Virtual Reality Wave Survival Game

Isaac Schwab

December 2017

Abstract

The goal of this project was to build a virtual reality wave survival game. The game is an immersive experience that showcases the current generation of virtual reality systems like the HTC Vive. The gameplay involves a player defending against waves of attacking enemy monsters. The player can interact with beam rifles and lightsabers to attack the oncoming enemies. A realistic environment was created to help immerse the player in the experience. Room scale is built into the game, thus allowing the player to use slight locomotion to avoid enemy attacks. The incorporation of fast paced waves and player movement, not only create a mentally stimulating game, but lead to a substantially physical experience for the users. The combination of mental and physical interaction in this game creates an immersive, enjoyable experience. This type of game truly showcases the power of the HTC Vive and the current state of virtual reality.

1 Introduction

Virtual reality (VR) games are a great showcase of what can be created with current generation VR systems. There are many different categories of VR games. They employ different techniques and gameplay styles that immerse the players. Realistic environments, exploration, interactive storylines, task driven events, intense scenarios, and object interactions are just a few of the techniques found in VR games. In the related work section I cover some games that use these gameplay styles to create awesome experiences.

My game focuses heavily on the idea of having immersive object interactions. These interactions occur between the player and the weapon objects, and the weapons and the enemies. I also worked to create a realistic environment that helps immerse the player and add to the already intense scenario of having demon monsters running at them. I discuss how I implement and achieve these tasks in my accomplishment section.

2 Related Work

This project focuses on two popular areas of virtual reality, gaming and motion tracked controller interactions. At its core, this project is a wave

survival video game. The genre of wave survival games are pretty common in the entertainment industry. While there are a variety of wave survival games, I will highlight a few that I have used for inspiration.

The Brookhaven Experiment is a horror survival shooter for the SteamVR platform. It was released July 5, 2016, and features both a room scale and standing play area. The HTC Vive and Oculus Rift are both supported, with the requirement that users have motion controllers [1]. The premise of the game is that players fight off waves of monsters using the motion controllers as guns to defend themselves. This game was actually my first experience with a virtual reality headset, and I have to say it blew me away. I was actually terrified the first few rounds of playing and audibly screamed. Thinking back on the game, what really enabled this game's level of immersion was the detail in the environment, and the interaction with the motion controllers (weapons). The game didn't have much for locomotion, the player is placed in a position where they have some movement but are essentially locked in that room scale area to defend. I also believe that the hit detection around the player was simply a capsule shape, so not body tracked limbs. Additionally, this game was all about intense scenarios. Anytime a game can get you to scream, it must be pretty immersive. The developers used sound and lighting to help create this intense scenario. Figure 1 shows this great use of lighting, also notice the attention to detail in the environment. This game really motivates the idea that interaction with the motion controllers and environmental detail will be important to the immersion factor of my game.

Figure 1: Player perspective from the Brookhaven Experiment.

Space Pirate Trainer is an action wave survival game for the SteamVR platform. This game was released as an early access game on Steam in 2016 and was officially released in October, 2017. This game takes full advantage of the HTC Vive room scale to create an incredible experience.

The HTC Vive and Oculus Rift are supported, with the requirement of tracked motion controllers [2]. The goal of this game is to survive against oncoming space robots for as many waves as possible. Players gain a score depending on how many waves they beat, and the way they destroy the robots. Similar to The Brookhaven Experiment, Space Pirate Trainer was very visually immersive due to the environment. The best feature of this game is how it makes use of room scale tracking. The hit detection for projectiles fired at the user is not static, moving around the room, and changing your stance allow users to avoid projectiles. This enabled the game to not only be visually immersive, but physically immersive. After playing this game I was actually tired. It was incredibly fun to play, and I believe this is due to the physical immersion. This game was used as a motivation to create a great room scale experience in my project.

3 Project Accomplishments

This section covers the planning and development of the core features of my game.

3.1 The Environment

The environment is one of the more important aspects of this game. I wanted to have one of the most immersive experiences possible, so the visuals play an important role. I decided on a winter mountain theme for the game. In the following subsections I will break down the various elements that create the immersive landscape and my motivations for including them.

3.1.1 Mountain Landscape

Upon putting on the VR headset and starting the game, the user will be immediately surrounded by a winter mountain environment. The mountains and surrounding features are really just a backdrop that add to the realism of the whole environment. In real life when you look off into the distance there are many details that you notice, buildings, trees, people, etc... When building the game's environment I knew adding details outside of the immediate play space would be important. Figure 2 shows a birds eye view of the whole landscape. When you look at this view it becomes apparent how important a landscape is to the overall immersion factor. This is why I spent a portion of my time focusing on the mountain landscape.

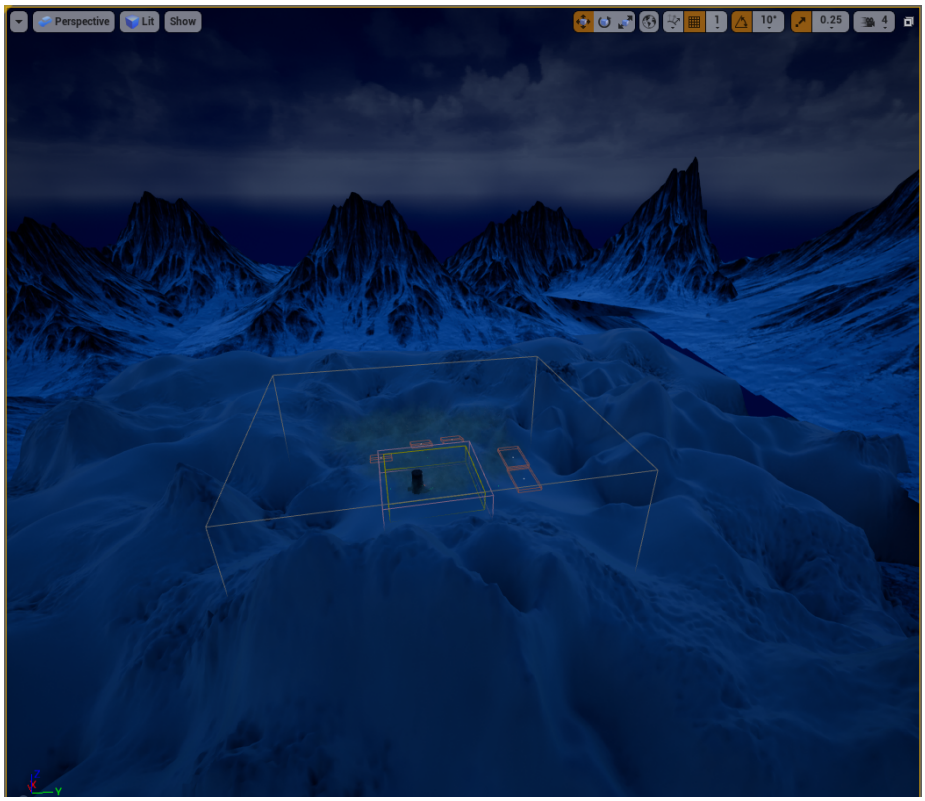

Figure 2: Birds eye view of the mountain landscape.

I will briefly highlight the techniques I used in Unreal Engine to create the landscape. The base of the mountain landscape was built using the engine's internal landscape modeling tool. This involved me sculpting the various contours of the landscape. Sculpt tools used were: sculpt, smooth, erosion, and noise. The surrounding mountain range is created by modifying scale, rotation, and position of a single mountain mesh to create the illusion of a range. To create the illusion of snow, a custom material was applied to the landscape. The material is based off assets provided by Unreal Engine.

3.1.2 Local Objects

Besides the surrounding mountain landscape, the player will also notice objects in the immediate vicinity. These objects serve two purposes. They add to the detail of the environment, and also add to the darker and gloomy theme. The second purpose is the border constraint that they add to the gameplay. VR headsets are unfortunately still corded, so I decided that having enemies running at the user from all directions could cause too much spinning, and in turn cause users to get tangled up. These objects create a natural constraint to keep enemies from running at the

player from behind, thus preventing conflicts with the headset cords. The main objects added to the environment were a tower, a statue, mountain rock, a table, and wooden pillars. The table acts as a spawn point for the player's weapons. This prevents the player from having to bend down to pick up the weapons to defend themselves. I actually thought of this idea after getting tired of having to get the weapons off the ground when developing their abilities. The statue has a fire emitter attached to it that acts as a source of light for the play area. The figure below shows these objects in the editor view.

Excluding the fire emitter, all of these local objects are static meshes. I don't have much experience 3D modeling or the time to learn how to do it, so the actual 3D models were pulled from free assets on the Unreal Engine marketplace [3]. Once imported into the engine I had to add materials and add custom collision constraints to the objects. The collision constraints define how other objects in the game interact with them on collision. Most of these objects act as a boundary, so their collision is set to block other objects.

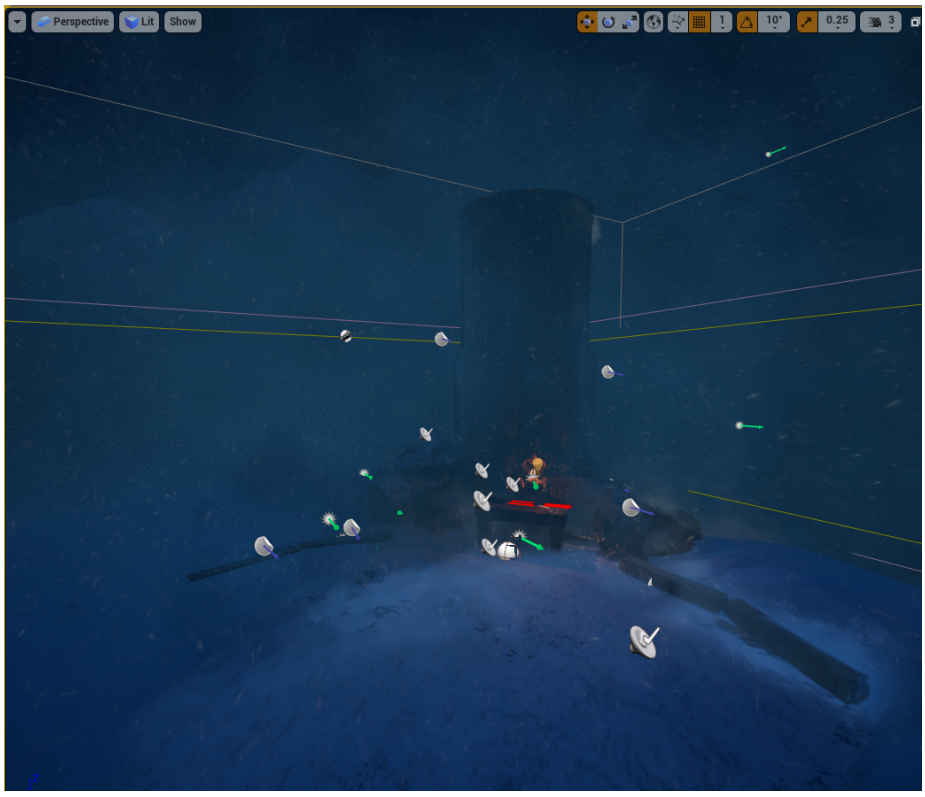

Figure 3: View of play area, notice the local objects and the particle snow effects.

3.1.3 Snow Effects

My favorite and probably the most immersive part of the environment is the snow storm effects. After building the mountain landscape, I wasn't totally convinced that it portrayed the theme I was trying to create. I felt it was a little boring, and was lacking "life" or movement. I spend about two weeks every winter in the rocky mountains, and my absolute favorite is being completely engulfed by a snowstorm. I decided that this effect could be beneficial for the game. Not only does it add to the realism effect, but the limited visibility amplifies the intensity of fighting off waves of enemies.

A variety of techniques were used to create the snowstorm illusion. Particle emitters are the core of the visual snow effects. Unreal Engine provides a great system for building advanced particle systems. I used a base snow particle system provided by Unreal [3]. I created multiple instances of this base system. These instances were blowing snow, falling snow, gusts of wind, and smoke that emulates the reduced visibility that occurs during a snowstorm. The use of sound is what really completes the snow storm effect. Snow storms commonly have high winds, so I found a variety of sound assets that mimicked the sounds of a winter storm. These were added to the environment as location based sound cues. This means that depending on the location of the player, they will hear different sounds. The particle effects and sound cues can be seen in the figure above.

3.2 Player Interactions

The interactions with the player, the weapons, and the enemy were the most time consuming and difficult step in creating this project. Normal interactions in Unreal Engine can be difficult, but then throwing in virtual reality interactions made it all the more challenging. I knew that these interactions would be the defining functions of my game, so I was determined to get them working correctly. All the little details in these interactions would in turn create the immersive experience that this game aims to create.

3.2.1 Player-Weapon Interaction

The first challenge that I needed to address was allowing the player to pick up weapon objects. I started with the VR template in Unreal Engine. This gave me a spawned player and tracked motion controllers. I then created blueprints for the motion controller objects. These blueprints contained events for grabbing and releasing objects in the environment. I mapped these actions to the Vive controllers grip button. When a player moves their hand so that it is overlapping an object, they then press the grip button. This attaches the object to the controller mesh. Pressing the grip button again releases the object that was in that hand. The blueprints for these interactions both behave similarly. The motion controller object has a collision sphere around it. When the grip button is pressed the blueprint first checks if it already has an object, if it does it detaches the

object (release event). If the controller was not already holding an object it then checks for any overlapping objects in the collision sphere that can be picked up. If there is one, it grabs the nearest object and attaches it to the root controller element.

The next function of the player-weapon interaction, is actually using the weapons. Both the lightsaber and beam rifle have trigger actions. I was especially happy with how the lightsaber effect turned out. Upon pulling the trigger the lightsaber "turns on", which is a combination of spawning the blade, and playing matching sounds for a lightsaber. Even though the action just turns the lightsaber on and off, I really felt it added to the overall polish and realism of the game. Pulling the trigger of the beam rifle fires a laser projectile. The beam rifle and lightsaber are important to the game because they are the only way the player can defend against the waves of enemies. The blueprints to create these interactions start to get more complicated. When the action event is fired on the lightsaber, I first get the location transform for the parent object. I use the location transform as the attach point for the blade object. I also spawn and attach both a turn on sound, as well as a looping sound that plays while the lightsaber is on. Also every tick of the game I check the velocity of the lightsaber. If the velocity is above a threshold a swing sound is played. Again this was a small addition, but I believe it adds to the realism and immersion of the game. The beam rifle action behaves similarly, except I spawn a projectile object. This object is not attached because it is spawned with velocity. A sound is also played as the projectile is fired. Every trigger pull a new projectile is spawned. I will cover the interaction of the projectile and blade with the enemies in the next section.

3.2.2 Weapon-Enemy Interaction

The weapon-enemy interaction proved to be the most difficult because of the VR component of the game. I had pretty lofty goals for these interactions, which probably made this more difficult. The projectile fired from the beam rifle collides the the skeletal mesh of the enemy. On collision a hit particle effect is attached to that location on the enemy, and the enemy health is lowered. Once the enemy health hits 0, the enemy is destroyed. Figure 4 below shows the beam rifle enemy interaction. The lightsaber interaction behaves differently. When the blade hits the enemy mesh, the skeletal bone it collides with is detached from the body. This creates the illusion of the lightsaber actually cutting the enemy. The lightsaber interaction kills the enemy in one hit. Both collisions create sound effects on hit.

Figure 4: Beam rifle shot fired, and view of collision effect on enemy.

Actually implementing these interactions took a lot of time. The three main blueprints that I had to work on were the enemy, the blade, and the projectile. The interaction for these events is triggered by a Hit Event. I then break this hit event to get the actual component in the skeletal mesh that was hit. This step caused me a lot of issues, I had to get the exact combination of collisions for both the projectile and the enemy to get the correct hit result to return. After receiving the bone name, I then take the skeletal mesh of the enemy and break the constraint that holds the bone to the body. The last big issue actually arose because of the VR motion controllers. When the sword collides with the enemy mesh, it isn't actually being detected initially. I found this issue had to do with the sword being attached to the motion controller component. However, this isn't an issue with the gun projectile because it is fired and is not attached to anything. I spent a long time searching for a solution to this but I couldn't find anything. I ended up building my own work around to fix this issue. I knew that the projectile being fired from the gun gave me the correct bone hit, so if my sword fired a projectile the detection might work. This did in fact work, however, swords don't fire projectiles. The solution for this was to fire a non moving, invisible projectile every frame (called a tick in Unreal Engine). Then the next frame destroys that projectile and creates a new one. This allows the projectiles to map with the movement

of the lightsaber blade. Solving this issue was a huge accomplishment for my project. It allowed the lightsaber interaction to actually work, which adds a lot to the gameplay. The other reason I wanted to highlight this accomplishment in such detail is due to the fact that these are the issues that arise with VR development. I spent more hours than I care to admit trying different solutions. Searching online for answers didn't help due to the infancy of VR development in Unreal Engine. In the end persistence and thinking outside of the box enabled this accomplishment. Figure 5 below shows the lightsaber-enemy interaction from in game.

Figure 5: In game action of lightsaber hitting the enemy.

3.2.3 Enemy-Player Interaction

This interaction defines how the enemy attacks the player. In this game the enemy characters cause damage to the player when they come into contact. The enemy makes an attack animation. Upon being hit by an attack the player will have the edge of their vision affected to represent the hit. If the player is hit twice by the enemy then the game ends.

To build this interaction I had to give the player blueprint a capsule collision component, and the enemy AI an attack collision component. When the enemy collision and the character collision overlap, an affect health event is called, which subtracts from the player's total health. It also triggers the attack animation and the visual indicator that the player is being harmed.

3.3 Enemy AI

The enemy characters have an AI component that allows them to find and run at the player. In this game mode enemies are simply mindless monsters, so their abilities are pretty limited. Upon being spawned each enemy AI continuously calls a function called TrackPlayer. This function is given the player object as the target. Once called, the AI will track the player and run to them. The run behavior and object avoidance is handled by Unreal Engine's internal AI class. The majority of the work for the enemy actually occurs in the character animation blueprint. This is where the running, attacking, and roar animations occur. The enemy demon mesh, and basic animations were available for free from an animator on YouTube [4]. For the majority of the development of my project I used the default Unreal Engine mannequin as the enemy, however, the default mesh was just not scary. Once I found the demon mesh set online I decided to try and incorporate it into my game. This ended up being much more complicated than anticipated. I had to make many modifications to the demon animation blueprint. After I successfully got the new enemy model spawning and chasing the player, I then realized that all my weapon-enemy interactions were broken. The issue was that the new demon mesh had no built in collisions. The solution was to create a new physics asset for the mesh. Other minor changes in the player, weapon, and spawner class were required. Figure 6 shows the demon enemy mesh and a glimpse at the blueprint that controls animation events.

In the end I was very happy with the results of incorporating this demon mesh and animations. The changes drastically helped improve the intensity of the game, which in turn leads to a better overall experience.

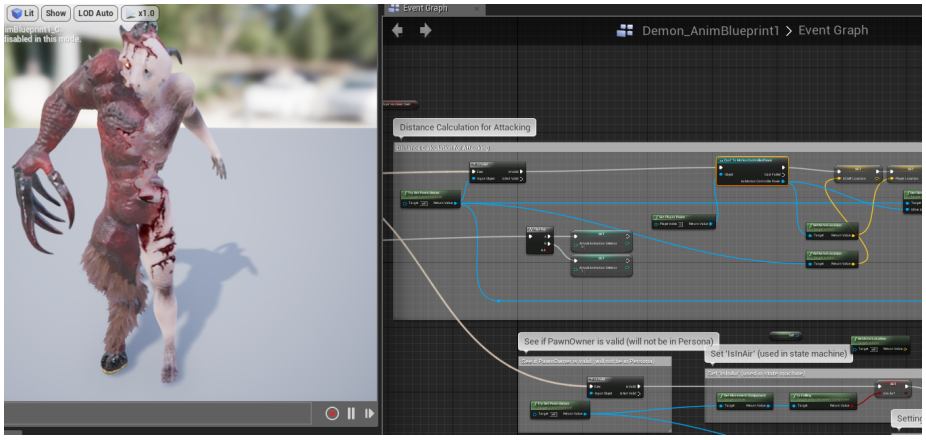

Figure 6: View of the enemy character and part of the blueprint that drives the animation events.

3.4 Round System

The last core function of the game is the round system. The round system encompases how I spawn enemies, regulate difficulty, display text to the user, and end the game. The majority of this work occurs in the game mode, enemy, and enemy spawner blueprints. I will cover the main parts of the round system in the following subsections.

3.4.1 Spawning Enemies

The enemies in the game are spawned at the beginning of each round. The actual mechanic that spawns the enemies is a blueprint object called EnemySpawner. These spawners are placed around the level, and each spawner has a volume. I placed the spawners such that enemies would run at the player from all directions, except from behind. The spawner objects know when to spawn a new enemy based on function calls from the gamemode blueprint. When called upon, the spawner first checks the current number of enemies that have been spawned, if that number is less than the max allowable enemies for that round it creates a new enemy. The enemy is spawned at a random location within the spawner's volume. The figure below shows the location for the spawners and their volume boxes.

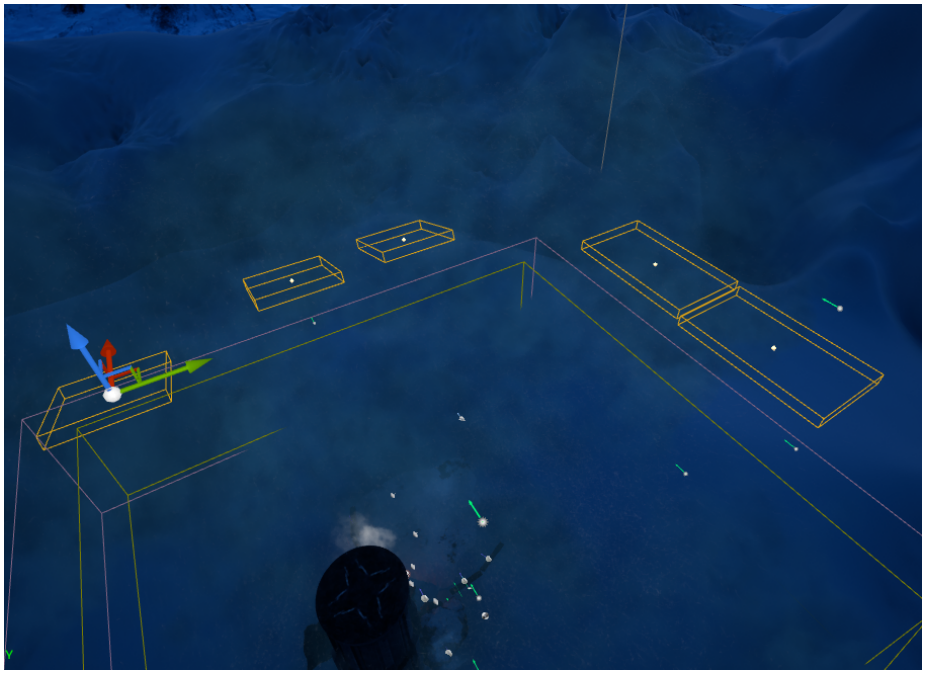

Figure 7: Visual layout of the EnemySpawner objects on the level.

3.4.2 Creating and Ending a Round

The creation of new rounds occur in the gamemode blueprint. At a high level my system works by first spawning a starting text component in front of the player. These text components are made in the Unreal Engine UMG editor. I then call a create round function. This function sets the difficulty for the current round (max allowable number of enemies to be spawned), increments the round counter, resets the enemies remaining, and resets the current enemies spawned. It then spawns a round number text component so that the player is notified of the starting round. The spawn enemies function described in spawning enemies section is now called. Now enemies are being spawned.

Ending the round occurs by checking every frame if there are 0 current enemies remaining. If there are 0 enemies remaining then the create round function from above is just called again. Also it's important to note that the enemies remaining count is decremented within the enemy AI blueprint when that AI is killed.

Lastly, to actually end the game, the player blueprint will call the Event Player Dead if their health is ever less than or equal to 0. When this is called the gamemode spawns the game over text component and sets the game mode to paused.

4 Conclusion and Future Work

The first time I tried on a virtual reality headset I knew that I wanted to create a VR experience. CSCI 5619 gave me the opportunity and the resources to attempt VR development. This project was an amazing learning experience. I spent around 70 hours creating this game, and learned a lot about VR development and Unreal Engine. The more I worked on the project, the more ideas I came up with and wanted to implement. Time limited me in the end. In this section I will wrap up this project by talking about some of the future work I would complete given more time and give my final insights on the outcome of the project.

There were a variety of additional tasks and ideas that I wanted to include in this game. I worked hard to complete all the core features, so I would describe most of these items as "polish" to the game. The location and animation of where the hand grabs the rifle and lightsaber aren't perfect. The orientation and model of the hand needs to be slightly modified. This change would just add to the realism of actually picking up and holding the weapons. I also would have liked to add some more body tracking functionality. This was an idea that I brought up in my initial project proposal, but in the end it would have been almost an entire side project to just to incorporate it with my game. The last big grouping of additional work I would like to implement can be summed up as modifications to the gameplay. Building enemies that fired projectiles at the player would add a whole new dynamic to the gameplay. I could then include additional interaction with a shield, or the lightsaber reflecting enemy projectiles. The last main function I would like to tweak would be the round waves difficulty. Variable run speed, dodging, and attack

abilities could all be added to the enemy AI so that they are more variable, this would create more challenging late game rounds. Additionally, adding ammo and cooldowns to the player weapons would also make the game more challenging and strategic.

Overall, I am very pleased with the outcome of the project. I was able to complete all the core functions that enabled enjoyable gameplay for the user. I discuss the accomplishments in more detail in the sections above, however, I want to highlight the player-weapon and weapon-enemy interactions in this section again. These interactions are what stand out as my most important accomplishment. While the environment details and the round system are important to this game, the actual interactions with the motion controller and the enemy are what showcase the power of VR. Picking up and using the weapons really helps create the immersive experience that I wanted to achieve, the player is no longer sitting in a chair and hitting buttons on a keyboard. They are now standing, picking up weapons, swinging a lightsaber, aiming, and firing a beam rifle. These type of interactions are what give VR games an edge over traditional monitor based video games. I plan to continue working on this game even after the class is done, it provides a great avenue into improving my VR development abilities, and it is very exciting time to be building virtual reality experiences.

References

[1] The Brookhaven Experiment. 2016. http://store.steampowered.com/app/440630/The_Brookhaven_Experiment/.

[2] Space Pirate Trainer . 2016. http://store.steampowered.com/app/418650/Space_Pirate_Trainer/.

[3] Unreal Engine Marketplace. 2016. https://www.unrealengine.com/marketplace.

[4] Free Horror Game Monster + Animations (Unreal Engine 4, Unity...Any Game Engine). 2016. https://www.youtube.com/watch?v=n3IH4qGf_ZA.