Assignment 1

Unreal

Jesse Wood
300312194

Learning to develop in a game engine

COMP 309
Computer Game Development
Engineering & Computer Science
August 6, 2020

1 Game Description

1.1 Main Action

My game is *Jump*. A third-person jumping puzzle adventure. It was created using the Unreal Engine (Games 2020b). The source code for the project written in C++ is available on a Github (Wood 2020b). A demo video of the game that features gameplay and extended commentary on the making of the game is available on YouTube (Wood 2020a).

One may deduce from the title, that the main action of the game is to Jump. A player must jump from obstacle to obstacle to reach the highest point of the level. The player's ability to jump is limited by there energy. Each jump takes away $\frac{1}{4}$ of their energy. Once they have no more energy they are unable to jump. Therefore they must restart the level. However, scattered throughout the level are collectables. These collectables restore the player's energy.

1.2 Components

The game has three levels. The *Main*, *Jump*, and *Final*.

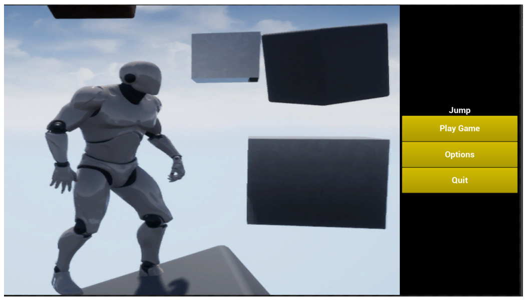

Figure 1: Main Menu Screen: The start of the game

The first level represents the Main Menu of the game. It generates a widget, where the player can use their mouse to navigate the menu. The player can start the game, or exit the application entirely. This menu provides options, where the player can adjust their desired screen resolution.

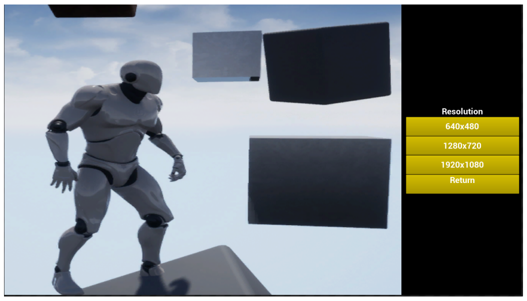

Figure 2: Main Menu Settings: Resolution options

The *Jump* level is the main action of the game (as discussed earlier).

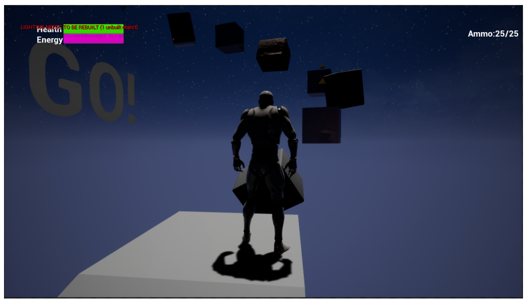

Figure 3: Jump Screen: The player is in a game

This level utilizes a variety of C++ blueprints. Including but not limited to *FloatingActor* (Games 2020a), *Travel Cube*, *EscalatorCube*, *ACollectible*, *Countdown*. The first three actors are the components that make up the level. The first offers a dynamic cube the floats up and down while spinning. The travel cube moves along the X axis, oscillating back and forth between two locations. The EscalatorCube provides similar functionality but along the Z axis. The countdown provides a dynamic countdown, where the user can set the desired duration, and finished on a "GO!" string whilst spawning an explosion emitter.

Perhaps one of the most pivotal C++ classes of the main action is the *ACollectible*. This class extends the *Actor* class. This class utilizes the *RotatingMovementComponent*. This allows the collectables to rapidly spin. The main functionality of this class is provided through the overridden *Overlap* method from *Actor*. From this class, we set the actor to be *destroyed* when the cube overlaps with another actor. This gives us the appearance that the actor has picked up a collectable when they interact.

```
void AACollectible::Overlap(
   AActor* OverlappedActor,
   AActor* OtherActor
)
{
            Destroy();
}
```

We then extend the *Third_Person_Character* class to play a sound when this actor overlaps with an instance of our *ACollectible* actor.

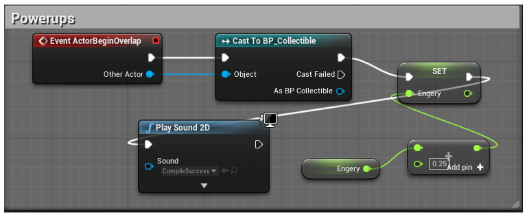

Figure 4: Third Person Character Blueprint: Powerup functionality

1.3 Hardest Part

The hardest part of the game to implement was Respawn functionality. This involved extending the *Third_Person_Character* blueprint to handle the destruction of its actor. When the player died, they were taken to the following screen.

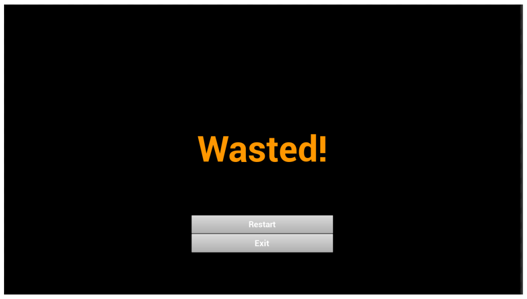

Figure 5: Respawn Screen: When the player has died!

However, when the game progressed to a new level, that actor was naturally destroyed to create the new level. The *Final* level was where the ending screen below was displayed, this was shown once the player had completed the game.

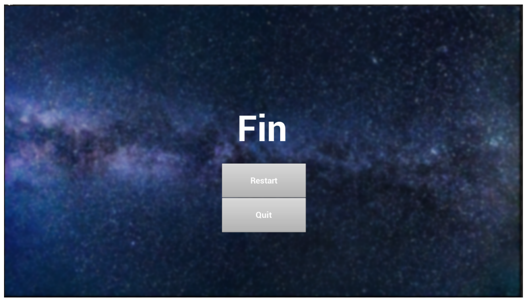

Figure 6: Final Screen: After the player has finished the game

So despite the intention of leaving to the *Main* level for the Main Menu, or the *Final* level for the end of the level screen, the *Third_Person_Character* blueprint would override these redirects and immediately take to the *GameOver* level.

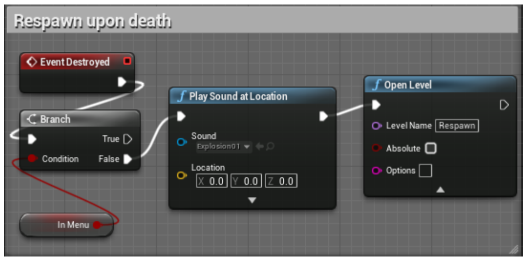

Figure 7: Third Person Character: Respawn functionality

The solution to this was to add a variable to the *Third_Person_Character* blueprint called *inMenu*. So before we restart the level because the player actor has been destroyed, we check to see if the player was in a Menu. This allows us to destroy the actor safely, without unintended side effects.

1.4 Most Interesting

The most interesting part of the game is the jump mechanic. The player's energy level determines whether or not they can jump.

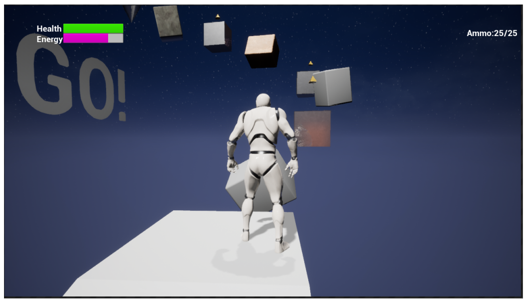

Figure 8: Energy UI: See the top left corner after a jump

This adds strategy to the jumping puzzle. Rather than just a journey from A to Z, we add an economy system to the game. The player must consider the path carefully, to maximise the efficiency of their path, and try to visit as many collectables as possible. This borrows from the area of Critical Path theory, which is a fundamental concept in Computer Science and Mathematics. The blueprint below shows how the players jump ability is depending on their energy.

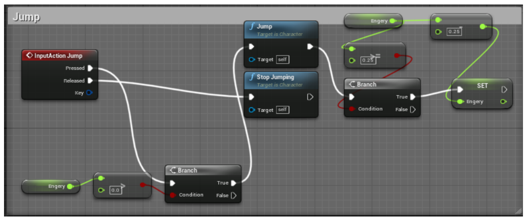

Figure 9: Third Person Character Blueprint: Jump functionality

References

Games, Epic. *Programming Quick Start*. URL: `https://docs.unrealengine.com/en-US/Programming/QuickStart/index.html` (visited on 08/06/2020).

— *Unreal Engine*. URL: `https://www.unrealengine.com/en-US/` (visited on 08/06/2020).

Wood, Jesse. *Jump Demo*. URL: `https://www.youtube.com/watch?v=R1xpIcRARMc` (visited on 08/06/2020).

— *Jump Github Repository*. 2020. URL: `https://github.com/woodRock/Jump` (visited on 08/06/2020).