.

Summary of the Results

Overall the general results from the survey was positive towards the game. Although under half of the people surveyed managed to beat the game, as the game is part of the roguelike genre which is considered to be a difficult hardcore players game, this is a reasonable amount of victories, considering the 3 play minimum. People clearly saw the randomness throughout their separate runs of the game, which was a very important part of the game itself.

The results have given good feedback to fix balance issues with the game, find bugs where the game is not working as intended and pave a direction for future development. In the end the results support the idea that the procedural generation of the game, and the content of the game falls firmly in the category of Roguelike.

5.4 - Feedback and Improvements

From the feedback given during the survey, there a few ways the project could be changed and improved upon.

Better Graphics and Audio

Audio and Sprites could be added into the game, to increase the players enjoyment. The games audio is would still be very basic. There would be simple sounds for interactions within the environment, such as picking up items, drinking potions and hitting enemies. The game would have a very simple soundtrack, which will switch between two simple 8 bit tunes, one for exploring and one for when the player is within a certain amount of distance of an enemy.

A tile based graphics system could be integrated. The walls could directly outlining the rooms could have a texture fitting a certain theme, such as cavern walls or trees for a forest. The floor would similarly have fitting tile textures. The main character would have an animated sprite, such as a warrior exploring dungeons, or maybe even a typically enemy character such as a goblin.

More Variety

There could be multiple versions of enemies in the game that follow slightly different AI. The basic enemy of the level could be a Cave Rat, which is just a giant rat with the ability to bite and that is all. The rat will have a high spawn rate, but will only attack the player if they are in a group larger than two. If the Rat is on its own, it will decide to run away from the player, until it runs into another rat. This will then trigger both the rats to move towards the player and attack, even if the player is outside of the second rat's detection range. The Rat will tend to wander in randomly in a small radius of its spawn point.

There could also be a Giant Spider. The Giant spider is a highly territorial creature, attacking both the player, and any other mob that is within its detection range. It detection range is quite small, and it does not move from its spawn point, unless an enemy is detected, or an enemy walks into one of its web traps, of which two randomly spawn within 5 tiles of the

spider. The spider can bite, which has a 25 percent chance to inflict a poison state on the player, or shoot web balls at the player, which do low damage but have a chance to stun.

The cave bat randomly moves throughout the caves, moving wherever it wills. It has low hp and damage, but if it detects the player it has a chance to spawn 2-4 more bats, which can swarm the player if unprepared. The bat can only bite the player for low damage.

The Troll is the pseudo-boss of the cave area, this will replace the boss that spawns by the exit. Killing the troll will result in a high level drop. The troll can throw rocks that spawn throughout the cave, or he can hit the player with his club for very high damage. The troll has very large hit points.

Extra potions and scrolls could be added. An antidote which would heal the player of the newly introduced spider poison. An invisibility potion which makes the player undetectable to enemies for an amount of turns (undoes if the player attacks an enemy, or enters a spider web trap). An oil flask which the player can throw to create a pool of oil. (The oil spreads out amongst tiles and if hit massively increases the radius of the fireball scroll). A poison flask which poisons enemies when thrown by the player.

Bug Fixes/Improvements

All of the items added into the game did not have “is Trigger” applied to their sphere collider. This caused these items to block enemy pathing and the players ranged attacks.

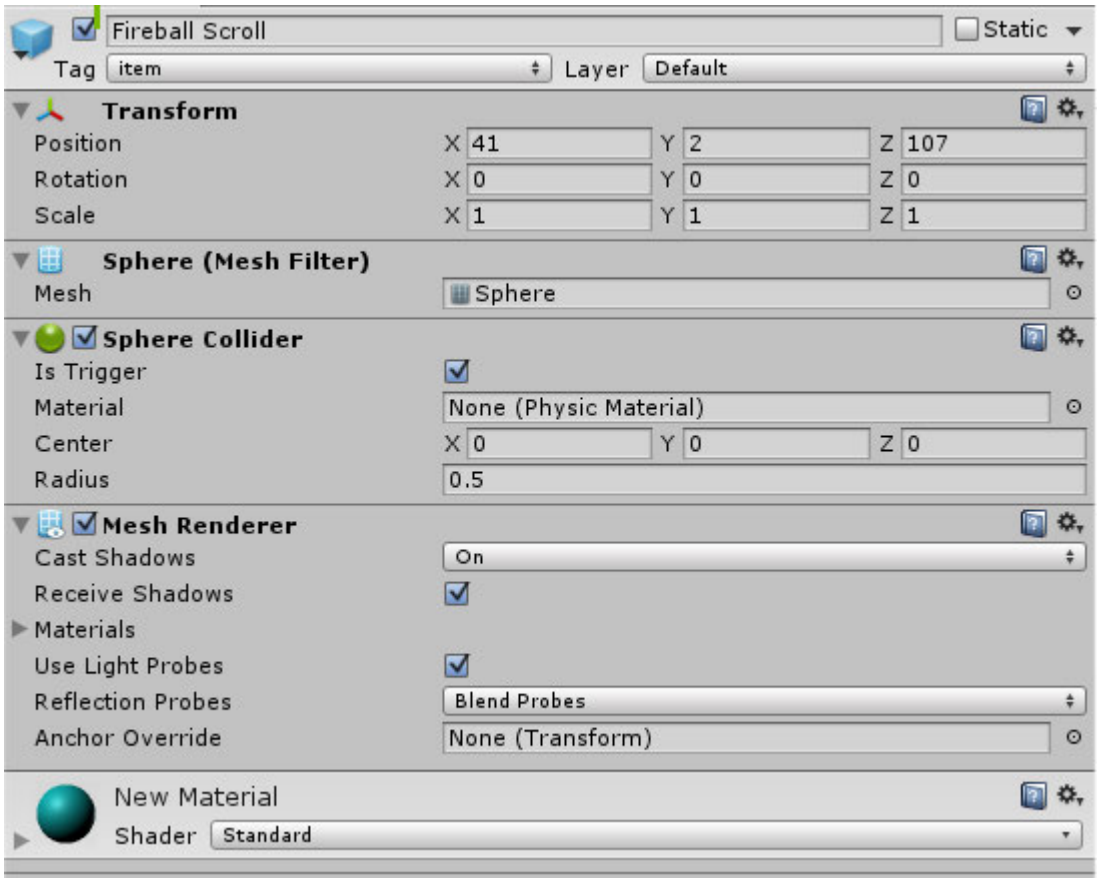


Figure 10- Item was not trigger bug, now fixed

The bows range was dropped, and its damage reduced, making it much less able to kill every enemy before they reached the player. Health potions were buffed as well, with their HP regain being doubled.

5.5 - Personal Reflection of the Project

This section of the report is a person reflection on both the development of the project, and the fulfilment of the goals set out to begin with.

Research

Using various online library's I was able to find many research articles about the various systems I have used while developing the game. With my own personal knowledge of roguelikes, and the help of various game score aggregation sites I was able to pinpoint various roguelikes containing features that I could implement into the game to satisfy the aims of a roguelike. I would have liked to have done more research into procedural generation methods in an attempt to mesh together various different methods to create even more interesting randomly generated maps, rather than just the simple random cave generation I ended up using.

Design

The design of the product fit reasonably well with how the project turned out in the end. By researching various methods of random generation, I was able to find a method I wanted to use that I was able to implement into the game itself. I was confident that the method of generating the level I chose would create levels that were appealing and random enough to offer a different experience each time. This was ratified by the players of the game all saying that the game was quite to very random. The idea to use the A* algorithm for the enemy pathfinding did not work out very well from the design stage. While I am still fairly satisfied with how the alternative pathfinding worked out, providing a working response, and allowing the game to actually function, the lack of A* is slightly disappointing. However, the A* was not a complete failure. The majority of the A* function worked, with the program finding the best path through the level after the initial step, which was a good insight into pathfinding in general and strong practise from creating code from pseudocode.

Implementation

Choosing Unity to code the game in was a very successful decision. Being able to use Unity's Scripts as an object oriented class system allowed a dynamic use of various different parts of

codes, all linking together in a fairly streamlines Game Manager. While the game manager did become a little code heavy in the enemy turn and player turn functions, which could in hindsight have been moved into separate scripts, this did allow to turn Unity's frame centric game loop to become forced into a turn based method.

Due to my experience with C# and my preference for programming over a blueprint system like Unreal Engine, I feel like choosing Unity was definitely the best choice. I got to develop my skills with Unity's specific syntax and methods, as well as practicing C# and object oriented programming itself. I am mostly happy with how my scripts turned out, being especially pleased with the generation of the level itself. The failure to fully implement A* was disheartening, but the workaround using Unity's functions, I feel was a solid piece of code. I hope to return to the A* algorithm in the future, and figure out fully where the problem lay with the enemy generating the first step of the path all the way to the south. I feel the rest of the path finding code was very strong, and could easily be applied to other games I could create if this bug was ironed out.

Summary

The aim of this project as a whole was to create a vertical slice of a procedurally generated roguelike game. As discussed during the reflection, I feel like this aim has generally been reached. I am very happy with the finished product, especially after, or maybe in spite of the mishap with the A* algorithm implementation. I am also very pleased with the increase in scope the feedback has given me, and even my own ideas for future development of the project, or a similar one, after this vertical slice has been completed.

Chapter 6 - Conclusion

6.1 - Final Remarks

This project set out to create a procedurally generated roguelike game that would implement procedural generation and enemy AI to produce a random feeling game that gave the player a different experience each time. It was crucial to the project that the procedural generation gave the players content that felt like a new area each time. If each time the level was created, and it felt too much the same, a player could quickly become disinterested and bored. The use of this random generation, and the different experiences it gave each time created a game that gave a varied, if sometimes a little difficult, experience that players seemed to enjoy.

6.2 - Further Work

Procedural Generation

More work could go into the procedural generation of the levels. Extra tiles could be added to the spawning scripts that happen after the main map is added, that could provide traps or even buffs for the player. Other implementation of procedural generation could be used, perhaps even having an agent system in place, to make smaller cave systems, but connect them by corridors to add a little variety to the world.

Enemy and Item

With further development, the various enemy types discussed in the evaluation could definitely be implement. Having more enemies and items would allow more of a random feel to the game. As enemies are randomly spawned, with different enemy types this would make it feel much more varied run to run.

Pathing

Extra work could definitely go into the pathing feature of the game. Enemy that are not currently chasing the player could randomly generate waypoints around them for them to path towards until the player comes into view. This would give the world a much more living and realistic feeling. Enemies could also continue towards where they last saw the player if the player manages to break their line of sight. At the current time they simple stop moving if the player breaks line of sight, and at least one of the testing group said this could be improved, even considering this a bug.

Graphics and Sound

The game very much suffers from the basic art style and lack of sound that it uses. Although this is very in keeping with the original Rogue, it is clear from the player surveyed that keeping such a simple design was a mistake in the modern era of games. Even a simple 8bit texture tile system, and 8 bit sounds would have pleased the players a lot more.

Appendix

Terms of Reference

Procedurally Generated Rogue-like Game

Course-Specific Learning Outcomes

• To study the history of computer games, game genres, game structures and game design principles and to use the skills acquired to specify and evaluate new game applications;

• To become knowledgeable in the use and development of computer graphics software/game middleware tools and to be able to apply this knowledge to the implementation of real-time interactive systems;

• To learn structured approaches to computer programming;

• To learn how the theories and techniques of behavioural systems can be used to enhance the playability and sophistication of computer games;

• To study a range of mathematical tools and techniques and to be able to use these, in particular, to solve problems in the area of game modelling and animation; and

• To gain an appreciation of the multidisciplinary environment in which commercial games are designed and produced and to acquire skills in project management and team working.

Project Background

Since the original Rogue back in 1980, many games have employed the strategies that this game used for its core development. Many "Clones" of Rogue have been developed over years, with varying degrees of similarity. Games started to pick and choose elements of Rogue as they went on, taking thing such as perma-death, random generation and turn-based combat.

Games that only follow this premise loosely are referred to as "Rogue-lites", while games that take a large number of these concepts are considered "Rogue-likes", named for the popularity of Rogue.

To create a Rogue-Like, the game generally has to hit the same elements as the original Rogue. Features such as the ASCII art however are not necessary to be considered a Rogue-like.

The game generally requires:

- Random tile-based levels.
- Permadeath.
- Turn-based combat.
- Emergent Gameplay.
- Fantasy Themes.
- Inventory System.

The original Rogue started a huge trend of procedural generation in video games. Procedural generation is the act of creating random levels using a variety of different methods. It uses an algorithm to create levels as opposed to having a human design them manually.

There are many ways of generating random tile-based levels, which have been used in various different Rogue-like games. Such examples include:

- The search based approach
- Binary Space Partitioning
- Agent based growing
- Cellular Automata
- Fractal and noise based generation.

Rogue itself used binary space partitioning to create its random dungeons, however many Rogue-likes have not followed this and have used these various other ways of procedural generation.

Each way of procedural generation has its own pro's and con's, which have to be considered when designing the game, as each method creates very different levels which all need to be approached with a different design in mind.

With the dawn of easy self-publishing due to the rise of high-speed internet, many independent developers with very small or even 1 man teams can now create and sell their games, while still reaching a wide audience. With such a small team often comes the problem of creating enough content to make their games worthwhile and keep the player engaged.

Procedural generation has been very helpful in this regard with many games choosing to procedurally generate their game content, and focus solely their design on mechanics, enemies and challenges.

Related Work

There have been many games that have used procedural generation to generate game content, with the majority being independently developed like Minecraft and various roguelikes such as The Binding of Isaac or Nuclear Throne, and some large publishers such as Nintendo and the Pokémon Mystery Dungeon series.

Cellular Automata has been used in many games and is a popular method of content generation. A piece of work evaluating a method of Cellular Automata has been created by the IT University of Copenhagen [Johnson, Yannakakis, Togelius. (2010).] The Delft University of Technology also created a piece surveying the various uses for Procedural Content Generation [Hendrikx, et al. (2013).]

Aim

A1- To gain a better understanding of the various ways to procedurally generate game content.

A2- To practise skills in coding game logic.

A3- Create a strong algorithm for generation of procedural game content that creates an enjoyable experience.

A4- To create a game that is structured so that new levels and enemies can be added with little difficulty beyond their design.

Objectives

O1- Study the various methods of procedural generation.

O2- Decide upon and practise with a given method.

O3- Develop a suitable Algorithm/Pseudocode for the generation of content using this method.

O4- Select a Game Engine to create game content using this method.

O5- Design content that could fit around this engine and method.

O6- Gain feedback on the randomness of the game using play testing.

O7- Test the usability of the game via play testing.

O8- Having created a working game, review findings with feedback and evaluate the development process.

Problems

There are many potential problems in the development cycle of a videogame. The code could be difficult to implement within the engine, too large a scope at the design stage of the game could run into problems with time management and the ability to deliver the original concept, leading to cuts or maybe rush unbalanced game elements.

For a Roguelike specifically, if the algorithms for procedural generation are not polished to the right amount, it could create levels that are not very fun to play, or even worse simply unbeatable. Parts of the level could be inaccessible if the code is not written properly, and this could lead to exits being unreachable, or certain elements being out of reach.

With the idea of random chance, and the turn based combat system, if many enemies spawn at once, certain levels might become completely unbeatable. With a roguelike this is expected to happen every now and again, but too often and it ruins the games replayability.

Due to the inherent random nature of the game, content will not only need to be designed very specifically to avoid balance issues, but it will also need to be implemented in very specific ways. The game will need to adhere to a steadily rising difficulty setting, and the balancing of the game will need to adhere to this while remaining interesting and fun.

Solutions

In this project the Rogue-like game will be created with the Cellular Automata method of procedural generation in mind. This solves the industry problem of independent developers not having enough time and resources to develop all of the game content by hand, and thus helps them create large amounts of content for their game.

Timetable and Deliverable

- Completion of first draft of Terms of Reference 10/10/2016.
- Completion of first draft of Ethics form 10/10/2016.
- Completion and submission of Terms of Reference 16/10/2016.
- Complete Very Basic Prototype of Level Generation 17/10/2016.
- Completion and submission of Ethics form 23/10/2016.
- Complete Basic Mechanics of Game (Movement, Health etc.) 31/10/2016.
- Completion of first draft of Literature Survey 06/11/2016.
- Complete Level Generation Code and Item/Enemy Spawning Code 14/11/2016.
- Completion and submission of Literature Survey 20/11/2016.
- Completion of first draft of Product Design 27/11/2016.
- Completion of Enemy Designs and Stats 05/12/16.
- Completion and submission of Product Design 11/12/16.

- Completion of first draft of Evaluation Design 22/01/17.
- Completion of Project 29/01/17.
- Completion and submission of Evaluation Design 29/01/17.
- Completion of first draft of Report outline 05/02/17.
- Completion and submission of Report outline 19/02/17.
- Completion and submission of Draft Slides 05/03/17.
- Hold Practise Presentation 13/03/17.
- Submit final Report and Product 23/04/17.
- Hold Presentation and 24/04/17.

Required Resources

The game will require a game engine, a scripting language, and some photo editing software for the creation of the in-game assets. An Audio-editing tool will also be necessary for the creation of audio cues and sounds.

The review and evaluation of the game will require willing participants to test the game.

ETHICS CHECKLIST

This checklist must be completed **before** commencement of **any** research project. This includes projects undertaken by **staff and by students as part of a UG, PGT or PGR programme**. Please attach a Risk Assessment.

Please also refer to the University's Academic Ethics Procedures; Standard Operating Procedures and the University's Guidelines on Good Research Practice

Full name and title of applicant:	[redacted]	
University Telephone Number:		
University Email address:		
Status: All staff and students involved in research are strongly encouraged to complete the Research Integrity Training which is available via the Staff and Research Student Moodle areas	Undergraduate Student ☑ Postgraduate Student: Taught ☐ Postgraduate Student: Research ☐ Staff ☐	
Department/School/Other Unit:	Computing, Mathematics and Digital Technology	
Programme of study (if applicable):	Computer Games Technology	
Name of DoS/Supervisor/Line manager:	Huw Lloyd	
Project Title:	Procedurally Generated Roguelike Game	
Start & End date (cannot be retrospective):	September 2016 - April 2017	
Number of participants (if applicable):	-	
Funding Source:	-	

Brief description of research project activities (300 words max):

The research project is to craft an algorithm that will procedurally generate levels and game content. The project will be created using Unity, and Visual Studio with C#. The Procedural Generation will be done using the process of Cellular Automaton.

	YES	NO
Does the project involve NHS patients or resources? If 'yes' please note that your project may need NHS National Research Ethics Service (NRES) approval. Be aware that research carried out in a NHS trust also requires governance approval. Click here to find out if your research requires NRES approval Click here to visit the National Research Ethics Service website To find out more about Governance Approval in the NHS click here	☐	☑
Does the project require NRES approval?	☐	☑
If yes, has approval been granted by NRES? Attach copy of letter of approval. Approval cannot be granted without a copy of the letter.	☐	☐

NB Question 2 should only be answered if you have answered YES to Question 1. All other questions are mandatory.	YES	NO
1. Are you are gathering data from people?	☑	☐
For information on why you need informed consent from your participants please click here		
2. If you are gathering data from people, have you:	☐	☐
a. attached a participant information sheet explaining your approach to their involvement in your research and maintaining confidentiality of their data?	☑	☐
b. attached a consent form? (not required for questionnaires)	☐	☑
Click here to see an example of a participant information sheet and consent form		
3. Are you gathering data from secondary sources such as websites, archive material, and research datasets?	☐	☑
Click here to find out what ethical issues may exist with secondary data		
4. Have you read the guidance on data protection issues?	☑	☐
a. Have you considered and addressed data protection issues – relating to storing and disposing of data?	☑	☐
b. Is this in an auditable form? (can you trace use of the data from collection to disposal)	☐	☑
5. Have you read the guidance on appropriate research and consent procedures for participants who may be perceived to be vulnerable?	☑	☐
a. Does your study involve participants who are particularly vulnerable or unable to give informed consent (e.g. children, people with learning disabilities, your own students)?	☐	☑
6. Will the study require the co-operation of a gatekeeper for initial access to the groups or individuals to be recruited (e.g. students at school, members of self-help group, nursing home residents)?	☐	☑
Click for an example of a PIS and information about gatekeepers		
7. Will the study involve the use of participants' images or sensitive data (e.g. participants personal details stored electronically, image capture techniques)?	☐	☑
Click here for guidance on images and sensitive data		
8. Will the study involve discussion of sensitive topics (e.g. sexual activity, drug use)?	☐	☑
Click here for an advisory distress protocol		
9. Could the study induce psychological stress or anxiety in participants or those associated with the research, however unlikely you think that risk is?	☐	☑
Click here to read about how to deal with stress and anxiety caused by research procedures		
10. Will blood or tissue samples be obtained from participants?	☐	☑
Click here to read how the Human Tissue Act might affect your work		
11. Is your research governed by the Ionising Radiation (Medical Exposure) Regulations (IRMER) 2000?	☐	☑
Click here to learn more about IRMER		
12. Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants or will the study involve invasive, intrusive or potentially harmful procedures of any kind?	☐	☑
Click here to read about how participants need to be warned of potential risks in this kind of research		
13. Is pain or more than mild discomfort likely to result from the study? Please attach the pain assessment tool you will be using.	☐	☑

Click here to read how participants need to be warned of pain or mild discomfort resulting from the study and what do about it.		
14. Will the study involve prolonged or repetitive testing or does it include a physical intervention?	☐	☑
Click here to discover what constitutes a physical intervention and here to read how any prolonged or repetitive testing needs to managed for participant wellbeing and safety		
15. Will participants to take part in the study without their knowledge and informed consent? If yes, please include a justification.	☐	☑
Click here to read about situations where research may be carried out without informed consent		
16. Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?	☐	☑
Click here to read guidance on payment for participants		
17. Is there an existing relationship between the researcher(s) and the participant(s) that needs to be considered? For instance, a lecturer researching his/her students, or a manager interviewing her/his staff?	☐	☑
Click here to read guidance on how existing power relationships need to be dealt with in research procedures		
18. Have you undertaken Risk Assessments for each of the procedures that you are undertaking?	☑	☐
19. Is any of the research activity taking place outside of the UK?	☐	☑
20. Does your research fit into any of the following security sensitive categories: • commissioned by the military • commissioned under an EU security call • involve the acquisition of security clearances • concerns terrorist or extreme groups If Yes, please complete a Security Sensitive Information Form	☐	☑

I understand that if granted, this approval will apply to the current project protocol and timeframe stated. If there are any changes I will be required to review the ethical consideration(s) and this will include completion of a 'Request for Amendment' form.

☑ have attached a Risk Assessment
☑ have attached an Insurance Checklist

If the applicant has answered **YES** to ANY of questions 14 – 17 then they must complete the MMU Application for Ethical Ap[redacted]

Signature of Applicant: _____ Date: 28/10/2016 (DD/MM/YY)

Independent Approval for the above project is (please check the appropriate box):

Granted

☑ I confirm that there are no ethical issues requiring further consideration and the project can commence.

Not Granted

☐ I confirm that there are ethical issues requiring further consideration and will refer the project protocol to the Faculty Research Group Officer.

Signature: _____ Date: 22/11/16 (DD/MM/YY)

Print Name: _____ Position: _____

Approver: Independent Scrutiniser for UG and PG Taught/ PGRs RD1 Scrutiniser/ Faculty Head of Ethics for staff.

References

Bramwell, T. (2013). *Spelunky PC Review.* Available: http://www.eurogamer.net/articles/2013-08-08-spelunky-pc-review. Last accessed 21/11/2016.

Cellar Door Games. (2013). *Rogue Legacy.* Available: http://www.cellardoorgames.com/roguelegacy/. Last accessed 21/11/2016.

Dilks, J. (2012). *FTL: Faster Than Light Review.* Available: https://www.videogamer.com/reviews/ftl-faster-than-light-review. Last accessed 24/11/2016.

Galin, E., Peytavie, A., Maréchal, N. and Guérin, E. (2010), Procedural Generation of Roads. Computer Graphics Forum, 29: 429–438.

Gaslamp Games. (2011). *About.* Available: https://www.dungeonsofdredmor.com/#about. Last accessed 21/11/2016.

Greuter, S., Parker, J., Stewart, N and Leach, G. 2003, 'Real-time procedural generation of 'pseudo infinite' cities', in Proceedings of the First International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, Melbourne, 11-14 February 2003.

Hendrikx, et al. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*. 9 (1), 1:8-1:11.

Johnson., Yannakakis., Togelius. (2010). Cellular automata for real-time generation of infinite cave levels. *PCGames*. 10, 1-4.

Lait, J. (2009). *Powder.* Available: http://www.zincland.com/powder/?pagename=about. Last accessed 24/11/2016.

McMillen, E. (2011). *BECOME A TRUE HERO!.* Available: http://bindingofisaac.com/. Last accessed 24/11/2016.

Meunier, N. (2011). *Dungeons of Dredmor Review.* Available: http://uk.ign.com/articles/2011/07/29/dungeons-of-dredmor-review. Last accessed 21/11/2016.

Parkin, S. (2014). *The Binding Of Isaac: Rebirth Review.* Available: http://www.eurogamer.net/articles/2014-11-12-the-binding-of-isaac-rebirth-review. Last accessed 24/11/2016.

Shaker, N., Togelius, J., and Nelson, M. (2016). Constructive generation methods for dungeons and levels . In: Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer. ISBN 978-3-319-42714-0. P33-45.

Shea, C. (2013). *Rogue Legacy Review.* Available: http://uk.ign.com/articles/2013/07/26/rogue-legacy-review. Last accessed 21/11/2016.

Subset Games. (2011). *FTL.* Available: http://www.ftlgame.com/. Last accessed 24/11/2016.

van der Linden, et al. (2013). Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*. 6 (1), 2-6.

Yu, D. (2009). *What is Spelunky?.* Available: http://www.spelunkyworld.com/whatis.html. Last accessed 21/11/2016.

Xiao, C., Hao, S. (2011). A*-based Pathfinding in Modern Computer Games. *IJCSNS International Journal of Computer Science and Network Security*. 11 (1), 125-130.