Assignment 5 – Process Work

Pre-production:

- Before starting I brainstormed some ideas and decided on making a Metroidvania style game.
- I looked for free assets to get some ideas for possible themes for the game, but nothing stood out.
- I decided that if I am going to make my own assets, I am going to pick a theme that I think could be good, that is not terribly overdone.
    - Decided on Eldritch Horror for the art style.

Reference art:

Notes:

- Art can have a black background with a muted red, blue, or green foreground.
- Eyes and tendrils are important.
- Should feel like another planet.
- Usually starts normal then gets weird.

I was having trouble with unity license stuff, so I reinstalled Unity hub and it fixed the issue. I then decided to start by making some assets for the game. I could have used default texture stuff, but I enjoy planning with the actual sprites I will be using.

I created a 9-slice asset for the walls that looks pretty good, as long as it is a size where some of the rocks are not cut off. I used colour theory and a palette that gives off a sort of horror vibe. (i.e., dark red). I made sure to set its collision to static so that it will not move once placed. I then placed it in the world to test.

Next, I created the player sprite. I first created the concept without animation as a png object with few layers. After talking with Tom about best practices, I realized that having each limb on a different layer and separating them onto a sprite sheet would allow for the best animation given the time restraints. For now, I will use the non-animating picture to test the main game mechanics and to make sure everything is working. If I have time, I will get it to animate and draw different sprite behaviors. If I do not, I will look for free assets online.

I then started to place some of the objects in the scene. I made the walls scale based on their size and tile properly so that they can be any size. I also made the bounding box for collision scale with the size. I added the player and although it obeyed gravity, it did not move when the player pressed a key. To solve this, I created a player movement script that handled the input. I expected that I would need this, and although I could have gotten a package from the unity store, I decided to write my own, which was good practice for learning how the unity engine works. What I did not expect, was when I moved the player, it would rotate at a crazy angle and flip out on the screen. After some trial-and-error messing with the bounding boxes, I saw in the documentation that there is a freeze rotation Boolean property that can be set in the code. After setting this to true, the player moved as expected.

I then added more walls and started building out the level to test more features and make sure the numbers for speed and jumper were all correct. After playtesting and iterating, the only remaining bug is that the player will sometimes stall when they collide with a wall and continue to hold that direction's key. This could probably be solved by setting the X velocity to 0 when the player collides, but I may add wall jumping later if I have time, so I will leave it until then, so I don't have to rewrite code.

After the player, I decided to add some enemies to the game. I created a quick test sprite and added it to the game. I then created a script to move it left and right. I played around with the numbers until it felt right. I also froze the rotation and made the weight very high so that the player could not push them around.

With that working I decided to add another enemy that follows the player within a certain range. To do this, I created a script that can be used in multiple places in the future that allows an object to detect another object within its range, with a specific tag. I decided to separate the detection script and

the go to target script. This will allow for different things to happen when something is detected. I thought about having go to target not need a detection script, and I may still implement it if I have other means of determining the target in the future. For now, the go to target script requires a detection script, and will automatically add it to any object that uses the former. I tested the speed and range values until I was satisfied with it working.

At this stage, I separated the remaining features of the game into two categories, Need to have and Nice to have.

Need to have:

- Player health
- Enemy health
- Player attacking
- Damage and collision
- Level design
- Goal
- Animations

Nice to have:

- Custom Animations
- More enemies
- Powerups and abilities
    - Wall jump
    - Double jump
    - Attack types
- Story
- Title page / instructions
- Multiple levels
- Moving platforms and other gimmicks
- Other stuff

Taking this into consideration, I decided to work on health next. In my research into unity, I learned about scriptable objects, and the person using them used Player Health as an example. This has advantages over static variables and creating variables within a script. I created a slider to control the health and tied that to the Player health scriptable object. I added a script to do contact damage to the player when they collide with an enemy, and it works! I also needed to create a maximum health scriptable value for the player so that it resets each time the game is played, instead of saving the previous value. This gave me an idea to use scriptable components to create a rudimentary save system in the game. I may do this later if there is time.

Next for the player attacking, my first idea is to have create a new temporary object when the user presses a button, that moves in a specific pattern, and has a hit box that can collide with the enemies. I know that I want it to be a melee weapon because I want the tentacle attached to the player to be the main weapon. Additionally, I like the physics of batting away the flying enemies and having

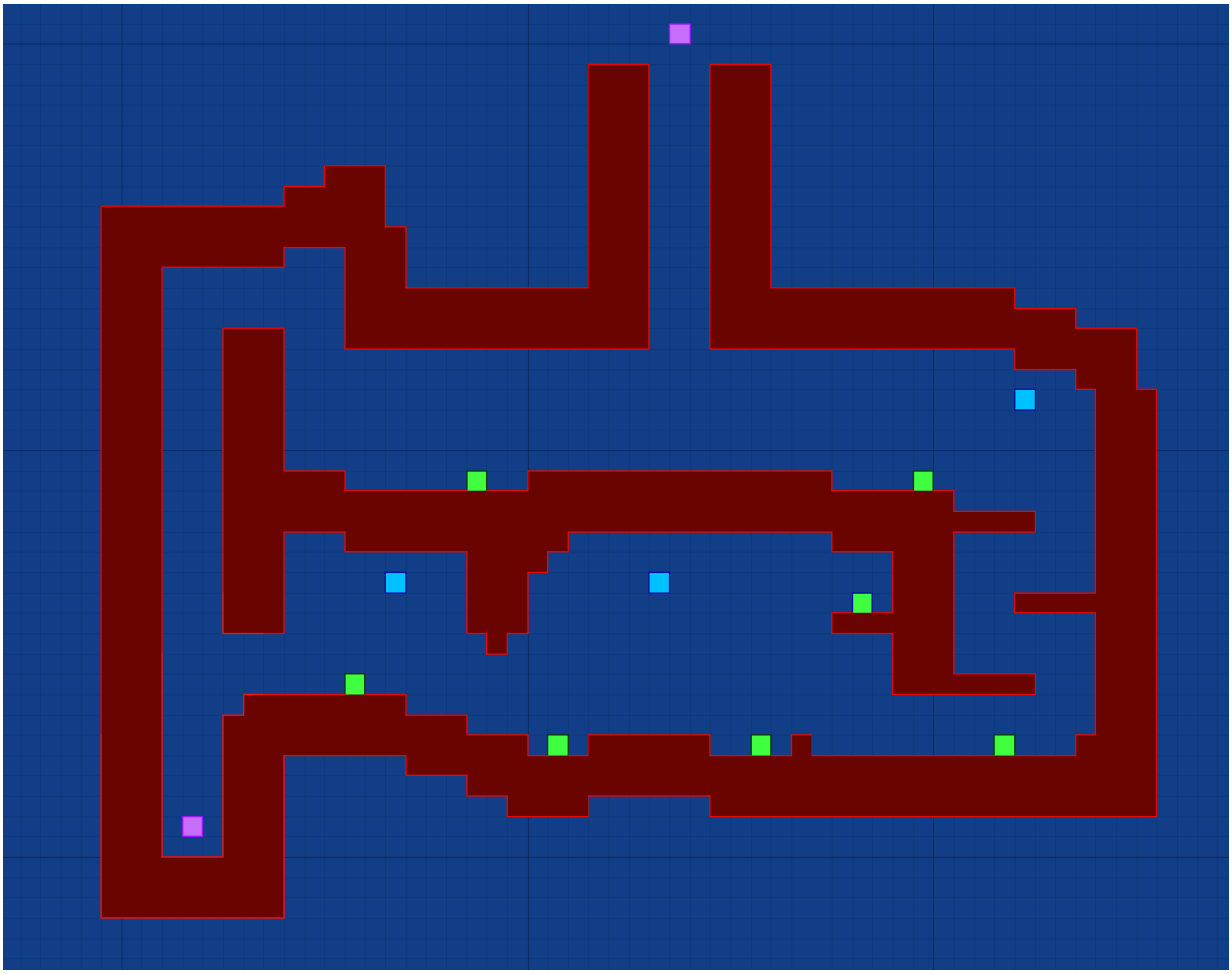
them bounce off the walls as they are pushed away from the player. It also it should be a prefab, and generic enough that the enemies can use it as well to do their attack animation.

Weapon script:

- Public variables:
    - Damage
    - Attack Speed
    - Range
    - Type
        - Thrust
        - Slash
        - Spin
        - Projectile (Might be a separate class)
    - Animation Component (If applicable when doing animation stuff)
- Private variables:
    - Collision box
    - Time attacking
    - Done attacking
    - Cooldown
- Subroutines:
    - Start
    - Update
    - On Attack
    - Create weapon object
    - Remove weapon object
    - On Collision Enter
        - Use other object type to determine effect.
    - Play animation (If applicable)

After planning out this script, I decided that Weapon should be more generic, so I will call it Attack and attach it to all objects that can attack. The weapon object itself should just be a prefab with a collision box and public radius and animation component. I realized as I was making the class that On Collision enter is for the weapon object, not the attack script, so I added it to the prefab. I moved a lot of the code into the specific attack type prefab, so I will have a different prefab that is similar for each attack type that will be attached to the game object that is attacking with the attack script. I had trouble getting the attack to move with the player, which I fixed by making it kinematic instead of using physics. I added health bars to the enemies and got the collision working using variables.

Next, I wanted to plan out my game's level using software called Grid Cartographer. It allows is basically digital grid paper with more functionality. The red squares are walls, the green and blue squares are enemies, and the purple is the goal and powerup.



I designed it like this to showcase the different elements of the game including the enemies and the physics. The powerup would flip gravity if I have time to implement it.

I will likely continue to work on this project after the deadline, and I plan to improve the art assets to improve my art ability over the break. Thanks for the great semester!