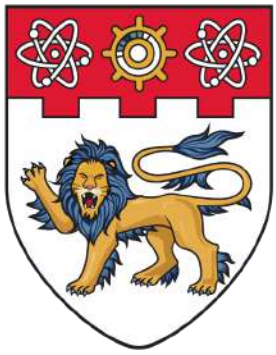


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4001 - VR Assignment Report

Team 13 - JANAS

Team Members:

Garg Astha

Gupta Suhana

Jaheezuddin Aneez Ahmed

Lee Jun Heng

Ramasubramanian Nisha

Table of Contents

Introduction **3**
SteamVR Input and Interaction System **3**
General Interactions **4**
1. Snap and Turn - Joystick (Right Controller) 4
2. Teleportation - Y button (Left Controller) 4
3. Continuous Movement - Joystick (Left Controller) 4
4. Jumping - A button (Right Controller) 4
5. Crouching - B button (Right Controller) 5
6. Grab and Throw 5
7. UI interaction 5

Some obstacle specific interactions **5**
1. Pressing button gameObject 5
2. Lever gameObject 5
3. Line Renderer (Shooting) 5
4. Climbing 6
5. Explosions 6

Mapping of controllers to different actions **7**
Theme **7**
Scenes and the obstacles **7**
1. UI Scene 8
2. Tutorial Scene 8
3. Shooting Scene 8
4. Enter Scene 8
5. Laser Scene 9
6. Lever Scene 10
7. Grab and Throw Scene 10
8. Climbing Scene 10
9. End Scene 11

Mechanics for Changing Scene **11**
User Experience **11**
1. Vibration feedback 11
2. Tutorial 11
3. Restarting the current scene on collision 11
4. Seamless scene switching and door animations 11

Future Work **11**
Appendix **12**
References **22**

Introduction

The objective of the assignment is to work in our respective team to design and develop an interactive VR game focusing on creating an obstacle course and overcoming it. The main focus is appreciation of the fundamental concepts associated with VR, learning how to create a VR game using Unity game engine, and designing and implementing a suitable application targeting an Oculus Rift.

Link to GitHub repository for code - https://github.com/AsthaGarg16/VR_project
Link to the zipped code on OneDrive - VR_project.zip

SteamVR Input and Interaction System

Actions are the center of the SteamVR Unity Plugin. In order to concentrate on the user's intent and their actions, SteamVR Input abstracts away the elements of your code that are device-specific. We just concentrate on the final phrase, "grasp the block," rather than implementing the code to recognize "pushing the trigger button down 75% of the way." Although we can rebind "grab" to our preference in a standard interface, the default definition of what "grab" implies must still be configured. Also, without requiring any changes to our code, when a new input device is released, we can publish bindings to share for that device.

SteamVR has separated actions out into different types of input and one output type. The ones used in the project are as follows:

- Boolean - true or false
- Vector2 - two analog values
- Skeleton - Orientations for each bone in a hand
- Vibration - Actuating haptic motors

The Interaction System is a series of scripts, prefabs and other assets. The major components used in the project are as follows:

- Player
- Teleporting
- Interactable
- Throwable
- UI and Hints

The scripts written for customized movement and obstacle interaction can be found in the appendix.

General Interactions

1. **Snap and Turn - Joystick (Right Controller)**

   On moving the joystick to the left or right the player rotates by 45 degrees in that direction about the Y axis without change in the position. This means that the player can look around the same point.

2. **Teleportation - Y button (Left Controller)**

   This gives a boolean value depending on whether it's pressed or not. When true, the player can see the teleport ray and is able to teleport to either teleport area or unlocked teleport point. THis is made possible by using the following components:

   1. TeleportArea

   - This is a teleport area that is made up of a mesh.
   - When teleporting onto these, the player will teleport exactly where they are pointing.
   - Add this component to any object with a collider and a mesh renderer to allow the player to teleport onto it.

   2. Teleporting (Prefab)
   - This prefab sets up the entire teleport system.
   - Dragging this into the scene gives the ability to bring up the teleport pointer in the game.

3. **Continuous Movement - Joystick (Left Controller)**

   Vector2 actions are a combination of two analog values. An X and a Y. These are used to move the player. The defined speed in the script is multiplied with time for displacing the player. Player has a Character Controller component added to it that does not make use of ‘Rigidbody’ physics. This integrates well with SteamVR Player prefab as it already has physical properties.

4. **Jumping - A button (Right Controller)**

   Boolean actions are values that are either true or false. So on pressing the A button on the controller, the script gives an upward velocity to the Character Controller. To make it smooth and improve the user experience, we use square root function with jump height and gravity to calculate the velocity in Y direction.

   To prevent double jumping, the script first checks whether the player is grounded. This prevents the player from initiating jump again while in mid-air.

5. **Crouching - B button (Right Controller)**
   Unlike jumping, pressing the B button on the controller reduces the height of the character controller component of the Player prefab to simulate the effect of crouching.

6. **Grab and Throw**
   The grabbing interaction can be performed by hovering over a grabbable object and holding down the GrabGrip trigger. This results in the grabbable object being picked up and held as long as the trigger is pushed down. The grabbed object can be released by releasing the GrabGrip trigger. Throwing interactions are enabled by computing the release velocity of the grabbed object using the player's hand movements when releasing the grabbed object.

7. **UI interaction**
   SteamVR plugin provides a "SteamVR_Laser_Pointer" script that handles the heavy functionality of creating a pointer, physics raycaster & line renderer. This script is added as a component to the controller of the Player prefab. The next step is to create a scene-wide event handler that holds three giant switch statements. Each statement resides within one of the three following types of pointer interactions: on enter, on exit, & clicked. The clicked event is the result of pressing the trigger which makes the laser green providing visual feedback to the player.

Some obstacle specific interactions

1. **Pressing button gameObject**
   This has been implemented by making use of "Interactable" and "HoverButton" components provided by SteamVR. On hovering over the button game object, it gets highlighted meaning it can be interacted with. On pressing down the button, the moving part moves downwards by a small amount. This ButtonClick event is detected by the button component (by HoverButton component) and the function for handling it is initiated from the defined script.

2. **Lever gameObject**
   This has been implemented using the interactable component in SteamVR and the hinge joint component provided by Unity which is based on RigidBody physics. Due to the interactable component, on hovering over the lever, a yellow highlight appears which means it can be moved. Each lever has a hinge joint component where the angular limits, anchor and axis have been set. The axis has been set as the z axis while the anchor is the bottom most point of the lever (the spheres). The angular limits are 45 degrees on either side of the z axis.

3. **Line Renderer (Shooting)**

   Line Renderer was being used for the laser component in the shooting scene
   It destroys the game object when the line renderer comes into contact with the game object collider to imitate it being shot down.The trigger button pressed was being enabled in the controller binding and when the trigger button is being pressed the laserGun function which has the transform position of the muzzle of the gun will initiate the lineRenderer component and shoot it forward to where the user is pointing with the gun.

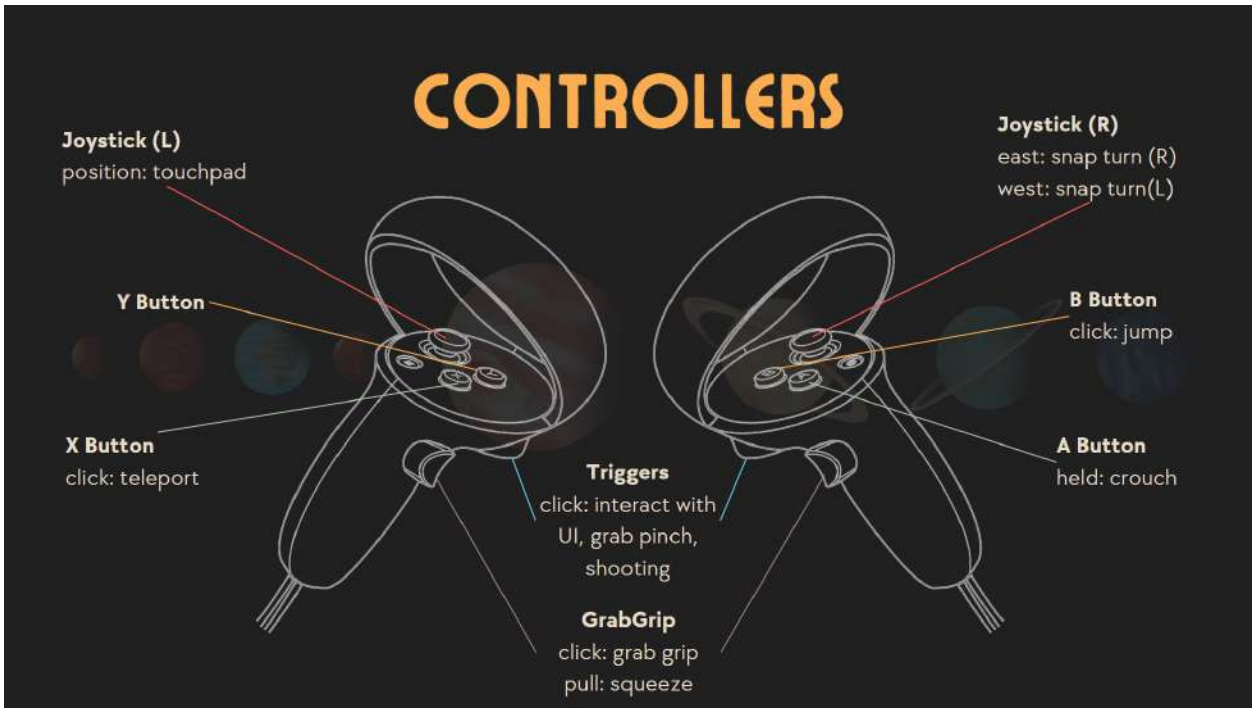
4. **Climbing**

   The controls for climbing is GrabGrip which gives a boolean value. The SteamVR input system is used to detect when the player has grabbed onto a climbable object, and then moves the player up the object at a specified speed. This is done by casting a ray forward from the player's position to detect the object tagged as ‘climbable’, and then using the SteamVR action system to detect when the player is grabbing onto the object and move them up it using the transform.position property.

5. **Explosions**

   In our game developed in Unity, we have implemented explosions using an "OverlapSphere" function that detects destructible objects within a specified blast radius. Once these objects are identified, we apply an "ExplosionForce" to them, which displaces and destroys them based on their physics properties. To enhance the player's experience and provide a more realistic visual effect, we have also used the Unity Particle System to create a visually stunning explosion effect. The Particle System allows us to generate various particle effects such as smoke and sparks, which add an extra layer of immersion to the explosion. This implementation of the explosion effect in Unity not only provides a realistic physics-based destruction system but also adds an engaging visual experience that enhances the player's immersion and overall enjoyment of the game.

Mapping of controllers to different actions



Theme

Just A Normal Adventure in Space is an interactive obstacle course set in space, inspired by sci-fi media like Loki. The player is stuck on an unknown planet when aliens start attacking. Destroy the aliens with a gun to gain access to the spaceship.However, navigating the spaceship is not easy - complete all the obstacles to gain access to the control room of the spaceship and fly home!

Drawing inspiration from sci-fi media, a clean and cohesive futuristic design was adopted. To create and immersive environment sky boxes were used for every scene. All the scenes were connected and decorated using the same kind of assets to create a cohesive environment telling a story. Working with different types of illumination, sufficient lighting was added given the dark space background.

Scenes and the obstacles

The environment and layout of the scenes are created using the Sci-Fi Modular Pack Asset. The materials and textures are used to create the spaceship environment. The obstacles are either made from scratch or using the Obstacle Course Pack provided by Unity. The general flow of obstacles follows the theme of being stranded on a planet and gaining access to a spaceship to come back to earth. An overview of the scenes and obstacles, along with the interaction is summarized in the tables below.

Sr. No.	Scene Name	Obstacles	Interactions
1.	UI Scene	UI buttons on canvas	Using laser pointer
2.	Tutorial Scene	Combination of basic controls	
3.	Shooting Scene	Moving Aliens 3D Game Objects	Picking up Laser Gun, trigger button
4.	Enter Scene	Moving platforms with Teleport Area	Teleportation
5.	Laser Scene	Combination of rotators, lasers, pendulums and falling platforms	Jumping and crouching
6.	Lever Scene	Button to open door, Moving levers to match pattern	Button and lever
7.	Explosion Scene	Destructible obstacles blocking the path	Grabbing and Throwing Explosives using trigger
8.	Climbing Scene	Ladder leading to door	Climbing

The detailed flow of the scenes is as follows:

1. **UI Scene**

   This is the first scene of the game where the player is given the option to either go to the tutorial or play the game. This is achieved by adding UI button game objects to the scene and using the UI interaction described above.

2. **Tutorial Scene**

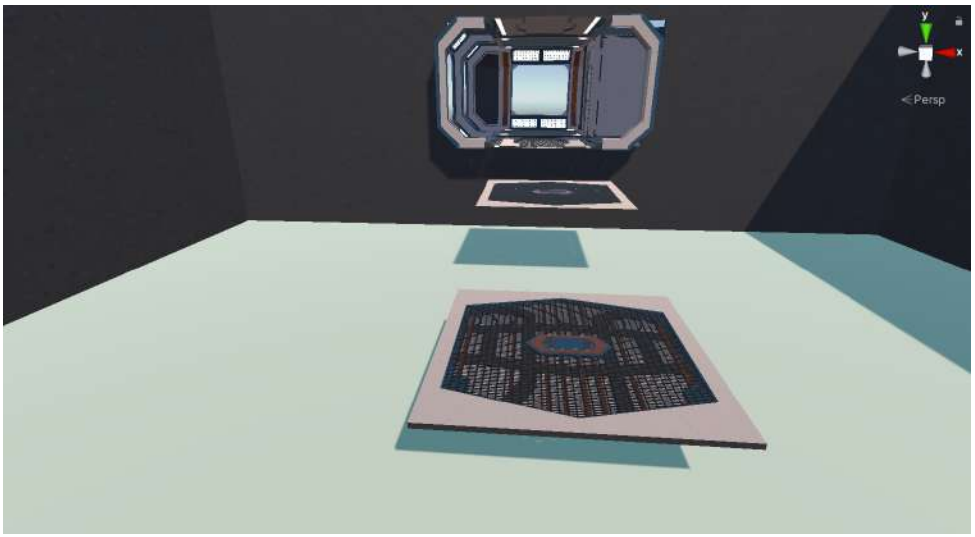
   This is a very simple scene which introduces the player to the basics of the controllers and what input maps to what action. This interactive tutorial eases the player into the game and gives an opportunity to try major actions before getting into the game.

3. **Shooting Scene**

   This is a scene whereby the player will find himself deserted in an unknown planet with aliens moving towards the player, they need to defend himself/herself by picking up the gun and shoot down the aliens in order to enter the spaceship to proceed with the next stage.

4. **Enter Scene**

   This is the first scene upon the entry of the player into the spaceship. Here the player can see a combination of moving and stationary platforms with a teleport area component attached to them. The player has to strategically teleport himself across them without falling or touching the green liquid on the floor as it will restart the scene. At the end of the scene, when the player comes to the door it opens and he moves on to the next scene.



5. **Laser Scene**

   This scene focuses on the player's ability to maneuver around moving obstacles and avoid colliding with them, which will restart the scene. The obstacles include rotators, laser beams, pendulums and falling platforms. The player times the jumps to avoid the rotators, jumps and crouches to go around the lasers, times movement to avoid pendulum and jumps from one platform to another before they crumble and he falls. In the end, he reaches a moving platform which takes him to the door leading to the next obstacle.

6. **Lever Scene**

   The player enters a small enclosed space with a closed door. The door opens on pushing the button placed on top of a column. The player then enters a room with 3 levers. The player has to move the levers to match the pattern of the picture placed on the wall to proceed to the next scene. This scene requires elements of accurate physical interaction as well as puzzle solving skills

7. **Grab and Throw Scene**

   This scene focuses on the grab and throw interaction, where the player's objective is to progress by throwing an explosive at destructible objects that are obstructing their path. To achieve this, we have provided the player with an explosive object that they can grab onto and throw onto the destructible objects in order to clear the path.

   In this particular scene, we have utilized the explosion mechanic as a puzzle-solving element, forcing the player to think creatively and strategize how to use the explosive object to clear the path. This implementation not only provides a unique gameplay experience but also adds an extra layer of interactivity and engagement for the player. The use of the explosion mechanic in this game scene provides an enjoyable and challenging gameplay element that enhances the player's overall experience.

8. **Climbing Scene**

   This scene features the last obstacle of the game i.e. Climbing. The player is in a room with a ladder. The player has to grab the rungs of the ladder to climb up and enter the control room above.

9. **End Scene**

   The end scene presents the player with an array of computers and controls. The player is greeted with a message saying the mission is complete and it is time to fly back home.

Mechanics for Changing Scene

The scene change is initiated by the basic script (can be seen in appendix) which works by detecting when the Player enters the last floor of the current scene. Upon this event, the new scene is loaded. The scenes have consistent aesthetics and design by usage of the same assets to give a seamless experience to the user. THere is also a door animation when the user ends a scene to provide an indication that the next room will now be entered.
This is achieved by detecting the distance between the door and player. When it is within the range of detection then the door opens and the player moves ahead to the floor which initiates scene change. This is done by adding a collider to the floor and enabling the trigger. When the Player tagged as “player” enters the floor, the collider detects it and loads the new scene.

User Experience

1. Vibration feedback
2. Tutorial
3. Restarting the current scene on collision
4. Seamless scene switching and door animations

Future Work/ Modifications

1. Countdown timer - Set time needed to clear the game to go back to Earth.
2. Best Record - Record the best timing to clear the game.
3. Health point systems can be incorporated as well. So the player loses the game if time runs out or if health becomes zero.
4. Tougher levels with more complicated obstacles and advanced interactions. For instance, the ability to control speed of movement, swinging interaction and more complicated pattern matching involving equations.

Appendix

Link to presentation:
https://www.canva.com/design/DAFbS7FKfT4/PhQoSyKhnCejGLZygo2R0A/view?utm_content=DAFbS7FKfT4&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Link to complete demo video on YouTube
https://youtu.be/fXW90JKVBcc

Screenshots of some important scripts used in the project

```
public class GameOver : MonoBehaviour
{
    // public TextMeshProUGUI gameOverText; // The Text UI component that displays the "game over" message
    // public Canvas canvas;

    // [SerializeField]
    //  Camera mainCamera;
    public GameObject gameOverUI;
    private bool isGameOver = false;

    public AudioClip gamesound;
    public AudioClip gameOversound;

    private AudioSource audioSource;

   private void Start()
    {
                gameOverUI.SetActive(false);
                audioSource = GetComponent<AudioSource>();
                audioSource.clip = gamesound;
                audioSource.Play();

    }

  private void OnTriggerEnter(Collider other)
    {
        // Check if the collider that entered the trigger is the game object we want to detect
        if (other.gameObject.tag == "Obstacles")
        {
            isGameOver = true;
            audioSource.Stop();

        }
    }

    private void RestartScene()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }



    private void Update()
    {
        if (isGameOver)
        {
            Debug.Log("Collision detected");
            gameOverUI.SetActive(true);
            audioSource.clip = gameOversound;
            audioSource.Play();
            Invoke("RestartScene", 3f);


        }
    }
}
```

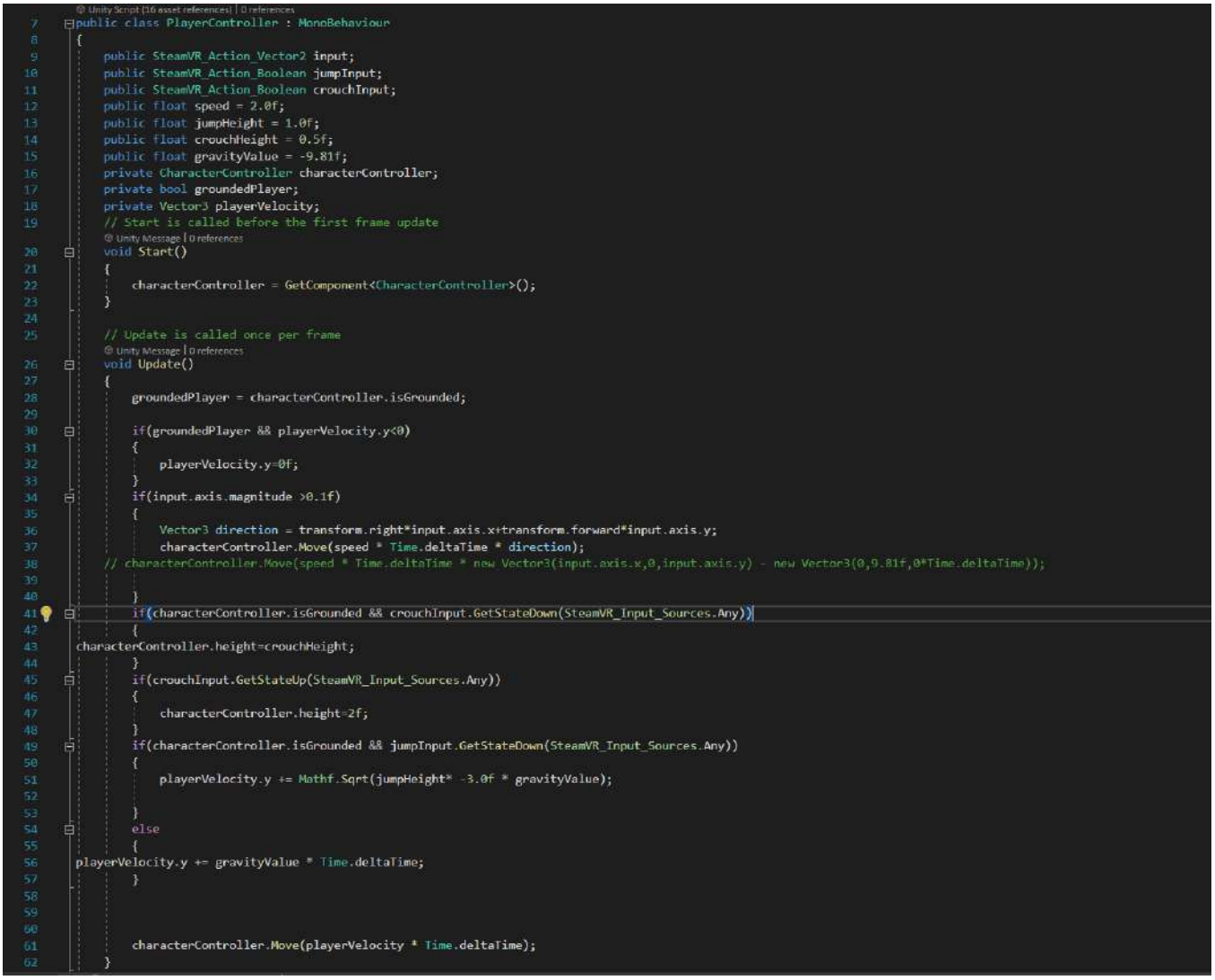
Script attached to player when the player fails to destroy all aliens and comes into contact with the player to initate game over and restart the scene. AudioSource component is alsoadded to the player for the background music.

```
public class AlienSpawner : MonoBehaviour
{
    [Header("Size of the spawner area")]
    7 references
    public Vector3 spawnerSize;

    [Header("Rate of spawn")]
    1 reference
    public float spawnRate = 1f;

    [Header("Model to spawn")]
    1 reference
    [SerializeField] private GameObject AlienModel;

    3 references
    private float spawnTimer = 0f;

    0 references
    private void OnDrawGizmos()
    {
        Gizmos.color = new Color(0, 1, 0, 0.5f);
        Gizmos.DrawCube(transform.position, spawnerSize);
    }

    3 references
    int count = 0;

    0 references
    private void Update()
    {
        spawnTimer += Time.deltaTime;

        if(spawnTimer > spawnRate)
        {
            spawnTimer = 0;
            SpawnAliens();
            count += 1;
            Debug.Log(count);
        }

        if(count >= 5)
        {
            enabled = false;
            return;
        }
    }

    1 reference
    private void SpawnAliens()
    {
        Vector3 spawnPoint = transform.position + new Vector3(UnityEngine.Random.Range(-spawnerSize.x/2, spawnerSize.x/2),
                                                              UnityEngine.Random.Range(-spawnerSize.y/2, spawnerSize.y/2),
                                                              UnityEngine.Random.Range(-spawnerSize.z/2, spawnerSize.z/2));

        GameObject alien = Instantiate(AlienModel, spawnPoint, transform.rotation);

        alien.transform.SetParent(this.transform);
    }
}
```

Script for spawning the Aliens

```
public class AlienMovement : MonoBehaviour
{
    1 reference
    public Transform playerTransform;

    [Header("MinSpeed")]
    1 reference
    public float minSpeed = 5f;
    [Header("MaxSpeed")]
    1 reference
    public float maxSpeed = 10f;
    2 references
    private float speed;
    3 references
    private float timer;

    [Header("rotationSpeed")]
    1 reference
    public float rotationSpeed = 2f;

    2 references
    private Rigidbody obstacleRigidbody;
    0 references
    void Start()
    {
        obstacleRigidbody = GetComponent<Rigidbody>();
    }

    0 references
    void Update()
    {
        Vector3 direction = playerTransform.position - transform.position;
        // Debug.Log(direction);
        speed = Random.Range(minSpeed,maxSpeed);
        obstacleRigidbody.velocity = direction * Time.deltaTime * speed;
        // Debug.Log(speed);
        timer += Time.deltaTime;

        if (timer >= rotationSpeed)
        {
            transform.rotation = Quaternion.Euler(0, Random.Range(0, 360), 0);
            timer = 0.0f;
        }

    }
}
```

Script for the Aliens to move towards the player



```
Unity Script (16 asset references) | 0 references
public class PlayerController : MonoBehaviour
{
    public SteamVR_Action_Vector2 input;
    public SteamVR_Action_Boolean jumpInput;
    public SteamVR_Action_Boolean crouchInput;
    public float speed = 2.0f;
    public float jumpHeight = 1.0f;
    public float crouchHeight = 0.5f;
    public float gravityValue = -9.81f;
    private CharacterController characterController;
    private bool groundedPlayer;
    private Vector3 playerVelocity;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        characterController = GetComponent<CharacterController>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        groundedPlayer = characterController.isGrounded;

        if(groundedPlayer && playerVelocity.y<0)
        {
            playerVelocity.y=0f;
        }
        if(input.axis.magnitude >0.1f)
        {
            Vector3 direction = transform.right*input.axis.x+transform.forward*input.axis.y;
            characterController.Move(speed * Time.deltaTime * direction);
    // characterController.Move(speed * Time.deltaTime * new Vector3(input.axis.x,0,input.axis.y) - new Vector3(0,9.81f,0*Time.deltaTime));

        }
        if(characterController.isGrounded && crouchInput.GetStateDown(SteamVR_Input_Sources.Any))
        {
characterController.height=crouchHeight;
        }
        if(crouchInput.GetStateUp(SteamVR_Input_Sources.Any))
        {
            characterController.height=2f;
        }
        if(characterController.isGrounded && jumpInput.GetStateDown(SteamVR_Input_Sources.Any))
        {
            playerVelocity.y += Mathf.Sqrt(jumpHeight* -3.0f * gravityValue);

        }
        else
        {
playerVelocity.y += gravityValue * Time.deltaTime;
        }



        characterController.Move(playerVelocity * Time.deltaTime);
    }
```

Script for player movement using touchpad and jumping crouching buttons

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
Unity Script (1 asset reference) | 0 references
public class SceneScene : MonoBehaviour
{
    public string sceneName;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }

    Unity Message | 0 references
    public void OnTriggerEnter(Collider other)
    {
        Debug.Log("loading scene");
        SceneManager.LoadScene(sceneName);
    }
}
```

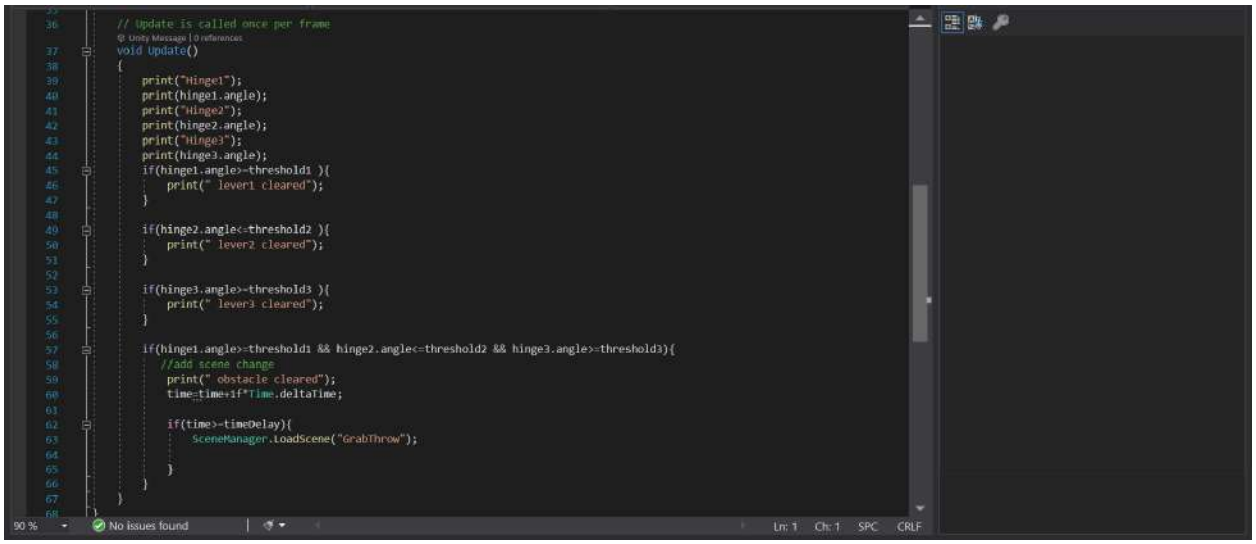
Script for changing scene when the player reaches the last floor of the current scene

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script (2 asset references) | 0 references
public class BasicDoor : MonoBehaviour
{
    // Start is called before the first frame update
    public float detectionRadius = 2.5f;
    public Transform playerTransform;
    private Animator doorAnimation;
    Unity Message | 0 references
    void Start()
    {
        doorAnimation = GetComponent<Animator>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (Vector3.Distance(transform.position, playerTransform.position) < detectionRadius)
        {
            print("reached door");
            doorAnimation.Play("Base Layer.glass_door_open",0,-1);
        }
    }
}
```

Initiate door opening animation when the player reaches close to the door

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script | 0 references
public class PosCheck : MonoBehaviour
{

    GameObject lever1;
    GameObject lever2;
    GameObject lever3;
    HingeJoint hinge1;
    HingeJoint hinge2;
    HingeJoint hinge3;
    public float threshold1=44.5f;
    public float threshold2=-44.5f;
    public float threshold3=44.5f;
    float time;
    float timeDelay;


    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        lever1=GameObject.Find("Lever_1");
        lever2=GameObject.Find("Lever_2");
        lever3=GameObject.Find("Lever_3");
        hinge1= lever1.GetComponent<HingeJoint>();
        hinge2= lever2.GetComponent<HingeJoint>();
        hinge3= lever3.GetComponent<HingeJoint>();
        time=0f;
        timeDelay=2f;
```

Script for initializing lever scene



```
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        print("Hinge1");
        print(hinge1.angle);
        print("Hinge2");
        print(hinge2.angle);
        print("Hinge3");
        print(hinge3.angle);
        if(hinge1.angle>=threshold1 ){
            print(" lever1 cleared");
        }

        if(hinge2.angle<=threshold2 ){
            print(" lever2 cleared");
        }

        if(hinge3.angle>=threshold3 ){
            print(" lever3 cleared");
        }

        if(hinge1.angle>=threshold1 && hinge2.angle<=threshold2 && hinge3.angle>=threshold3){
          //add scene change
          print(" obstacle cleared");
          time=time+1f*Time.deltaTime;

          if(time>=timeDelay){
              SceneManager.LoadScene("GrabThrow");

          }
        }
    }
}
```

90 % | No issues found | Ln: 1 Ch: 1 SPC CRLF

Script for checking if levers are in correct position

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Explosive : MonoBehaviour
{
    // [SerializeField] private GameObject obstacleExplosion;

    public float delay = 3f;
    public float radius = 5f;
    public float force = 5f;

    public GameObject explosionEffect;
    // public GameObject obstacle;
    // public ParticleSystem part;

    float countdown;
    bool hasExploded = false;

    // Start is called before the first frame update
    void Start(){
        countdown = delay;
        // part = obstacle.GetComponent<ParticleSystem>();

    }

    // Update is called once per frame
    void Update(){
        countdown -= Time.deltaTime;
        if (countdown <= 0f && !hasExploded){
            Explode();
            hasExploded = true;
        }
    }

    void Explode(){
        //Instantiate(explosionEffect, transform.position, transform.rotation);

        Collider[] colliders = Physics.OverlapSphere(transform.position, radius);

        foreach (Collider nearbyObject in colliders){
            Rigidbody rb = nearbyObject.GetComponent<Rigidbody>();
            if (rb != null && nearbyObject.gameObject.CompareTag("Obstacle")){
                rb.AddExplosionForce(force, transform.position, radius);

                Instantiate(explosionEffect, rb.transform.position, rb.transform.rotation);
                // part.Play();
                // Destroy(nearbyObject.gameObject, part.main.duration);
                Destroy(nearbyObject.gameObject);
            }
        }

        Destroy(gameObject);
    }
}
```

Script that enables a throwable object to explode

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Valve.VR;


public class ControlClimb : MonoBehaviour
{
    public SteamVR_Action_Boolean climbAction;
    public float climbSpeed = 1.0f;
    private bool isClimbing = false;
    private Transform climbTarget = null;

    // Start is called before the first frame update
    void Start()
    {
        //climbAction = SteamVR_Input.GetAction<SteamVR_Action_Boolean>("default", "Climb");
    }



    private void StartClimbing() {
        RaycastHit hit;
        if (Physics.Raycast(transform.position, transform.forward, out hit, 2.0f)) {
            if (hit.transform.CompareTag("Climbable")) {
                isClimbing = true;
                climbTarget = hit.transform;
            }
        }
    }
```

Script for initializing Climb Scene

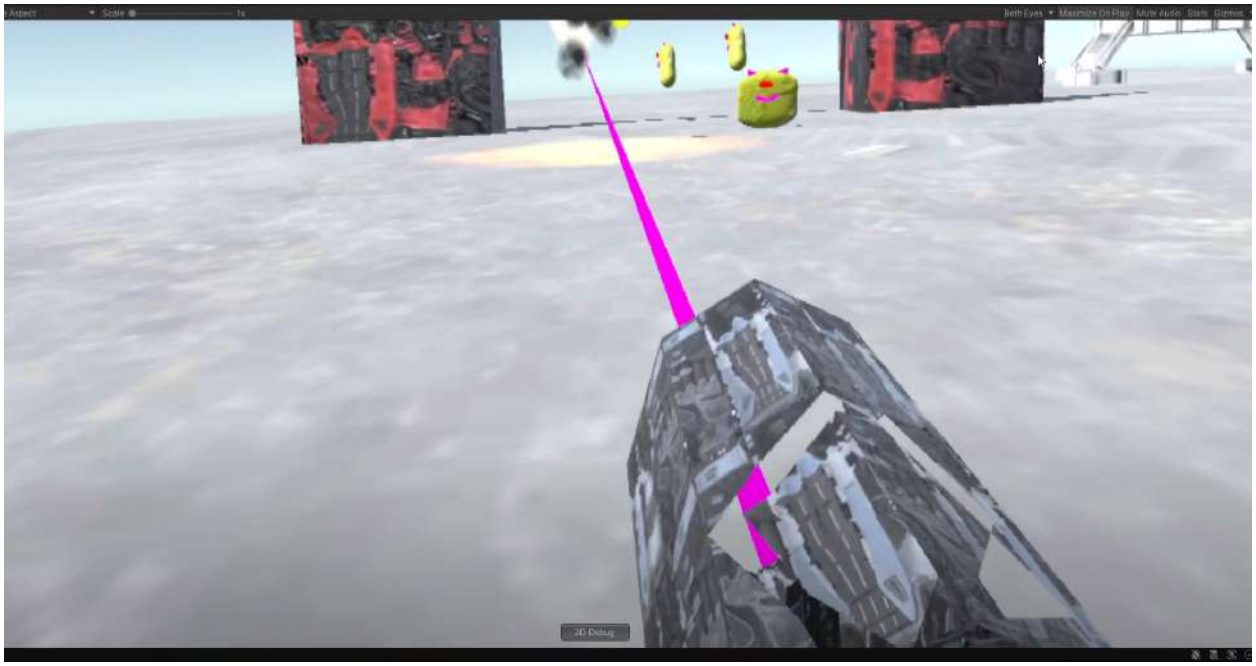
```

    private void StopClimbing() {
        isClimbing = false;
        climbTarget = null;
    }



    // Update is called once per frame
    void Update()
    {
        if (climbAction.GetStateDown(SteamVR_Input_Sources.LeftHand) || climbAction.GetStateDown(SteamVR_Input_Sources.RightHand)) {
            StartClimbing();
        } else if (climbAction.GetStateUp(SteamVR_Input_Sources.LeftHand) || climbAction.GetStateUp(SteamVR_Input_Sources.RightHand)) {
            StopClimbing();
        }

        if (isClimbing) {
            Vector3 climbDirection = climbTarget.position - transform.position;
            climbDirection.y = 0;
            climbDirection.Normalize();
            transform.position += climbDirection * climbSpeed * Time.deltaTime;
        }

    }


}
```

Script for checking if player is grabbing onto a ‘Climbable’ object and accordingly moving player up the ladder

Scenes as scene in Game Mode

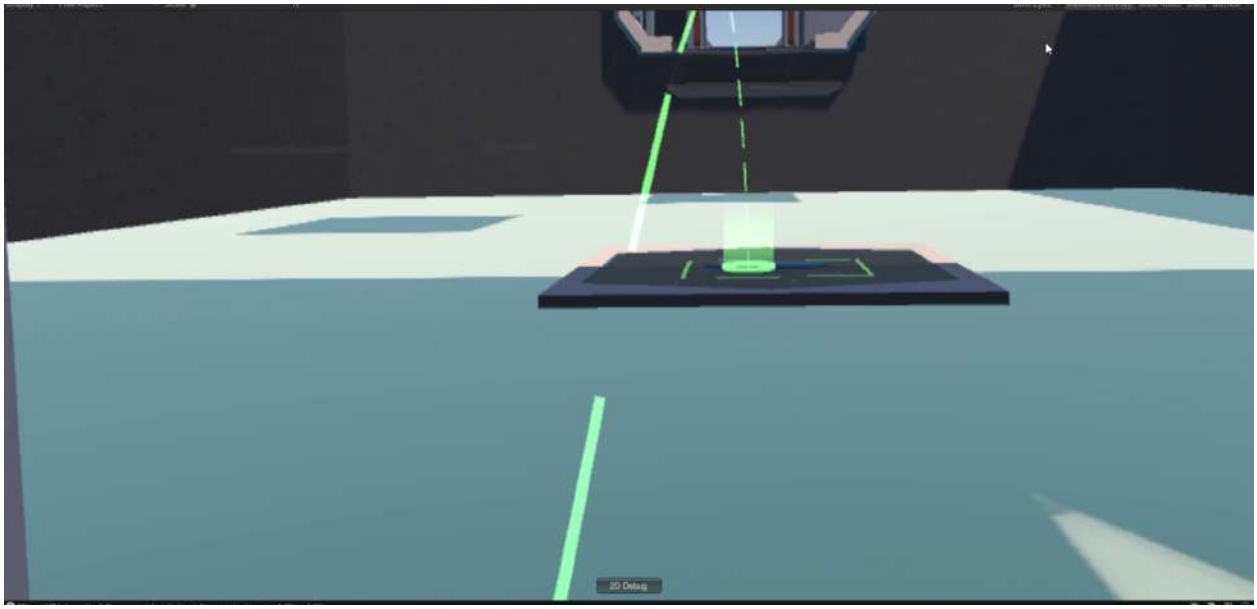
Shooting Scene

Interaction with Laser Gun object



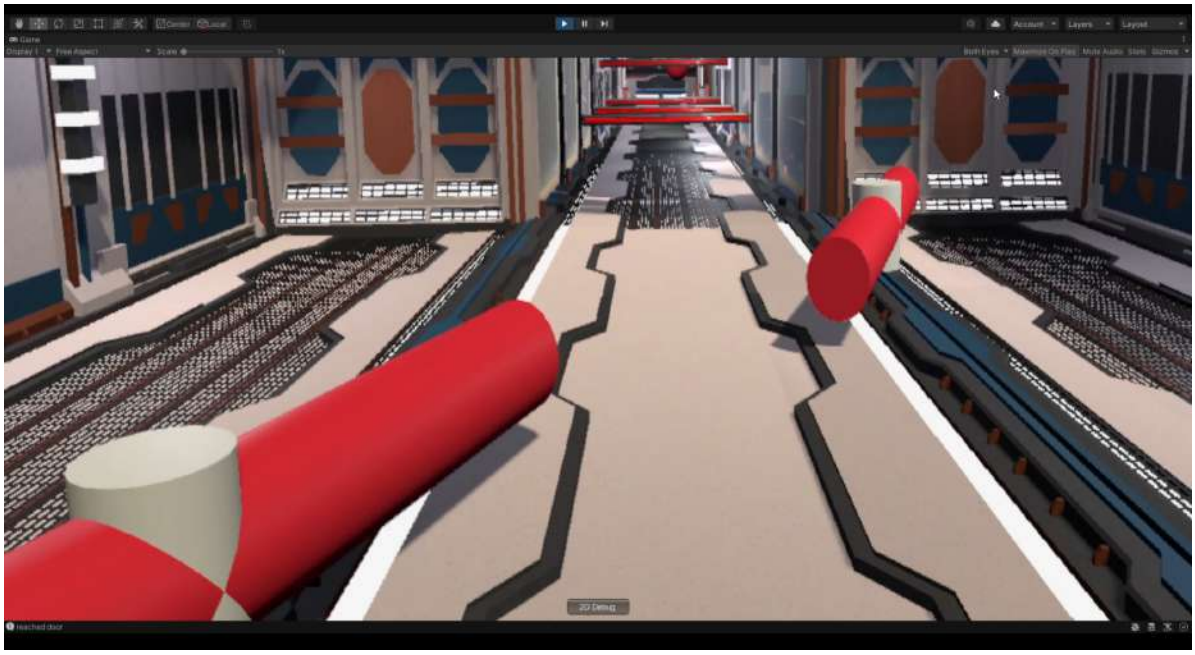
Aliens getting destroyed when in contact with Line Renderer Component and initiate explosion particle system.

Teleportation Scene

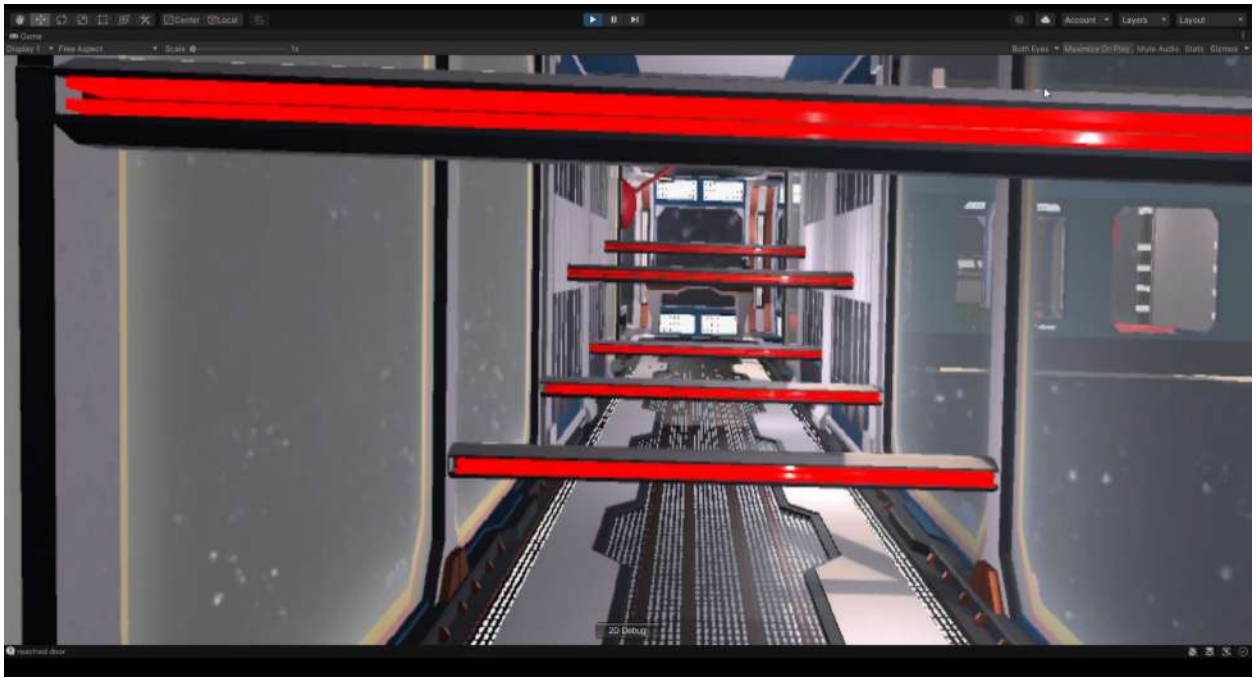


Moving Platforms

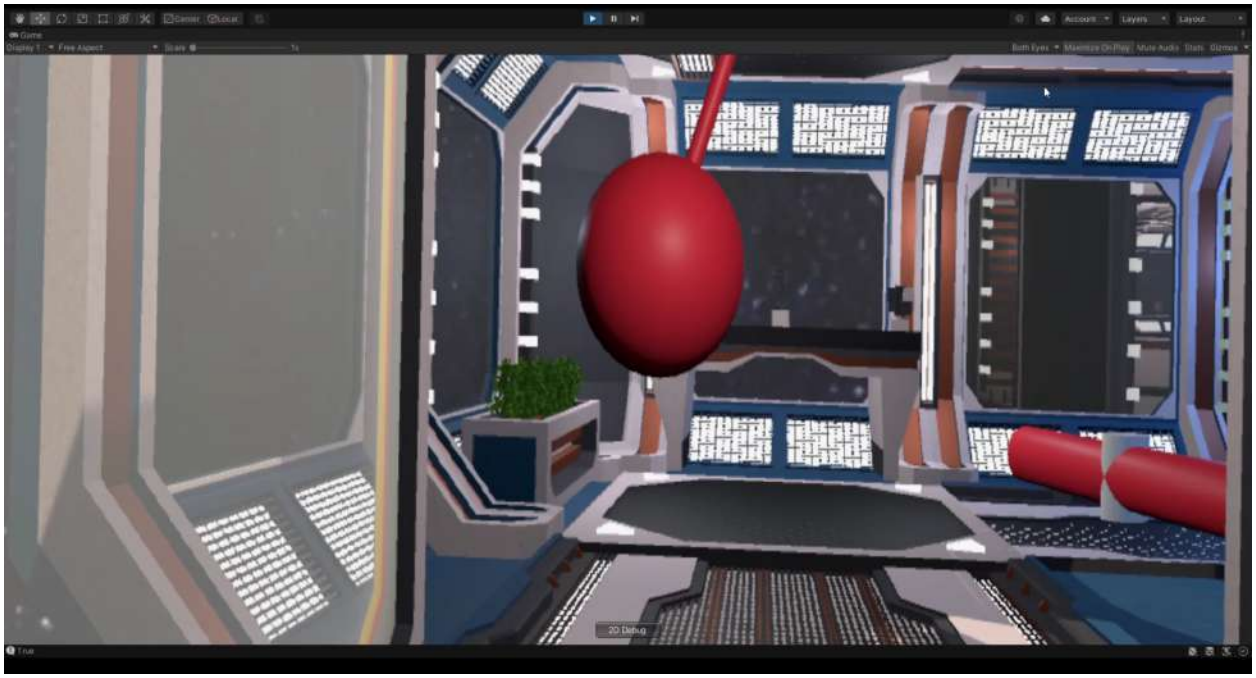
Laser Scene



Rotators with collision



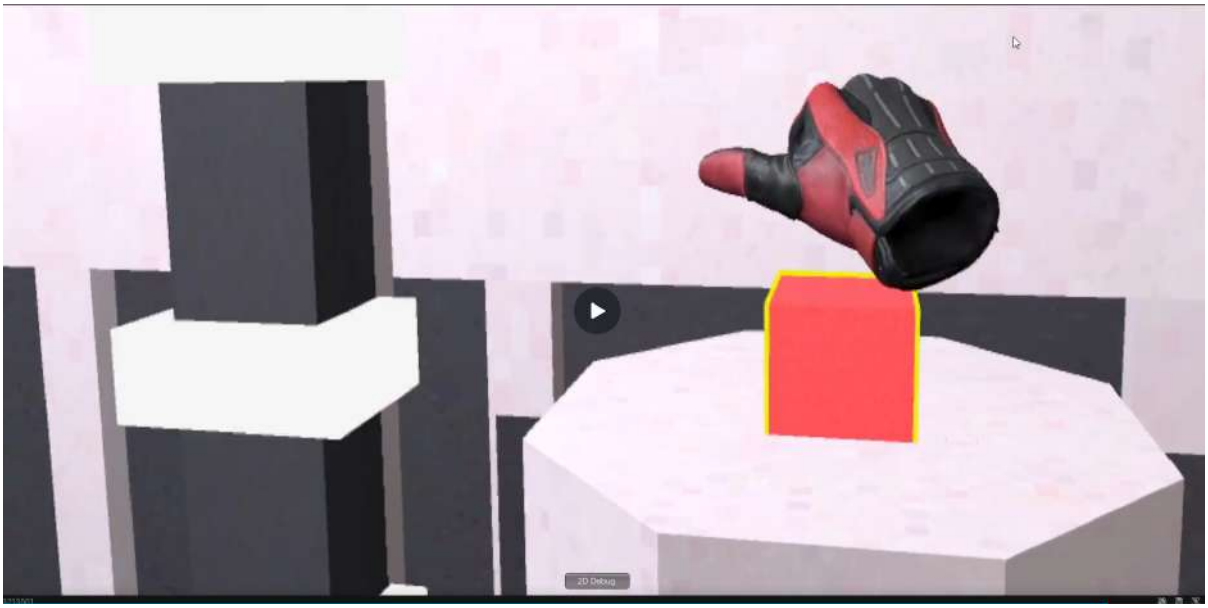
Laser beams with collision



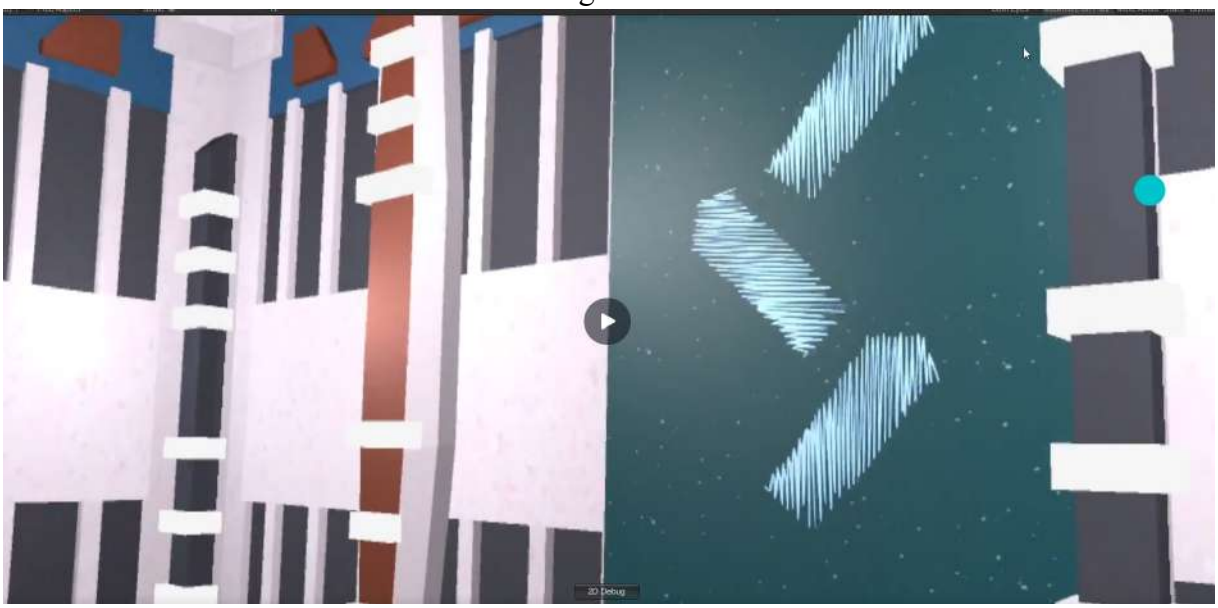
Pendulum with collision

Falling platforms (that crumble a few seconds after contact with player)

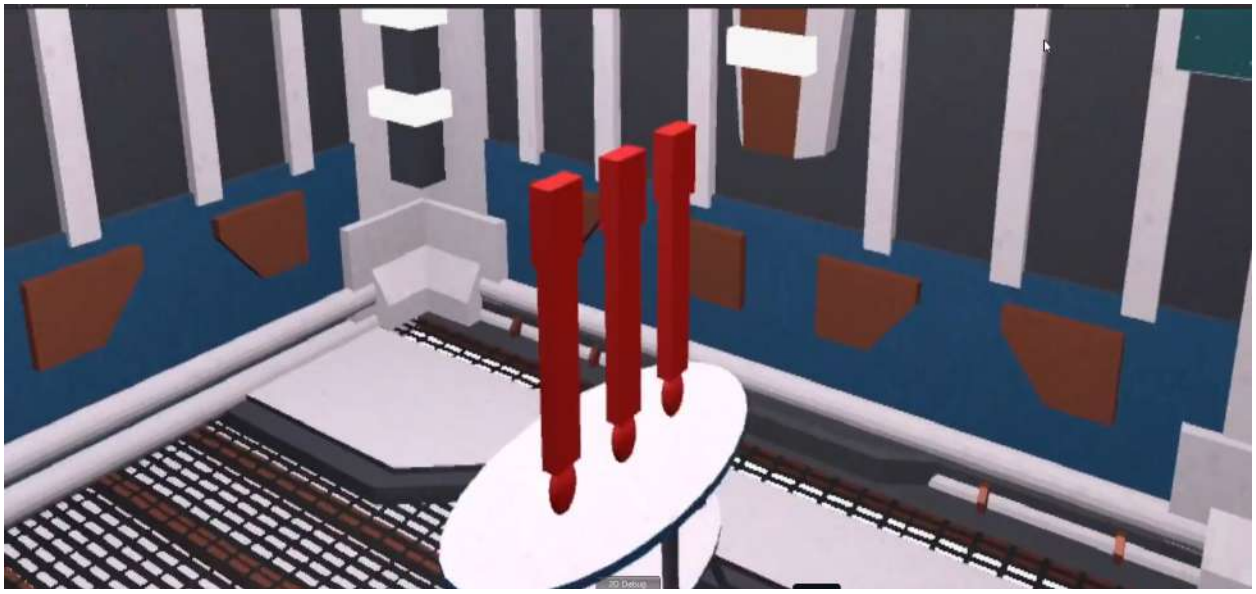
Lever Scene



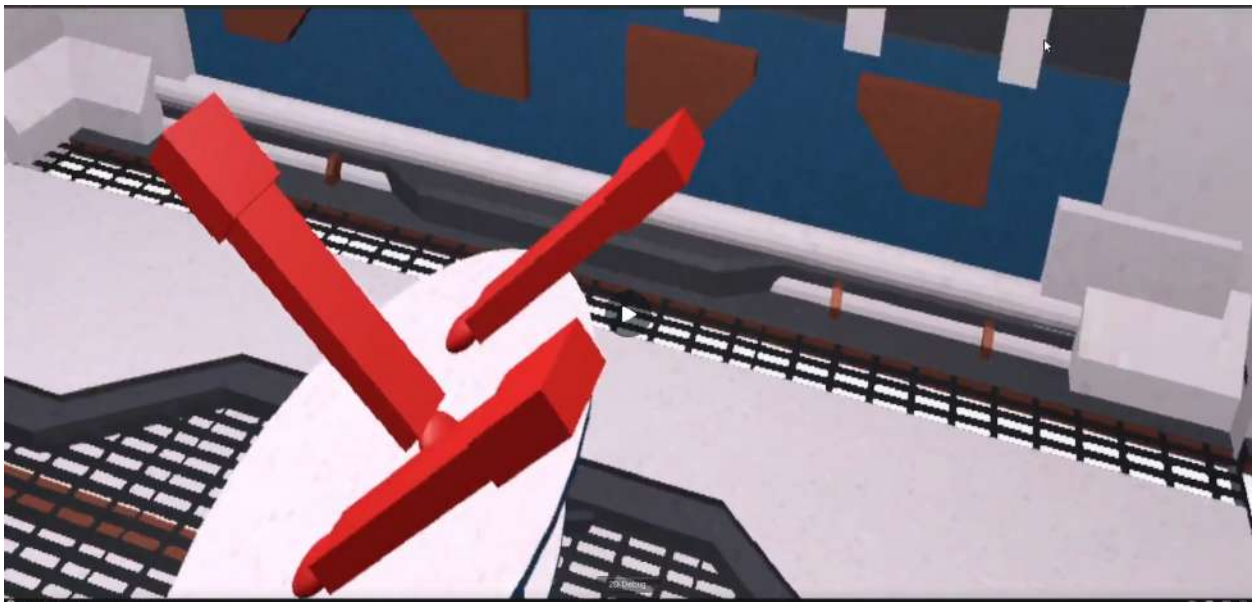
Pressing the button



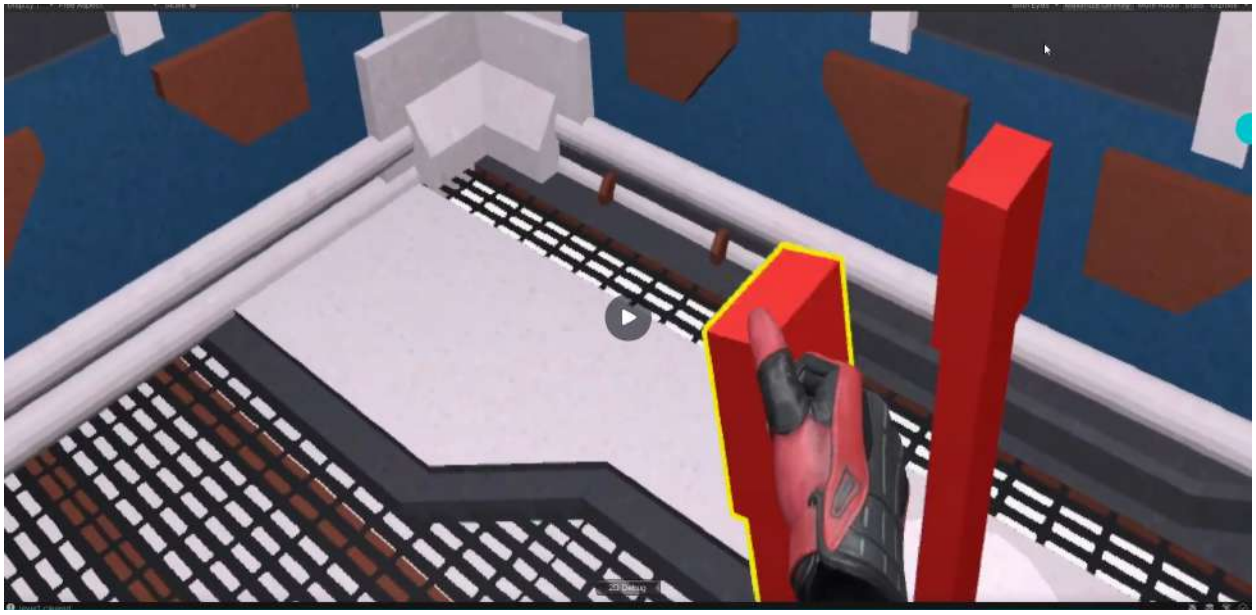
Picture that serves as hint for lever



Initial position of levers

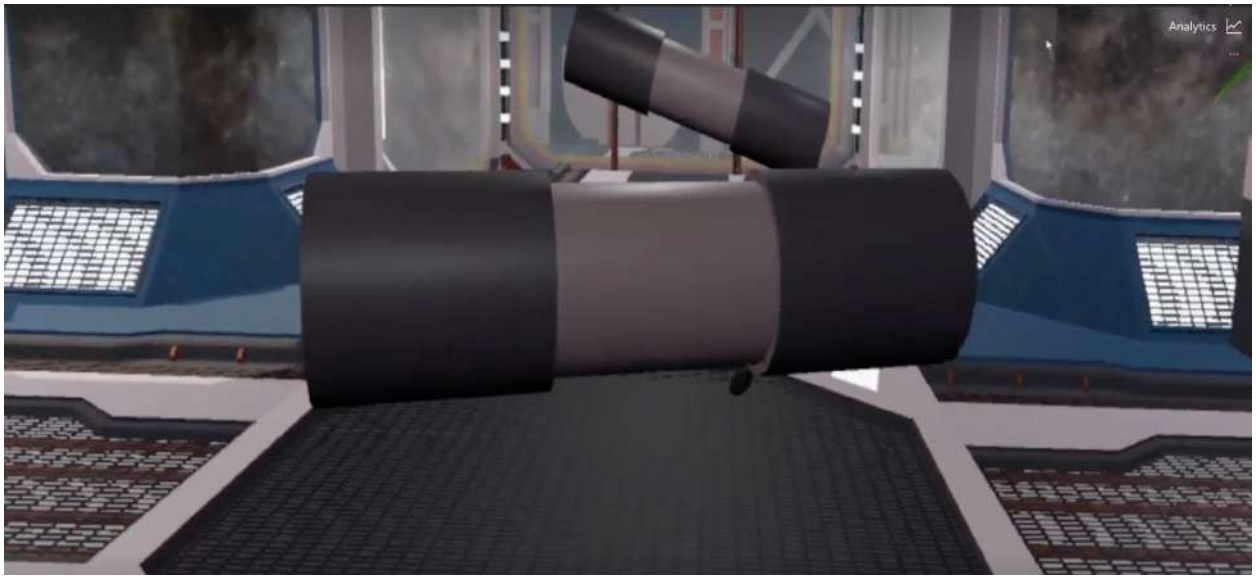


Final position of levers



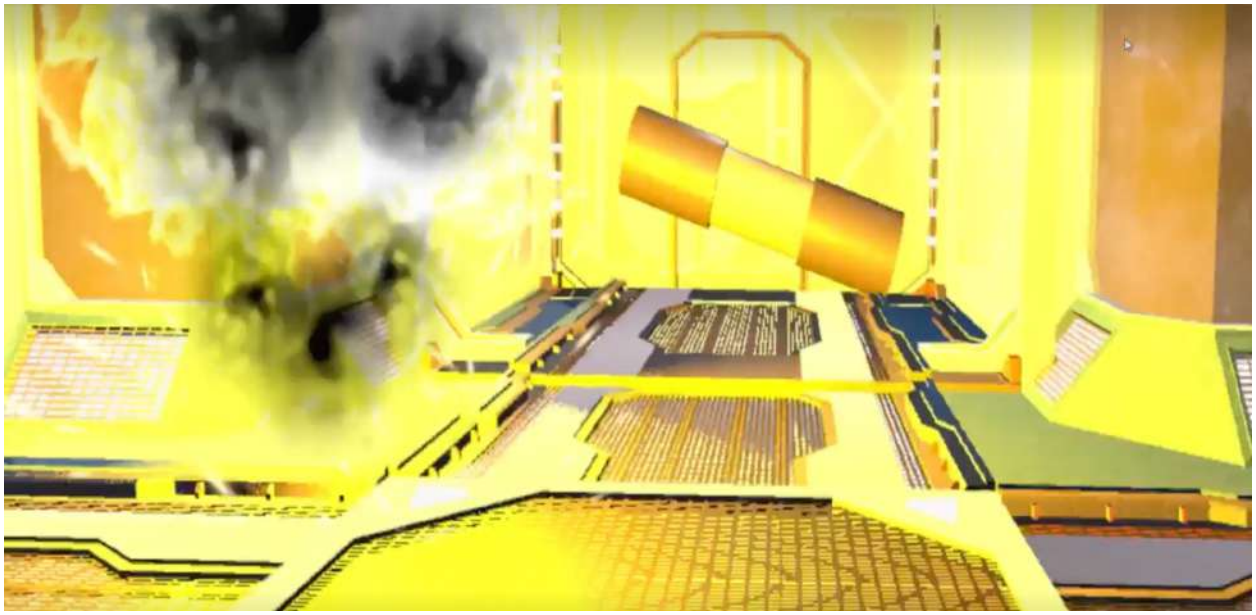
Pushing the lever

Explosion Scene



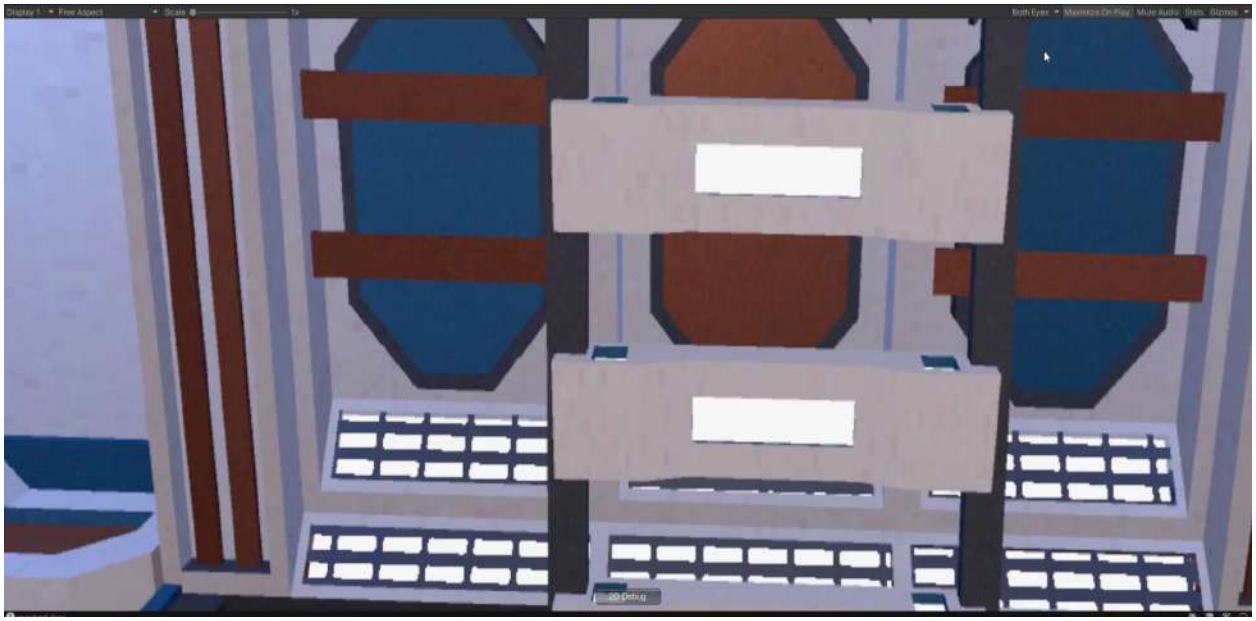
Obstacles blocking the path

Grabbing and Throwing Explosive Object

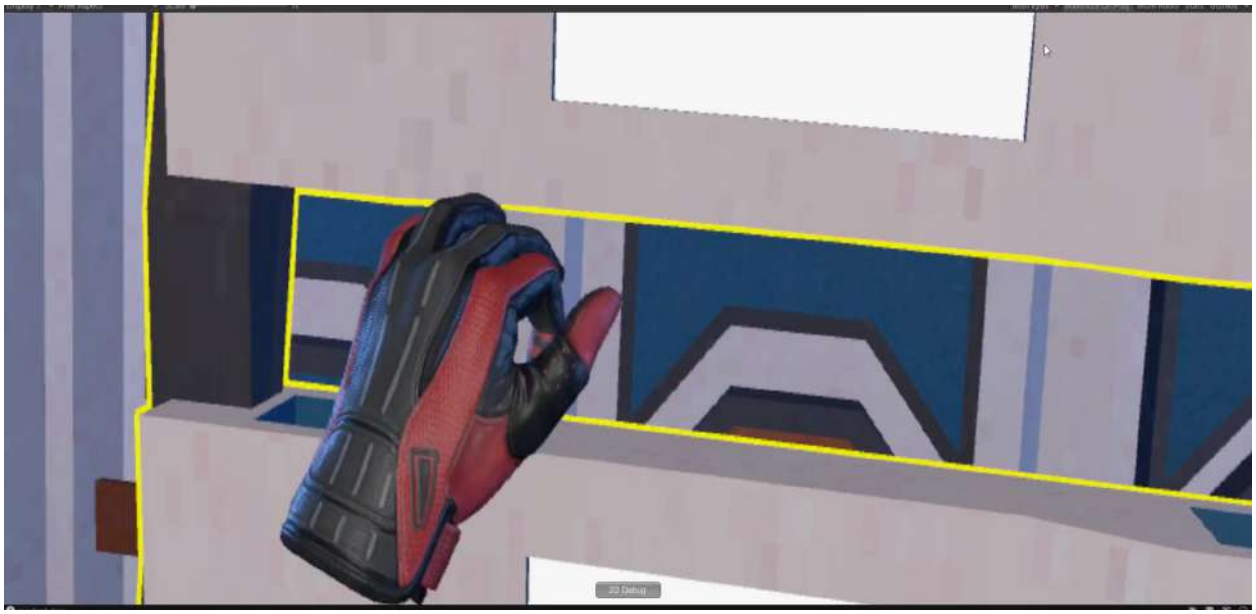


Obstacles Exploding

Climbing Scene



Ladder for climbing scene



Hand grabbing onto ladder so that player can move up

End Scene

Mission Complete message in End Scene

References

Input System | SteamVR Unity Plugin. (n.d.). https://valvesoftware.github.io/steamvr_unity_plugin/articles/SteamVR-Input.html

Najera, J. (2021, December 10). *SteamVR 2.0 Tutorial: Laser Pointer - Jesus Najera*. Medium. https://setzeus.medium.com/tutorial-steamvr-2-0-laser-pointer-bbc816ebeec5

S. (2019, November 14). *VR Buttons in Unity*. SLIDEFACTORY | Solutions for Mobile, Web, and the Metaverse. (VR, AR, Web3). https://theslidefactory.com/vr-buttons-in-unity/

Technologies, U. (n.d.). *Unity - Manual: Character Controller component reference*. https://docs.unity3d.com/Manual/class-CharacterController.html

Tvtig. (2021, August 4). *How To Create An Explosion Effect In Unity* [Video]. YouTube. https://www.youtube.com/watch?v=cvQiQglPI18