Final Project Documentation

SUNY Morrisville

CITA 312 – Intermediate Game Design

# Introduction

This document will store my updates on my final project for CITA 312. I will store my thought process behind each GitHub commit I make for my game and any logic for the code that goes into making the game.

Link to GitHub: <https://github.com/JeremyRoalef/CITA_312_Final_Project>

# Commit “Add Third-Person Camera Perspective”:

In this commit, I followed a simple YouTube guide on how to set up a third-person camera perspective. There were other concepts of player movement in that video relative to the camera perspective, but I chose to ignore that part because I want to try and code it myself.

I added a third-person camera perspective by installing the cinemachine package from the Unity registry, creating a FreeLook Camera, and messing with some of the attributes that come with it.

Some essential attributes were the Mouse X speed and Mouse Y speed (two attributes that may be changed in the future to add mouse sensitivity settings to the game), the “Orbits” section for the top, middle, and bottom rig camera position, setting the binding mode to world space instead of the default option, and adding an extension for cinemachine collider (script).

In the cinemachine collider (script), I added some collision layers to the camera and ignored the objects tagged “Player.” Finally, I changed the “strategy" to “Pull Camera Forward” so collisions will move the camera closer to the player when close to an object.

Link to YouTube video: <https://www.youtube.com/watch?v=UCwwn2q4Vys>

I do not mean to follow tutorials for creating this video game, but I needed guidance to set up a third-person camera. After an honest attempt at making my script for a third-person camera (that failed miserably), I felt I was missing something obvious and was right. This will be one of the only times I will seek help creating this game. Rotations are just complex to work with.

# Commit “Add Movement relative to the camera’s direction”:

I added the input system package in the Unity registry for this commit. I am very comfortable using the built-in input system, but I feel I should challenge myself by working with a different system than I’m comfortable with.

The logic for the player movement is surprisingly complicated. I want to move the player relative to the camera’s forward and right direction. Thus, I cashed a reference to the camera object, stored the camera’s forward and right vectors in variables, and normalized their magnitude. From there, it was a matter of finding the correct movement vector and setting the rigidbody’s velocity to the proper vector.

The forward and right vectors are derivative of the camera’s position and rotation. However, these vectors are in 3D, and I need to work in a 2D system for player input. Thankfully, this was easily handled because I could ignore the y coordinate in the forward and right vectors.

The real challenge was getting the correct velocity. My logic to get this was to create a Vector3 variable and add the forward and side movement values to the Vector3’s values. This somehow worked, and I’m not entirely sure how.

I have no clue how I managed to get the code to work. I threw random ideas at the code until one solution worked. I drew a small visualization of the directions, which helped me better find the current solution to the movement.

One change I am thinking about is changing the method of movement from setting the velocity to adding a force. When I set up the jump mechanic, I will be using force for smooth jumping, so it is only natural to be consistent with the type of movement I am doing for all inputs.

# Commit “Add Jump Mechanic”:

Adding the jump mechanic was straightforward. It is as simple as applying a force in the world’s upward direction. The only challenging part was setting the logic where the player can jump.

The player can jump when they are colliding with the ground. I used the OnCollisionEnter and OnCollisionStay events to get this to work. I set up a switch case if I add more collision logic based on game object tags in the future.

I found a funny bug where the player can attach to walls and not fall to the ground even though they should be falling. There is some friction between the objects that is way too high for the player to fall. I will have to find out how to change the friction between these objects.

I need to update the physics settings and increase the gravity factor. The player takes too long to fall to the ground, and the jumping does not feel smooth as a result.

# Commit “Fix Sticky Walls || Update Movement Script || Update Gravity”:

For this commit, I fixed the bad feeling of the jump by updating the project settings to make the gravity stronger. Now, the jump feels much more lively. I also updated the player movement from applying a velocity to applying a force and also clamped the minimum and maximum velocity so the player cannot zoom across the map. I used the physics materials to apply frictions to the ground and the walls. This removed the bug from before where the player was sticking to the wall, and the ground friction lets me control the player’s speed based on another condition.

# Commit “Update Clunky Camera Movement”:

There was some dampening effects happening with the camera position, causing the player movement to act strangely and having weird things happen with the camera itself.

# Commit “Set Up Basic Platform Movement”:

I added a platform to the game. The platform moves by looking at its given platform path object and using linear interpolation to go from one position to another. However, the current Lerp needs to allow the platform to move back to its original position. I also need to change how the travel percentage is calculated to get constant velocity through the path. The current method gets the platform from point a to b, b to c, and c to d in x amount of time, meaning the velocity of the object is changing each time. I want the platform to from point a to b, b to c, and c to d at x velocity.

# Commit “Add Forward/Backward Platform Movement & Constant Velocity || Add Delay Variables”:

I have found a formula to get constant velocity. The formula is as follows:

Changing the formula to solve for the total time I want the platform to move from point a to b at a constant velocity given the distance between points a and b allows me to calculate the percentage of travel not based off of elapsed time alone, but the total elapsed time divided by the time calculated. In other words, the velocity stays the same but the time changes based on the distance between the two path points.

I added some delay variables to control the initial delay of the platform movement and the delay between moving from each point in the path. However, there are new bugs where the platform will do its initial delay and then the delay between paths at the same time.

# Commit “Update Coroutine Delays”:

Fixed the coroutine delays such that the first path has no delay due to the initial delay, but the rest have the delay between paths. Also, changed the behavior from a foreach loop to a for loop so I can better control the conditions. I used a Boolean to change the behavior of the platform from moving forward to moving backward. However, this code is repetitive and could probably be combines into a single iterator instead of using an if-else statement. I might not bother with that because the platform does not apply a velocity to the player when the player jumps onto it, causing the player to fall off the platform despite not moving. I will have to consider if there’s a way to move the player without changing my code, but if worse comes to worse I will apply a rigidbody to the platform, disable gravity, use velocity vectors to move the platform, and set the velocity instead of using linear interpolation.