The Cycle:

Furthering Independence as a Game Designer

Ian Campbell

Hendrix College

A capstone project submitted for the degree of

Bachelor of Computer Science

Hendrix College 2022

Acknowledgements

The Computer Science Department - Hendrix College

Play-Testers - Dylan Campbell, Madison McGuire, Connor Escajeda, and Cooper Elliott

Abstract

For the past two years, I have worked in game development, and during this time, I have never been able to work on a passion project of my own. This project provided me with both the motivation and time in order to do this. My goal for this project was to create a third-person 3D Metroidvania-style game using Unity starting from nothing in order to further my skills as a developer. Specifically, I wanted to create an interactive world with resource management, NPC trading, and enemy combat with an AI that learns based on player gameplay. Beyond this, I intended to create a large portion of the assets myself (music, 3D models, textures, shaders, etc.). I did not have experience in this sort of creative process, and it was something I wanted to get better at as I progressed throughout development. Although I did not reach the level at which I had hoped at the beginning of the semester, I was able to learn a ton about 3D modeling and Unity as a whole.

Contents

List of Figures

Chapter 1

Introduction

The inspiration behind this project is found in two games, *The Legend of Zelda: Ocarina of Time* (OOT) [15] and an untitled game made as the final project in my introduction to computer science class (Fig. 1). OOT offered incredible world-building and interaction with the non-player characters (NPCs), so it was very easy to get into the game and care about it. The untitled project (it itself, inspired from *A Dark Room* [14]) was where I first fell in love with game development and computer science as a whole. I started working on it when the COVID-19 pandemic was weeks old (and going outside was taboo) so I had lots of free time to pour into it. Working with simple text manipulation, some loops, and a lot of if statements I was able to create a world that a player could move around in and explore. Ending my first class, I was amazed by the progress I was able to make considering just a few months prior I didn't know the first thing about coding. It seemed fitting to end my college career similarly, using my project as the template and seeing how far I could get.

1.1 Inspiration into Development

The intention of this project was to recreate this game, but in the 3D interactive world as described in the abstract. The story, enemies, mechanics, and map were all a basis from which I had built this game (Fig. 2). I had to rework some things since I was able to include much more in a 3D world, and not include other things due to limitations that will be discussed. OOT became a large inspiration when thinking of how to translate my project into a three-dimensional space since it allows

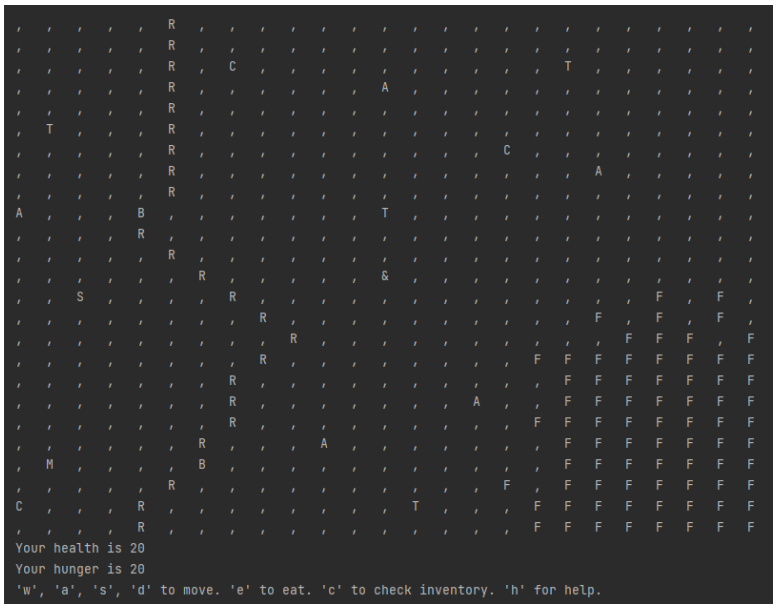

Figure 1.1: Map from *Untitled Project*

for open-world exploration while providing a linear story with interactable NPCs to provide the player with direction. It also has a wide variety of enemies, and combat seems both intuitive and engaging. *A Dark Room* also contains exploration, but mechanics for hunger and thirst, along with managing storage space, make it much more difficult. Along with this, the story is not served on a platter to the player, but slowly unfolded through exploration and gameplay. These aspects from both games were very important in considering this project as balancing survival with exploration, interesting enemy mechanics, a good AI for both enemies and NPCs, and storage management were the focus points of this project.

1.2 Personal Development

A project like this is something I had been wanting to do for a long time, but being in college and working over breaks it has never been easy to find the time or motivation to work on it. This project provided me with an opportunity for both. Beyond just making the game in a 3D world, this was also a great opportunity for personal

Figure 1.2: Current map from Unity version

development in areas of game development that I was not as experienced in. I had zero experience in aesthetics and asset creation, and I truly only had a slightly below-surface-level knowledge of Unity's workspace. I wanted to finish this game and have modeled some of my own objects and have them all colored to a complementary palette, create my own music, and generate my own animations.

Chapter 2

Background

2.1 Expanding Knowledge

While I have worked on 3D games in the past, it had been over a year and it was only for a small amount of time. While many of Unity's [16] features carry over from 2D, there are some added layers of complexity that I had no experience in, such as lighting, shaders, and world-building to name a few. Understanding how to utilize these tools was necessary for the game to look both good and coherent. I also had very little experience when it comes to 3D modeling or creating music. Despite this, I felt very inclined to make this something I created myself. For the scope of this game, modeling all objects myself would have been impossible given the time constraints. The same applied to developing a soundtrack. For all outside resources (except for player animations and model), the Unity Asset Store [17] was used.

2.2 Developing AI

There are two types of NPCs, townsfolk and enemies. Townsfolk are the people who live throughout the world in towns. There are four enemy types that are found in specific areas. The original goal was that they would keep an overall 'memory' of the player and their actions, so if they do good by slaying monsters and helping the town out with quests, merchants will be more inclined to lower prices; However, if the player attacks townsfolk, then merchants will raise prices or avoid the player entirely. A similar idea was applied to the enemies in the game. They would learn

the players fighting style and respond appropriately. For example, if the player favors ranged weapons over melee, the enemies will have more ranged fighters and vice versa. They will also tend to stay away from areas the player frequents, while also taking advantage of areas the player does not go to. Unfortunately, this concept was not able to be applied to the final product. Issues with the 'memory' of the player were not able to be figured out in time and were scrapped off the final product.

2.3 World Overview

The player starts outside their house, facing north towards the middle village. Within the village are two intractable NPCs, the old woman and the merchant. The merchant offers food, potions, and weapons for coins. The old woman offers exposition and guidance for the player throughout the game. She sends the player off on quests and gives small hints on how to complete certain areas.

Figure 2.1: View from atop the player's home of The Fields

2.3.1 The Fields

The Fields are the main play area of the game. It is a large area that comprises most of Fig. 2 and is characterized by hills covered in green grass with short trees that occasionally bare fruit. The fields contain important small areas such as towns,

enemy campsites, and abandoned houses. Being so large and having easily available food, it is a relatively safe area for players where they can let their guard down and explore the world. It is here that the first special item can be found, the lantern. This allows a light to be toggled on the player and provides them sight in the cavern.

Figure 2.2: Player in the market of the main town

2.3.2 The Caves

There are three caves [4] in the world; two have no extra hidden interior area and contain some treasure, and one that leads to a much larger cavern and will serve as the player's first dungeon. This cavern will include new enemy types specific to the cave. Mostly comprised of Gronchs (a peaceful enemy, unless harmed) and a single earth elemental (actively angry), it is more dangerous than the fields the player is used to, but not too difficult. The purpose of this area is just a bit of disorienting exploration. Even though the layout is simple, the new dark environment makes it easy to get mixed up. In one room of this cave will be a special prize, the snorkel. This item will allow the player to enter the river and swim freely without losing health, allowing them access to the third portion of the game.

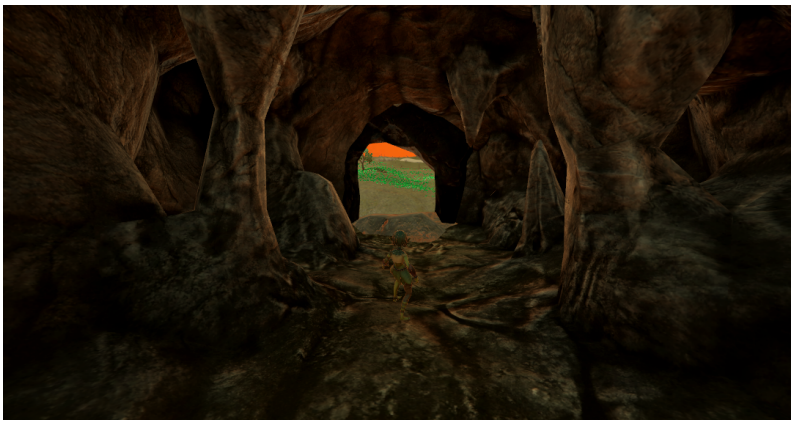

Figure 2.3: View from inside one of the cave entrances

2.3.3 The River

Now equipped with the snorkel, the player can safely enter the second dungeon, the underwater ruins. This is a puzzle-focused area where the only enemy is how quickly the player can solve the puzzle before they starve to death. Using runes of different values inserted into pedestals of different shapes, the player can manipulate the water level, allowing them to swim up or down to new areas. In this area is the knockoff lens of truth (KLoT), an item taken directly from OOT that allows for hidden things to be seen. This item is required to see a pedestal that allows them to raise the water level to the very top of the exit.

2.3.4 The Forest

The forest can be seen in the bottom right corner of Fig. 2 and is considered the third dungeon area. It is comprised of tall trees and a fog that thickens as the player walks deeper inside. Using the KLoT, a series of arrow-shaped signs give the player direction on how to navigate the fog. Straying too far off the path will result in the fog becoming more opaque until the player's screen is almost completely gray and they teleport outside of the forest from 'getting lost'. At the end of the forest is a house with the old woman, where she gives the final instructions and a chest containing the medallion, the last special item.

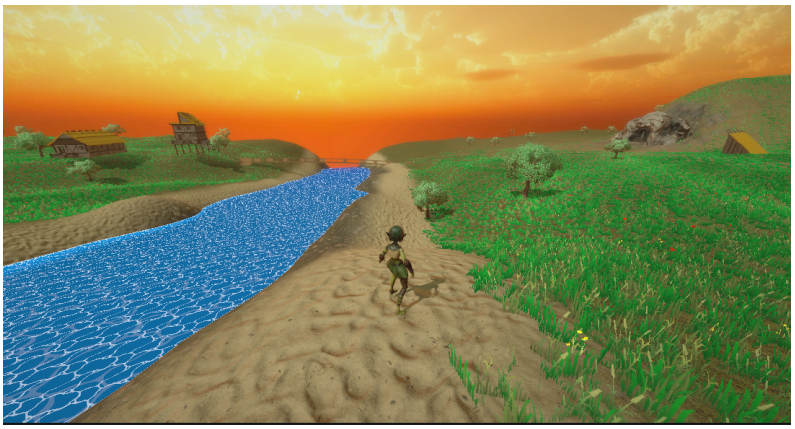

Figure 2.4: Player standing beside the river

2.3.5 Story

The story is not fully incorporated into the final product, sadly. The game should start with a bit of expositional text telling the player "your head hurts, the wind pushing you north into town. The last thing you remember is talking to the old woman.". As the game progress and the player talks with the old woman more, she makes small comments that make it seem like she knows the player, and she gets more hateful as well. In the forest, talking with her reveals information about someone seemingly close to the player being buried, and the woman doubts the player will be able to break "the cycle". Then, in the final boss fight, if the player was able to defeat it they would wake up, visit their loved ones grave and place the medallion down, then walk away. If they fail the fight then the area would collapse in on both the player and boss and the game would restart from the beginning with the player greeted with the same text.

2.3.6 The End

At the cemetery, if the player interacts with the correct grave, found by following the arrowed signs starting from the correct position, then nothing happens. At this point, the grave would open up revealing a staircase that would lead to the final boss fight

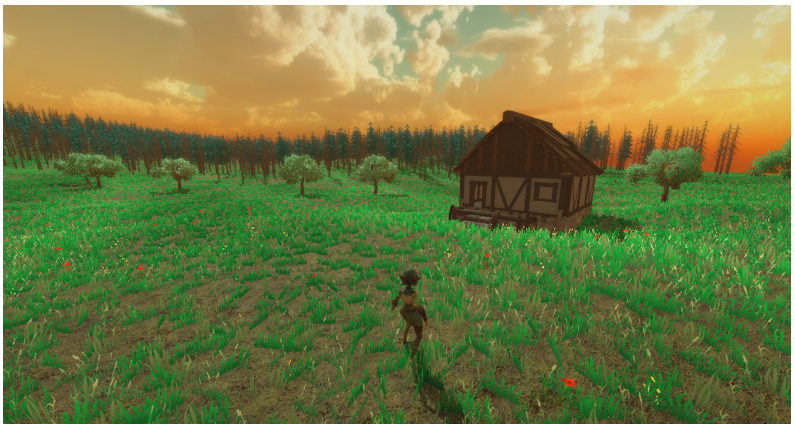

Figure 2.5: Forest in the distance

with two endings but unfortunately, I ran out of time before I was able to implement this. If the player picks the wrong grave, a skeleton appears in front of them.

2.3.7 Purchasing and Currency

Throughout the world, the player can collect several items such as stones, branches, and fruits. Some important items that are unable to be found in the world will be able to be bought at the market that can be found in each town such as the bow, health potions, and extra food. Transactions will be mediated by gold coins, which may be obtained from chests or killing monsters.

2.4 Unity Lingo

There are some important aspects of Unity that are necessary to know to fully understand this paper.

2.4.1 Unity Space

A gameobject is the most important aspect of Unity games. Each one has a name, a "tag" (a string allowing classification to a certain group), a "layer" (an integer allowing classification to a certain group), and a "transform"(a collection of the gameob-

ject's position, rotation, and scale). They can be used as both parents and children of other gameobjects, and have the ability to have "components" (a wide variety of helpful tools and aspects that alter how the gameobject interacts in the world) attached. The most useful component in this game is a "collider", which allows for physics interactions between other gameobjects. Colliders can either be a trigger (do not interact physically with objects, but detects when another collider enters its space) or not (prevents other colliders from entering its space) and come in basic shapes like cubes or spheres, or can match the mesh of their gameobject. "Raycasting" is a helpful physics tool that allows a ray to be shot from a specified position and rotation for a determined distance. These rays will return the first collider they hit, and can even be specified to ignore colliders belonging to gameobjects a part of a certain layer (or look specifically for a layer using bit-shifting).

2.4.2 Unity Code

Scripts written in C# can also be attached as components and can include public variables that can be assigned in the Unity UI. Scripts naturally come with a "Start" function (Which is called once, as soon as the game is run) and an update function (called once every frame). There is one script that is the most important, the GameManager. A script with this name is seen, both by developers and Unity, as the major script that contains the most prominent functions and loops, and can also provide connectivity between scripts. Another important scripting factor are "Coroutines" which, similar to the update function, are called repeatably separate from the script that it finds itself in. These can be dangerous as without a specified wait time between calls, can crash the Unity system with an infinite loop.

2.4.3 Animations

Animations in unity are made much easier by the "Animator" window. This drag-and-drop system lets the user easily create new states (that contain a single animation) and directional transitions between states. Transitions occur when certain conditions are met which can be a multitude of data types, although in this project they are all booleans. Transitions can also consist of an "exit time", which controls how much of

Figure 2.6: Example of script component and its public variables

the animation must be played before it can transition. This is useful for animations such as jumping, where the animation must be seen longer than when the button is pressed.

2.4.4 Canvases

Canvases are UI elements within unity that, when active, will constantly display information within the view of the player, overlayed on top of everything else. They allow simple things like text or images to be shown, as well as interactable objects like sliders and buttons.

Chapter 3

World Building

There are four walkable zones in the game, two terrains and two large gameobjects. The first terrain is the main area comprising of the fields and the forest. Second, is the terrain that are the mountains that surround the world. The two gameobjects are the first and second dungeon, the cavern and the water ruins. To get to different areas, a teleporter script was used. Upon collision between the player and invisible gameobjects with this script, it would teleport the player to a specified position and change certain aspects of the world such as skybox, environmental lighting, music, and activation of other gameobjects (this was done to help not render hundreds of unnecessary objects or have AI run where unnecessary.

3.1 Terrain Editing

Unity's built-in terrain tool is extremely useful and powerful [12]. The most useful sub-tools for this game were sculpting terrain height, painting textures, and painting trees and grass. All three are possible by using differently shaped brushes and being able to change their size and strength. Much like painting on the screen, I was able to create hills, mountains, and valleys. I could paint different textures [13] onto the terrain to get sandy paths, rocky-looking mountains, and green fields. Tree and grass painting allows for the large-scale placing of 3D and 2D objects in a quick and specific manner. This was actually made easier by using an outside tool, Vegetation Spawner [18]. It allowed for the random placing of trees and grass specific to the textures already painted on the terrain, so I wouldn't have to paint the entire thing myself

and I wouldn't have to worry about trees spawning in the river or in towns, or the wrong trees in the wrong area (since the forest uses different trees).

3.2 Main Terrain

This is by far the most diverse area. Whereas the other areas mostly serve a single purpose, the main terrain contains towns, enemy campsites, the forest, the graveyard, abandoned houses, loot caves, and the river. All of these sub-areas in one place help create a world that both feels diverse and easily explorable.

3.2.1 Towns

The towns are fairly simple in nature. I ran out of time before I could customize each one or furnish the houses, so each town is an exact copy of the other in both structures [11] and townsfolk [9]. This is good in some ways as it allows familiarity for the player and it isn't necessary to go to any specific town, but it also loses some of the engagement since it is a bit odd. Towns have a different ground texture so no trees would spawn inside of it and there would be better control over the grass that appeared.

3.2.2 Enemy Campsites

Campsites were a reason to have enemies clumped together [9] [10]. The smoke offers an easy way to spot them in the distance so players can avoid them, or seek them out. Currently, they are the only place to find enemies in the main terrain (excluding the starting skelly), although original plans had more enemies spread out the world and campsites being special by containing a chest or two.

3.2.3 Abandoned Houses, Loot Caves, and the River

There are a few abandoned houses[11] and a couple of loot caves [4] in the world. They serve to add some structural diversity and as an easier way for players to earn coins or extra food without having to fight enemies. The river helps to separate the

world and contains the entrance to the water ruins. If the player enters the waters before obtaining the snorkel, they will lose health at a constant rate immediately.

3.2.4 The Graveyard

This location is just East of the center town and is the ending area of the game. A small puzzle leads the player to the correct grave by following the arrow-shaped headstones to the headstone with a cross inside a circle [9]. I realized after publishing the game that I didn't make it obvious what the ending headstone should look like, and that there were multiple interpretations of the 'second row, second column' hint the old woman gives. If the player selects the wrong grave that has the same shaped headstone, then a skeleton appears to attack them for 'disturbing their slumber'. When included, the correct grave will disappear upon selection to show a staircase leading to the final boss fight.

3.2.5 The Forest

Working on this area was a pain. At first, I wanted the fog to appear as it does now, but instead of following a path of signs the player would only get a hint as to where the ending house was, and wander the forest while fighting off enemies that were tall and skinny like the trees [9] so it wouldn't be obvious that they were there until last second. I did not have the skills to both design this enemy and animate it, so I instead opted for a more creepy puzzle-focused area. Following the signs is another idea from OOT and the reason why I included the knockoff lens of truth in the first place. In order to recognize if the player was on the path or not, the GameManager generates a linked list of all the signs in the order that the player should follow. If the player comes within 5 meters of the correct 'next' sign, it generates a large cube between that sign and the next. If the player ever leaves this cube, then the fog-out sequence will occur in which the fog gets thicker until the player can't see anything and is teleported outside the forest to try again.

```
public GameObject pathObjMaker(GameObject higher, GameObject lower)
{
    GameObject temp = GameObject.CreatePrimitive(PrimitiveType.Cube);
    Vector3 between = higher.transform.position - lower.transform.position;
    float dist = between.magnitude;
    Vector3 scale = temp.transform.localScale;
    scale.Set(15f, 15f, dist + 10f);
    temp.transform.localScale = scale;
    temp.transform.position = lower.transform.position + (between / 2.0f);
    temp.transform.LookAt(higher.transform.position);
    temp.GetComponent<MeshRenderer>().enabled = false;
    temp.GetComponent<BoxCollider>().isTrigger = true;
    temp.AddComponent<PathwayColliion>();
    temp.layer = 8;
    return temp;
}
1 reference
public void nodeChanger(GameObject collided)
{
    if (collided.gameObject.name.Equals("ArrowSign (40)"))
    {
        slowfog = false;
        StartCoroutine(SignGone());
    }
    if (curNode == null)
    {
        if (collided.Equals(signll.First.Value))
        {
            curNode = collided;
            nextNode = signll.Find(curNode).Next.Value;
            ctn = pathObjMaker(nextNode, curNode);
        }
    }
    else
    {
        if (collided.Equals(signll.Find(curNode).Next.Value) ||
            collided.Equals(signll.Find(curNode).Previous.Value))
        {
            prevNode = curNode;
            curNode = collided;
            if (ctp != null)
            {
                Destroy(ctp);
            }
            ctp = pathObjMaker(curNode, prevNode);
            if (signll.Find(curNode).Next != null)
            {
                nextNode = signll.Find(curNode).Next.Value;
                Destroy(ctn);
                ctn = pathObjMaker(nextNode, curNode);
            }
        }
    }
}
```

Figure 3.1: Code for when player collides with sign (nodeChanger) and helper function to generate cube between two signs (pathObjMaker)

3.3 Mountain Terrain

This is a rather simple terrain. It is slightly larger than the main terrain so that the very edges could be raised into mountains. The player is blocked off by invisible walls from exploring this area much at all. Its main purpose is to block the player's view from the endless void that surrounds them.

3.4 The Cavern

This section was modeled in Blender [3] and is one large gameobject that uses a mesh collider to prevent the player from falling through it. I was able to achieve the shape by extending the sides of a cube in lots of directions, flipping it inside out, and adding sub-dividers, smoother, and a dark brown material for color. I had started this section fairly early into the semester, so I was able to spend more time on it and fill it with different enemies and props. It is mostly flat, except for one section that raises in elevation. This was more of a test to see if that would translate well and if it would be able to still be walked on. While it was, it is also a lot steeper than expected and was a good lesson on expectations on what is built in Blender and how it translates into the game. Since this area is all about more difficult exploration, I wanted to include multiple forks in the path and have a bit more player decision involved since it wasn't easy to see the whole area at once, unlike the main terrain.

Figure 3.2: Topside view of the cavern

3.5 The Water Ruins

Ruins is a bit of an exaggeration, but this was honestly something I was not sure I would be able to get into the game. I had hit a lot of snags the last couple of weeks

and had started this a few days before turning in the project. Even though it was a rush job, I loved creating this area because of the design process that went into it. Whereas the cavern was following a tutorial and randomly making it as I went, I had to design this level around a puzzle. I found a lot of inspiration from OOT once more, as it has a water area in which you must raise and lower the water level and it is fairly infamous as the most difficult dungeon in the game due to the out-of-the-box thinking needed. I designed this area with a similar goal in mind with it seeming like a lot of possibilities, while truthfully only including a few until the player has collected all the runes. Of course, translating the puzzle from paper to in-game resulted in exploitations that I did not think of, which is why it was necessary to make it possible to take runes out underwater or while dry, but only put them in when the player was above water. Additionally, specific design choices resulted in exploitation such as the speed of the water level changing. I wanted it to be fast enough the player wouldn't have to wait around forever, but slow enough so that there would be a bit of waiting to think about the next steps. After submitting the project and before writing this, I had someone play the game and they had the idea of swimming up the water as it was lowering, disrupting the order of collecting the runes and making it possible to beat the level without collecting the '-1' triangle rune. This area was likely the greatest lesson in game design and how there will always be someone who tries things in ways I would not think of.

Chapter 4

Player

The player moves and interacts throughout the world using the player character (PC) [8] and different keyboard and mouse inputs. The PC consists of multiple components, scripts, and child gameobjects. Its children are in control of displaying the model of the character, containing its animator controller, and including the camera that allows the player to see. The PC has a collider, character controller, movement script, and a script for controlling health and hunger.

4.1 Movement

The movement script is a modified version of a first-person controller script [6] that allows for jumping, running, and gravity effects. It controls movement by using the character controller component to move in the direction of the current rotation using speed specified by public variables. A coroutine in the GameManager checks every 0.2 seconds if the player is underwater by casting a ray from the player's waist upwards, checking for objects of a specified layer designated for water. If this ray hits, then the player is considered underwater and movement is changed to allow only forward movement in roughly the direction of the camera's rotation (determined by the mouse's axis data in both the X and Y direction).

4.1.1 Animations

The animations [8] of the player are controlled separately from the movement script. I made this decision at the beginning of the project and have regretted it since, but never got around to compiling everything into one script. It gathers data from multiple scripts every update to determine the state of the player and has the animator controller play the correct animation. Working through the logic of animations was a difficult part of this project as many factors came into play that I did not expect, and the system can become confusing to navigate with too many transitions.

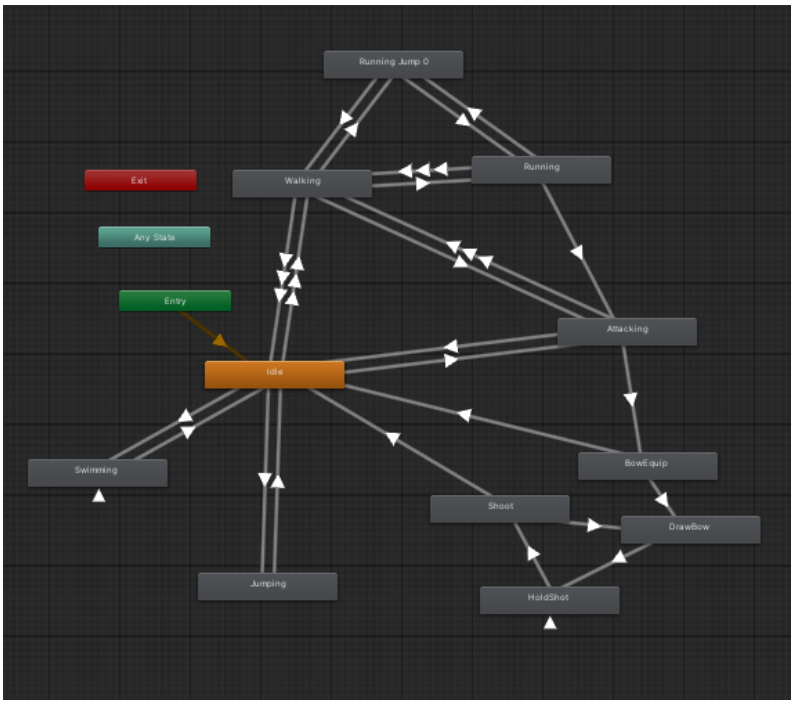

Figure 4.1: Animator UI showing different states and transitions for the PC

4.2 Interactions and Scripting

The player affects their PC's interactions with the world in two main ways (barring movement itself): interactions using the 'e' key, and items held.

4.2.1 'e' Key Interaction

Input using the 'e' key is almost always handled by other objects throughout the world having trigger colliders that check if the PC enters their space. If so, and the player presses the 'e' key, then a thing happens. Common occurrences of this are when the player picks up items like rocks or sticks, or when they interact with chests to open them and collect the item. The other way it is handled is within the GameManager during its update function. Due to issues using the Vegetation Spawner asset, trigger colliders attached to tree objects would no longer become triggers and would physically interact with the player and prevent the player from collecting fruit. To get around this, the GameManager, upon 'e' input, finds the positional data of each tree, calculates its distance from the player, and if it is less than 3 meters adds fruit to their inventory and deletes the tree. This initially caused some performance issues with spammed 'e' inputs, but I was able to optimize it to where it no longer had a problem.

4.2.2 Items Held

The most common type of held item is the weapon. Upon use, the correct animation(s) play and trigger colliders on the weapon collide with an object with a tag of an enemy, which calls the correct damage function for that enemy. Two other items exist though to interact with the world: the lantern and the KLoT. The lantern has a child gameobject with a point light component allowing it to accurately cast light from within itself. The KLoT, when toggled on, will call a function to turn certain gameobjects on or off, depending on what is necessary. It will also affect the post-processing color to give a slight red hue.

4.3 Canvases

The player can control multiple canvases, and interact with the world to have others show up. The most-seen canvas only contains images that display the current health and hunger of the player. The player has control over toggling other canvases, such as their inventory with 'q' or the pause menu with 'escape'. Interacting with special

NPCs like the old woman or merchant has the dialogue canvas pop up, then the store canvas if the player talks to the merchant.

Figure 4.2: Well progressed player's inventory

Chapter 5

Non-Player Characters and AI

5.1 Overview

All NPCs consist of the same base code that controls their movement and use Unity's built-in NavMeshAgent (NMA) component to help direct their movement. For this game, the Navigation in Unity bakes objects as either walkable, obstacle, or undefined. NMAs would then stay on walkable surfaces and direct their paths towards their destination using A* search avoiding obstacles. Objects baked as undefined had no influence. The NPCs have two movement modes, random walk and directed movement. The random movement consists of picking a random position within a certain sized radius (typically 20 meters) from their starting position and, if the movement is possible, set their NMA destination to that position and idle for a few seconds before repeating. Directed movement for enemies would trigger once they became 'angry' and would cause their NMA's destination to be a distance away from the player optimum for the enemies' attack strategy. Although the directed movement was not fully incorporated for townsfolk, it would trigger when the player antagonized them and would cause them to become scared and run away from the player.

5.2 Enemies

There are four types of enemies within the game, two in the fields and two in the cavern. Enemies consist of three states: walking, chilling, and attacking. They cycle through walking and chilling until they are angry, which can come from being

damaged, seeing the player in a field of view, or (in the gronchs case) being alerted. The field of view works by collecting a list of colliders in the "player" layer within a specified radius of the enemy. Since there is only one gameobject in the game using this layer, the player, it means they are within the radius. It then calculates the direction of the player and if they are within the section of the circle considered the enemy's eyesight, casts a ray in the direction of the player. If it does not collide with anything of the "obstruction" layer then the player is considered seen. Enemies have specified health and damage values, and upon death, their gameobject is destroyed and a random range of coins is dropped in place depending on enemy difficulty.

5.2.1 Skelly

This is the most basic type of enemy. There is one just a bit ahead of the player when they start the game as sort of practice on fighting, and can also be found in campsites around the fields. They are a skeleton [10] equipped with a sword and shield (although they don't use the shield). If the player runs outside of their sight range, they will go to the last place they saw the player.

5.2.2 Plague Doctor

These are the only ranged enemy in the game so far [9]. Upon seeing the player, they will move into range for their attack and start casting a spell. The spell takes a few seconds to charge up but upon expulsion, it will follow the player until it disappears after a few seconds. If the player moves out of range of the plague doctor then they will resume their pursuit until they can no longer see the player. This enemy is also found in campsites around the fields.

5.2.3 Earth Elemental

There is only one of these in the game, and it can be called the 'boss' of the caverns although fighting it is completely optional [5]. It floats around and upon seeing the player, will punch at it with shocking speed and follow after them. A good strategy of escape is to run away as it is generally too large to traverse the cavern tunnels

quickly. It is also difficult to shoot the earth elemental with the bow, as the player needs both hands and is unable to have the lantern equipped.

5.2.4 Gronch

This is my favorite enemy and the first one I developed [7]. These are somewhat peaceful enemies who survive underground on the glowing yellow rocks seen throughout the cavern. They are the only enemy to not use the field of view system and while they won't fight the player initially, if they are hurt by the player's weapons they will alert all other gronchs in a certain radius of danger and they will all go after the player in anger. They strike with their fists, but move slowly and are easy to escape from.

Figure 5.1: Enemies (Top left: Skelly, bottom right: plague doctor, top right: earth elemental, bottom right: Gronch)

5.3 Townsfolk

These peaceful entities exist in towns throughout the world [9]. There are five kinds, but only two are useful to the player: the merchant and the old woman. They all consist of three states: walking and chilling like the enemies do, but also talking. Talking can only occur with the merchant and the old woman. Talking with the merchant will bring up a small line of dialogue followed by the store canvas where the player has access to purchasing items. Talking to the old woman will bring up expositional text that is dependent on the amount of progress made, typically incremented by special items collected or areas visited.

5.4 Animations

Most NPC animations are directly correlated to the state they find themselves in [2]. Chilling causes an idle animation, walking causes a walking one and attacking calls for the attacking animations. There is preliminary work for both damaged and dead states so that the game can feel a bit better, but I did not have time to incorporate these as well.

Chapter 6

Conclusions

If I had to summarize this project in one sentence it would be "I don't know why this doesn't work, time to try a different way". So often I had a goal in mind that I would try to accomplish in different ways before finally finding a solution that was nowhere near my original idea but I had been led to by previous failures. Working on this game has been the most difficult and most fun project I have had in my academic career. It was really tough finding the motivation to prioritize this project in the first half of the semester since it was easy to rationalize that this was due in over a month, while there was always something else due in a few days. Towards the end of the semester (and especially the days leading up to presentation day) I would work on this from the moment I woke up to the moment I went to sleep only stopping occasionally for small breaks and to eat; partially from trying to get everything in the game, but mostly from having the time I didn't previously and enjoying the learning process and finding out all sorts of new things about Unity [16], Blender [1], and game design as a whole.

6.1 Lies, Lies, and More Lies

Game design in a nutshell. More often than I would like to admit I would struggle with some aspect or idea that seemingly had no solution or scripting method possible to change what I wanted, and then I would figure out I was over-complicating the process and I could easily achieve my goal in a very simple manner. My favorite instance of this that I love to show to people is that when developing the fog for the

forest area, the fog only affects what is in the game world, so the skybox would look the exact same when looking straight up. That isn't how fog works. After spending a while trying to figure out different solutions or get specially-made fog skyboxes, I put a big inverted gray cube around the PC camera. Anything far enough from the player looked like the fog had accumulated too much to see clearly but in reality, they were just looking at walls. Another good one is how the player starts the game with all items equipped, but their gameobjects did not become active until certain conditions are met so it is as if they aren't there at all.

6.2 Unity Tips and Tricks

When this project started, I had used coroutines just enough before to know that they existed. By the end of it, I had used coroutines for some of the smallest things because of how much I loved them. I found them most useful for a custom update function for less important checks, such as checking if the player is underwater or in fog. It isn't necessary to check for that every single frame, because a few times per second is just as perceptible to the player and can save lots of processing power. Additionally, I had never really touched the physics system or built-in functions of Unity, but they are extremely useful. This includes raycasting, collider interactions, and layers. It was both easier to code, and easier on processing, to be able to disable collisions between objects of certain layers (such as player and water collisions so that raycasting still worked but the player could not walk on water) or specifically look for specified-layer object colliders within a certain radius of an enemy (useful for field of view detection).

6.3 Future Work

While I got nowhere near what I had planned, the amount of work that I had put into this project was so much more than I thought I would do. I loved working on this and will keep working on it when I can. First will be fixing the many bugs and issues that I have become aware of as others play my game, then will be including the actual ending to the project. As far as the scope of the story or gameplay of the

project is concerned, I don't imagine I would change much about it. I'm happy with what has been included and how long the gameplay is. The ultimate goal would be to continue my process of learning more about the aesthetics of game development and piece by piece transition each outsourced asset with something of my own creation until it is something that I can truly say was built by myself.

Bibliography

[1] Blender. https://www.blender.org/. Accessed: 2022-12-12.

[2] Blink Studio Animations. https://assetstore.unity.com/packages/3d/animations/free-32-rpg-animations-215058. Accessed: 2022-10-25.

[3] Cave in Blender Tutorial. https://www.youtube.com/watch?v=SrqGlA9mpYw&list=LL&index=2&t=168s. Accessed: 2022-10-20.

[4] Caves. https://assetstore.unity.com/publishers/11721. Accessed: 2022-09-26.

[5] Earth Elemental Model and Animations. https://assetstore.unity.com/packages/3d/characters/creatures/elemental-animations-free-227604. Accessed: 2022-11-10.

[6] First-Person Controller. https://sharpcoderblog.com/blog/unity-3d-fps-controller. Accessed: 2022-08-28.

[7] Gronch Model and Animations. https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/mini-legion-rock-golem-pbr-hp-polyart-94707. Accessed: 2022-11-10.

[8] Mixamo 3D Models and Animations. https://www.mixamo.com/#/. Accessed: 2022-08-28.

[9] Polytope Studio. https://assetstore.unity.com/publishers/35251. Accessed: 2022-09-09.

[10] Skeleton Model and Animations . https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/lowpoly-medieval-skeleton-free-pack-181883. Accessed: 2022-11-26.

[11] Slavic Town. https://assetstore.unity.com/packages/3d/environments/fantasy/free-slavic-medieval-environment-town-interior-and-exterior-167010. Accessed: 2022-09-25.

[12] Terrain in Unity Tutorial. https://www.youtube.com/watch?v=2XdQkwSw-bA&list=LL&index=8&t=257s. Accessed: 2022-09-04.

[13] Terrain Textures. https://assetstore.unity.com/packages/3d/environments/landscapes/terrain-sample-asset-pack-145808. Accessed: 2022-09-09.

[14] *A Dark Room*. https://en.wikipedia.org/wiki/A_Dark_Room. Accessed: 2022-12-12.

[15] *The Legend of Zelda: Ocarina of Time*. https://en.wikipedia.org/wiki/The_Legend_of_Zelda:_Ocarina_of_Time. Accessed: 2022-12-12.

[16] Unity. https://unity.com/. Accessed: 2022-12-12.

[17] Unity Asset Store. https://assetstore.unity.com/. Accessed: 2022-09-04.

[18] Vegetation Spawner. https://assetstore.unity.com/packages/tools/terrain/vegetation-spawner-177192. Accessed: 2022-09-25.