Adelynn Klemme

Professor Ellertson

GIMM 110

24 October 2022

Froggy Adventures Rhetorical Analysis

When it comes to individuality and how I show my efforts in the GIMM program, I believe that my individual game is a great artifact to display my knowledge as I grew through educational opportunities. Froggy Adventures has multiple design choices and coding content that I believe pushed my limits as a Freshman with gaming.

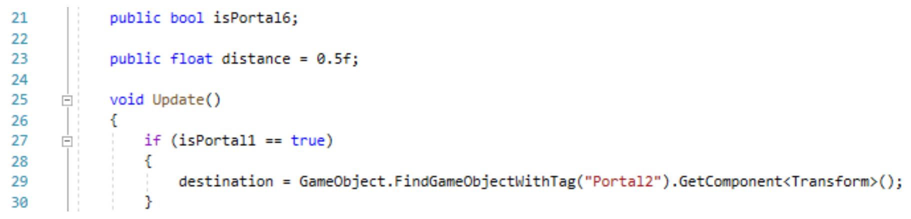
To begin with the first main focus of my game, there are three topics I wanted to focus on with specific design choices, those being color, art style, and the theme of the game. When it came to the initial planning of what I wanted my game to be, I knew I wanted to start my game off with bright game objects such as a cobalt blue or lime green shade. This is because I didn't want my game to look dreary or have objects to fade into one another due to a lack of color. I believe this stems from my own personality as I love things that stick out, like my big earring collection that consists of wild things such as fortune cookies and dinosaurs. By selecting bright colors to be the main focus, I am putting a little bit of myself into the game to make it unique.

As for the art style, I stuck with a 2D pixelated art style because I wanted my game to feel like a classic arcade game. I have an attachment to games with this style such as Minecraft, Blockheads, and Tiny Tower because I grew up with them as a kid since technology was still fairly new. I wanted a piece of my childhood to reflect in my game which is why I took inspiration from these games to have a 2D pixelated game. My game objects vary in the number of pixels I use for each object, sizes from 16 pixels to 160 pixels in width and height, to be able to create different size objects and design effects. This style also reflects in my introduction animation video due to the fact that I wanted it to match my game so I made each new frame feel blocky and abrupt like pixels in animation.

In regards to the last design element, I wanted my theme to take the audience outside of themselves which is why I chose to have my main character be an animal. In particular, I chose a frog as I believed I could illustrate the mechanics of a frog with animation better based on the understanding of frogs motion I have of them. This is due to my Art 112 class at the University of Idaho when I did research into all kinds of frogs to depict them in a comic strip art assignment. Due to this background knowledge, I believe this gave me leverage in my game to enhance the visual understanding from the audience. From here I built the surrounding game environment off of aspects frogs are associated with. First, I implemented 3 levels in which the frog climbs, starting underground and ending in the forest trees as they tend to be found in such places. Second, I added in a teleportation portal as it is common for frogs to disappear from the surface of water quickly and reemerge across the pond. Next, I added a little leaping pad to give the player a higher jump when needed due to the fact that frogs have strong hind legs. Finally, I wanted the trap in my game to reflect a poison like puddle to relate to the handful of frogs that are poisonous to the touch. Game choices and aspects such as those listed above influenced the specific game play and physics solves I chose to implement later in my game as well.

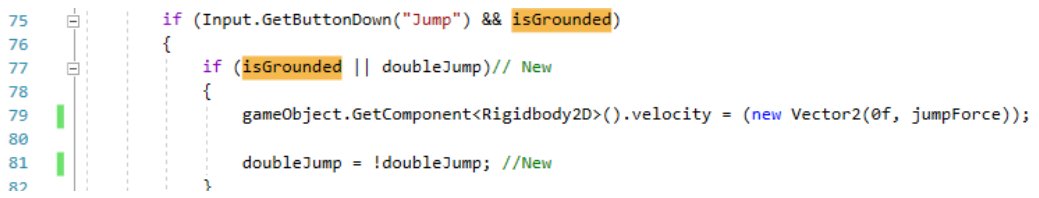
Shifting to the second overall focus of my game, the content of my game consisted of many physics solves such as teleportation, a jumping pad, double jumping, left shift sprinting, player respawn, and camera movement. To add such physics solves into my game, I used both tutors from the internet and provided in class coding examples to manipulate my game objects. For the sake of the length of the analysis, I will only be discussing the four difficult coding issues

I ran into and how I solved them. Starting with the first solve, teleportation was one of the challenging physics changes to implement into my game. I did as such using the code provided below, found in my “Teleportation” script. I built a total of 6 portals by creating public booleans to access the portal in the Update method. By using if-else statements to look if a desired portal is being collided with, I was able to set the teleport destination to a second portal on the same level. However, I did run into an issue of continuous teleportation of the frog which I solved by adding a float of one in the positive x direction and a float of two in the negative y direction at the second destination to get the character out of the portal’s box collider.



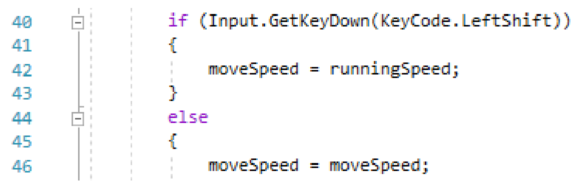
```
public bool isPortal6;

public float distance = 0.5f;

void Update()
{
    if (isPortal1 == true)
    {
        destination = GameObject.FindGameObjectWithTag("Portal2").GetComponent<Transform>();
    }
```

For my second physics change, I struggled with getting my character to double jump at any point as the character’s jump would cut out in the middle of action. I solved this issue by implementing the code to the left, also found in my “Movement2D” script. By detecting if the player is grounded or if the w key is pressed, I allow the player to then jump and decide if it is allowed to jump again based on if it is grounded or double jumping. I allotted a double jump by setting a public jumping force variable and implementing the variable in the y spot of the vector’s velocity within the jump method, with the jump method called in the Update method.



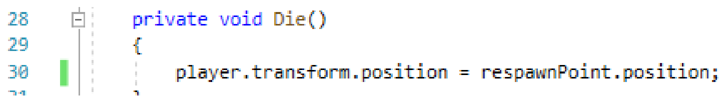
```
if (Input.GetButtonDown("Jump") && isGrounded)
{
    if (isGrounded || doubleJump)// New
    {
        gameObject.GetComponent<Rigidbody2D>().velocity = (new Vector2(0f, jumpForce));

        doubleJump = !doubleJump; //New
    }
```

In the next challenge I faced, I struggled with using the left shift key to make the frog run faster when pushed with the d key. This code is located in my “Movement2D” script with the main components pictured on the below. By using if statements to monitor the computer's keyboard, I was able to set the “moveSpeed” to the “runningSpeed”, which is a float of 20 (double the move speed), when I pressed the left shift key. Otherwise, I had to reset the movespeed back to itself using an else statement. The issue I ran into with this particular physics solve was that I could only sprint in the positive x direction. This issue was solved when I put the sprinting code in my Update method rather than in its own method.



```
if (Input.GetKeyDown(KeyCode.LeftShift))
{
    moveSpeed = runningSpeed;
}
else
{
    moveSpeed = moveSpeed;
```

My last problematic physics solve was my player respawn code. Before I switched my player to respawn at the start when dying, I initially had my character reload the entire level. This caused an issue with the timer and mushroom count that keeps track of the players score. I resolved such an issue by removing the level reload and replacing it with a respawn point. The code I used to solve this issue is pictured to the left and it is found in my “PlayerLife” script. To get my character to respawn, I had to have a publicly accessible variable named “player” change the x and y position inside the game to a dedicated respawn point I set using a different game object. I set this transformation in a private “Die” method which is called when the player collides with an object that has the tag “Trap”.Although these examples are only a portion of the code, I believe that these examples required the most research and failures to get my game to where it is now.



```
private void Die()
{
    player.transform.position = respawnPoint.position;
```

As I learned how to fail throughout my game, it brought me to the conclusion that I learn the most about my game and code when the game had compiler errors. It forced me to learn the coding environment I constructed and how each line of code functioned as a whole. Overall, I found this assignment to be very insightful, educational, and fun as I chose the design elements and game components I wanted in my game.