PathFinder

CS 6334.001 VR Project Report

Saffat Ahmed Aidan Gow Akhil Shashi Rachel Pitts

Introduction

Our project is a VR simulation of building and using paths from one point to another. We created an experience of interactivity with different objects and obstacles for the user that requires various levels of problem-solving to overcome. Our motivation was to make a game that would allow the player to freely solve these puzzles and create their own paths to a goal.

We used the Unity engine to build our game and used free assets from the Unity Asset Store to create our game world. The VR implementation was mainly done through Unity's XR Interaction Toolkit which allows us to easily program and use a VR headset and VR controllers. Other aspects were created from scratch like the terrain and several scripts.

The game has two levels that the player can traverse through and complete. Each with its own unique puzzles and surroundings. Supplied with a variety of interactable objects, the player will be able to problem-solve their way to the right flag goal. This game is playable mainly through the Oculus Quest 2 but also features PS4 controller support and future plans for keyboard and mouse support. The player will be able to place and rotate the objects so that they may make a path through the terrain toward the goal.

Method

When the project was introduced to us, we were given a series of questions to help decipher the correct method for our Pathfinder game. Since our project was focused on virtual reality, we would need to build a virtual world either from a pre-generated world or from scratch. As for the player interaction, the main idea is that we want our player to get from point A (spawn point) to point B (end goal) with a series of objects in between to help accomplish this.

We can separate this into two tasks, building the environment and player/object interaction.

Environment

In order to create a terrain, we grabbed a series of online assets to properly construct this. Those assets were Crates00, flag, Free_Rocks, Rocks and Boulders, TerrainSample Assets, and Tree9. We used these assets to spawn a series of objects that will be used to create the environment. We have constructed three zones, a starting zone with trees(Starting) that connects to a desert area with a lot of rocks(Cliff), and a similar area as a grayish mountainous region(Bridge). Each of the surrounding zones(Cliff and Bridge) outside of the starting area has its own goal flag. The starting area is a very green area with trees and some objects that can be interacted with in the surrounding levels. All of these zones are surrounded by a tall mountain range that forms the perimeter of the map. The surrounding areas' mountains are darker than the starting zone. These are depicted in the below figures.

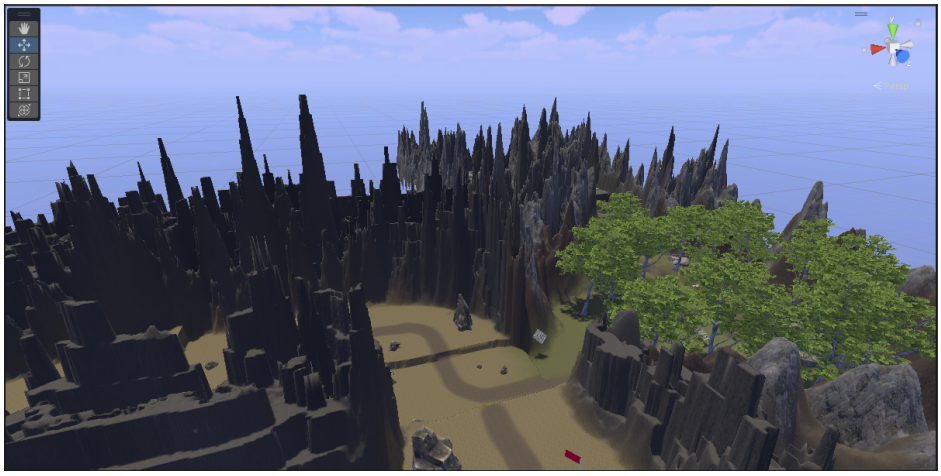
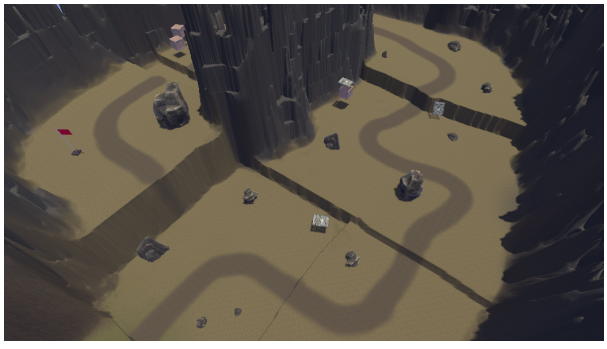
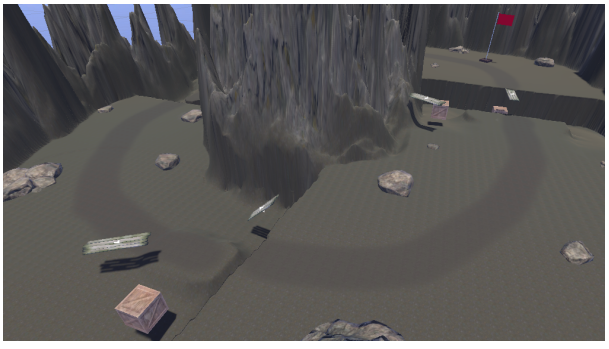


Figure 1: Aerial View of the Whole Map, with Each of the 3 Zones





Figures 2 and 3: Aerial View of the Cliff(left) and Bridge(Right) obstacle courses

In order to help with immersion in the virtual world, we added ambient wind and bird sounds. Unity makes interaction with idle objects and the environment pretty straightforward. Most of the work came from constructing the entire zone or level. In order to make sure the direction of

the gameplay was clear, we have enough interactive objects to solve a certain region. And if the user makes too big of a mistake like an object going out of bounds, the object will respawn in the area where it can be used again.

Interaction

Now that the environment or level has been constructed, we can create the player and let them play the game. We need the player to be able to interact with the objects so that we can pick up and place them where they need to go so that they can progress to the goal. These objects are crates, stairs, and planks, all made of wood. Each is featured in the starting zone so that the user can test it out as much as they want. The stairs will be able to ascend to about the height of the crate and the planks can be used to cross long gaps. The objects can be placed on each other. Each is depicted in the figure below.

Figure 4: Interactable Objects

In order to add intractability between the player and the objects, we used a package called the XR Interaction ToolKit. This toolkit creates a high-level, component-based interaction system for creating VR and AR experiences. It provides a framework that makes 3D and UI interactions available from Unity input events. We use this alongside the XR rig, which is used to track movement from the headset to the in-game camera. So now we can accurately move the camera according to the player's head movement. As for the hands, each features a ray laser and whenever the laser highlights an interactable object within its range, the player can use the grip buttons to be able to grab and move them around. The player can also rotate and position the object with great accuracy with corresponding hand movements or the control stick. As for the exact controls, we can use:

- B(Hold) to sprint
- A to jump
- Left thumb stick to move (Continuous Move Provider)
- Right thumb stick to either rotate an object or turn the camera instantly(Snap Turning)
- Y to switch the camera between first and third person point of view
- X(Hold) to enable snap turning
- Grip Button to grab objects if the ray is pointing at grabbable object

Below is a graphic for the controls on the Oculus Quest 2.

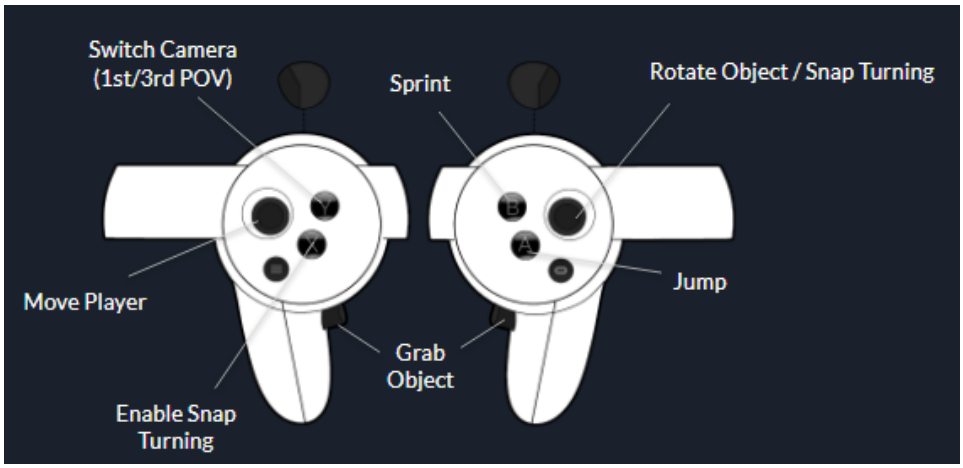


Figure 5: Oculus Controls

A PS4 controller is also an option for users for use with the headset. The game will automatically opt for gamepad controls if a PS4 controller is detected. For the PS4 controls, we use:

- Square (Hold) to sprint
- Cross to jump
- Left thumb stick to move
- Right thumb stick to rotate or turn the camera (Snap Turn)
- R1 (Right Bumper) to enable snap turning
- R2 (Right Trigger) to grab available object

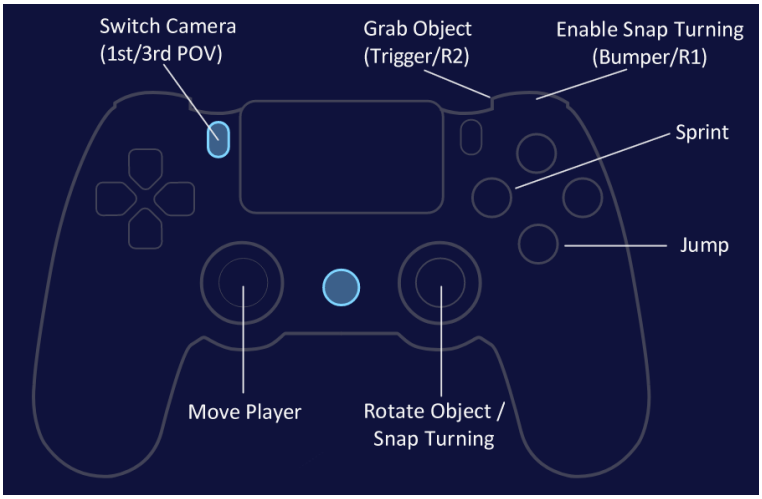


Figure 6: PS4 Controls

Experiments

For our game to work properly, we had to perform many experiments to handle different aspects of the game like player movement and object interactions. Most challenges we faced, we overcame by doing many repetitive experiments by slightly adjusting certain parameters so every game aspect would work well with each other.

Player Controls

The user has multiple controls that can affect the in-game player to perform certain actions. While some controls like the XR interaction controls were easy to implement, we still needed to adjust certain elements to match what we had planned for the simulation.

The first experiment we conducted was how we should go about with the physical movement of the player character in the game and what controls would work best for the user. The two options given by the XR Interaction Toolkit were the continuous move option and the teleportation option. Initially, we tested the teleportation option as we were learning the unity system through a VR tutorial as shown in Figure 7 and realized it might not be a great fit for our game. Since our game requires the player to make paths and use those paths to cross gaps or higher elevations, having the ability to teleport defeats the purpose of making a path. Therefore, we opted to use the continuous movement option that moves the in-game player character directly using the control stick. One of the negatives of the continuous move option is that it could cause the user to lose balance or get motion sickness since the user themselves aren't moving in real life. Changing the moving speed and adding an optional sprint button could mitigate some effects for some people but we decided to experiment more for a better solution.

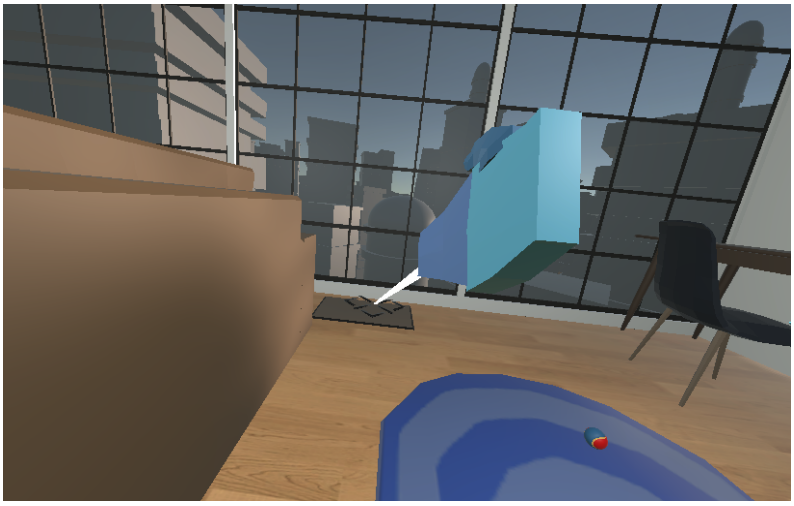


Figure 7: Teleportation Controls in VR Demo Tutorial

To prevent motion sickness while moving in first person, we decided to experiment with different camera options that allowed the in-game player character to move, without having to move the camera view the user was seeing through. We opted to use a third-person perspective camera that could show a representation of the player character's current position so you could still freely control the movement as shown in Figure 8. We believed this to be the best option since anything else we possibly considered (changing movement back to teleportation, removing any camera movement besides headset tracking, etc) would dramatically change many core mechanics of the game. Implementing the third POV camera proved to be a bit difficult as we wanted to have the camera be in a static position but still be able to view the in-game player from any position. To do this, we designed a system that would have a dynamic third-person camera that was always following the player's position that could be switched to at any moment by pressing a button. When switched to, the camera would stop actively following the player to prevent unnecessary motion sickness. Pressing the same button again would switch the camera back to the first POV and the third POV camera would again actively follow the player character from a set distance away. With this setup, we were allowed to have a static third-person camera to prevent motion sickness and still be able to see and perform any actions with the in-game player character by dynamically changing its position when not actively using it. We were also able to change the location of the third POV camera based on not just the position of the player character, but as well as the rotation of the player character that is being tracked by the VR headset. With this, the third POV will always be a set distance behind the player character relative to their position and rotation. This gives the added feature of allowing the user to adjust the position of the camera simply by rotating their headset up, down, left, or right.

Figure 8: The third-person perspective camera viewing the player character

For separate controller support we experimented with using different methods of switching between the VR Hand controllers and a regular gamepad controller (tested specifically with a PS4 controller) during runtime and on initialization. We opted for an initialization approach to controller setup, in which a hierarchy of controllers is used to determine the main controller when the game starts. For convenience of testing, this hierarchy was established as: Gamepad Controller > VR Hand Controllers. This is due to ease of disconnection for the gamepad controller versus the changing the VR controller setup.

In order to display if an object is selected to be picked up, we cast a ray forward from the center of the camera, and on hitting an interactable object switches the material of the object to red in order to highlight it. When the user looks away from the object the material is reset to its default value. This can be seen in Figure 9. Object highlighting is disabled when using the VR hand controllers, as seen in Figure 10. This method was chosen over using a reticle on the screen since a constant reticle can be unpleasant for VR headset users when alternatives can be used.

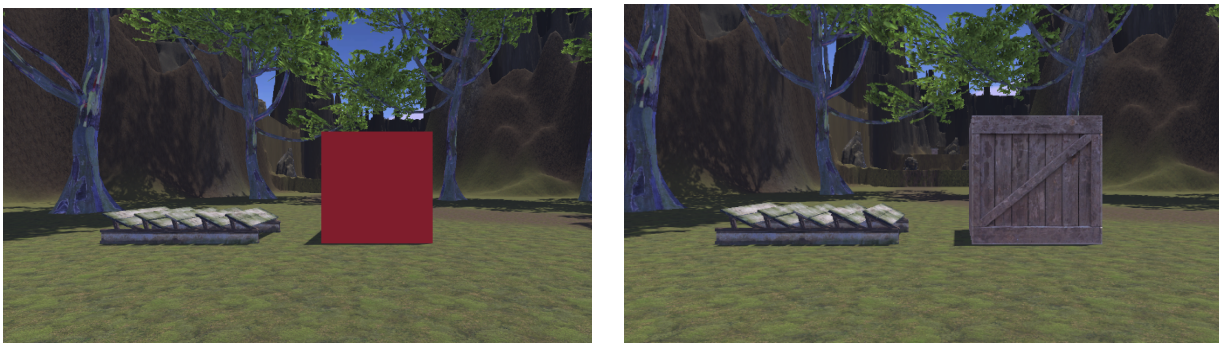


Figure 9: A highlighted and unhighlighted crate

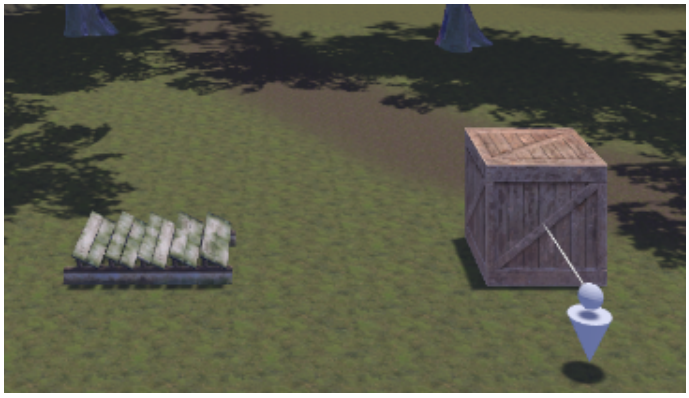


Figure 10: Third-person perspective using VR hand controllers on an unhighlighted crate.

Collision

A major aspect of our game relies on the collision of the objects, terrain, and player to work perfectly which naturally required a lot of experimentation in order to get the collision interactions to work properly. We initially assumed that the collision between everything that had a collider would break under the default unity physics system as objects are being forced against each other by multiple forces like the player, terrain, or other objects so it could be possible that they intersect or collide in an unnatural way. Thankfully, we didn't experience too many problems with the physics system but we did consider some alternatives just in case.

One alternative was to implement sockets, which allowed us to place an object in a specified position and it would be unable to move. This could prevent any unnecessary movement of the object when the player's collider jumped or pushed it from the side. This solution was eventually not considered since it would remove the free-form object manipulation we had initially planned since it would tell the player where the object was required to go for every obstacle. The other alternative was to reduce the friction of a given object with different colliders. This could allow for the object not to slide around and aggressively collide with anything unintended. In the end, this solution was not required but if needed, it would be easy to implement.

One unexpected problem we ran into when creating a demo for the game was how the objects should be manipulated. We naturally thought the obvious way was for the user to grab an object by either moving the VR hand controllers so the in-game hand could grab it or cast a ray that could be pointed at an object from a set distance away and put it in the player's hand. This implementation would work fine for smaller objects but larger objects posed a significant problem with the collision of the player character. If the object's collider overlapped with the player's collider, unexpected jagged displacement of both would occur causing them to fly out of bounds or move too aggressively for VR headset users. To prevent this, we experimented with ways we could stop this sudden movement. The following table shows several parameters and approaches we tried as well as the results/side effects of each:

	Stop Unintended Player Movement?	Stop Unintended Object Movement?	Side Effects?	Comments
Disable Change in Y Position of Player	Partially	No	Player is unable to move up and down	Can not use since jumping and moving up and down elevations is crucial
Disable Change in X/Y/Z Rotation of Player	Yes	No	Player collider unable to rotate on X/Y/Z axis	No need to use these rotations for the collider
Disable Collision of Object When Held	Yes	Yes	Object could fall through terrain and other colliders when let go of in certain positions	Usable but requires either some way to disable collision when specifically colliding with the player or respawning the object if it falls through terrain collider.
Manipulating Object from a Set Distance	Yes (unless close)	Yes (unless close)	Objects are slightly limited in manipulation	Helpful for large objects since it is already hard to manipulate objects larger than the player

After attempting each option, we found using a combination of disabling any rotation of the player collider, disabling the collision of the object when held, and manipulating the object from a distance to be the best fit for our game. With all combined, we are able to prevent unnecessary movement between objects and the player when held while keeping freedom for the player to move how they want. Some of these solutions were critical for us like disabling rotation since, without it, we would have the player falling over which would be very unsettling for the user to experience. To prevent the side effect of the disabled collision of the object when held, we decided to make an object manager script that would respawn the object if its position was too far from its initial spawn location. This helps if the object accidentally falls through the terrain as well as helps if an object is too far to be grabbed.

Scale

When designing the obstacle courses, we had to consider a suitable size for each of the objects that would be able to work for different circumstances. Having the object be too small wouldn't make it very useful if the path made by it can be crossed using the basic movement options of the player character. Having the object be too big would make it very difficult to manipulate as well as keep it from clipping through the terrain. After testing different sizes, we landed on a suitable size for each object that scales well with the player and the rest of the terrain as shown in Figure 11.

Same as for the object sizes, we experimented with different terrain obstacle sizes. For the same reasons, we needed to get the scale of the terrain right alongside the objects or the obstacles would become incredibly easy or almost impossible. Unity's terrain editor was easy enough to use and setting different heights and distances for the terrain iterating on the scale of the obstacles was easy to do.

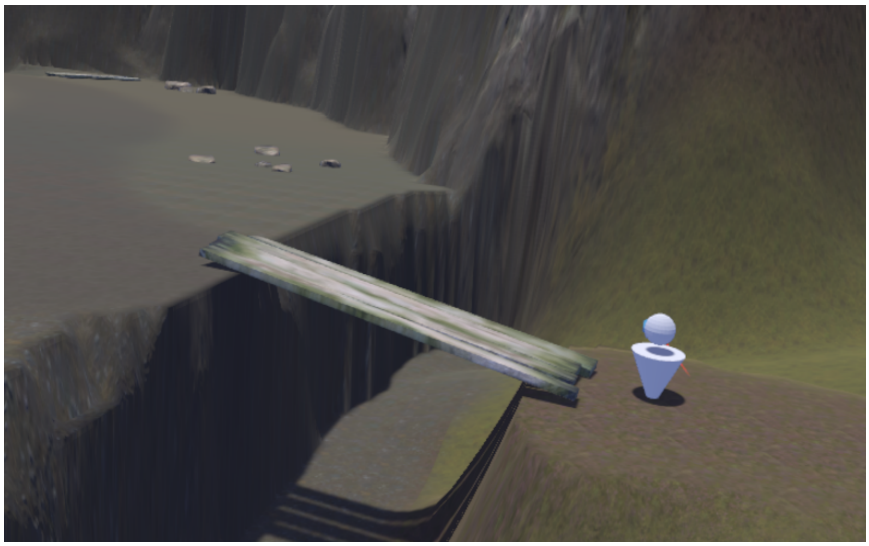


Figure 11: Creating a bridge using the plank object

Conclusion

In conclusion, our project allowed us to gain a comprehensive understanding of the Unity engine and its capabilities for creating virtual reality experiences. Through the development of PathFinder, we were able to create an engaging and immersive VR game that challenges players to use their problem-solving skills to create paths from one point to another. This project has provided valuable experience and knowledge that will be beneficial for future Unity projects and virtual reality simulations. Overall, we are pleased with the outcome of our project and are excited to continue exploring the possibilities of VR development using the Unity engine.

Contribution

Saffat Ahmed

- Designed and implemented the VR controls for the game
    - Implemented XR Interaction Toolkit for VR headset and hand controller compatibility
    - Wrote the Player Controller script that maps the player actions to the VR hand controllers (Sprinting, Snap Turning, Jumping)
    - Wrote the Camera Switch script for managing and switching to the third-person camera.
- Created terrain for starting area and obstacle courses
- Wrote Interactable Manager script for managing objects' distance from the initial position so they would respawn or despawn objects that are too far.

Aidan Gow

- Facilitated team communications and organized meetings outside of class
- Wrote significant portions of final report
- Created a plan/layout for the proposal, midterm, and final reports
- Experimented with the collision of objects

Akhil Shashi

- Experimented Keyboard and Mouse controls for the game.
- Created terrain for starting area and obstacles for the initial version of the game.
- Implemented Keyboard and Mouse controls for the initial version of the game.
- Assets exploration and implementation.

Rachel Pitts

- Experimented with gamepad controller support
- Implemented PS4 controller support and switching mechanism between controllers
- Created object highlight system for PS4 controller use
- Implemented collision sfx for objects