Technical Document

**Proposal:**

The game will start with a beginning menu where the user chooses to start the game or quit. The idea is to make a portal esque game where the user can fire a “portal gun” at walls in order to solve a series of platforming puzzles to reach the next level. This portal gun fires a pair of “Wormholes” that connect the ends of one portal to another for use in the puzzles. These portals will make the user believe that they are in fact connecting two parts of the level, allowing them to use these portals in a variety of ways such as falling through one portal to launch themselves out the other, retaining their speed when either you or an object travels through it. The game will also implement the concept of having objects partially inside the portal, therefore requiring the user to “cut” the object in order to represent this on both sides of the portal. Once the user fires a third portal, the first portal fired will be despawned and the second portal fired will stay and be used as a new entryway. The mechanics of the gun will be similar to that as in a first person shooter, where when the object fired from the portal gun collides with a flat surface of some kind, it will create the portal. The player will also be able to pick up various objects around the scene by using a button on the keyboard. The layout of the game will be 5 full levels consisting of large rooms with various hazards, objects, and platforming segments that the user has to traverse in order to get across the room and enter the exit room that often will require the user to fulfill some kind of condition, such as pressing a button or placing a cube on a pressure plate before opening. Some of these hazards will include bottomless pits and a wall that only activates during a specific time period. If a player falls victim to one of these hazards, the user will “die” and be respawned at the beginning of the level and have to repeat the processes. There will be no fall damage in the game and the room will be encompassed by an invisible wall in order to prevent the player from leaving the room or going places unintended. At the end of the levels, the user will be given a game complete screen. The player will also have unlimited attempts at the puzzles meaning there will be no real way of creating a “Game Over” besides quitting via the in-game pause menu. The pause menu will also have a restart button to manually restart the level if the user chooses to do so. The games in game interface will also feature a visual indicator showing how many portals have been fired, and potentially include a visual difference between the two portals.

**Technical Implementations:**

Standard features:

1. Setup of 3D Virtual Content(Models, sounds, etc.)

2. GUI

3. Startup and In-game menu(Pause, Options button, and quit game button)

4. First person shooting for the portal gun onto various walls throughout the room to create the portals.

5. Hazards(bottomless pits)

6. Implementation of game logic

7. 5 Complete levels with significant differences between them

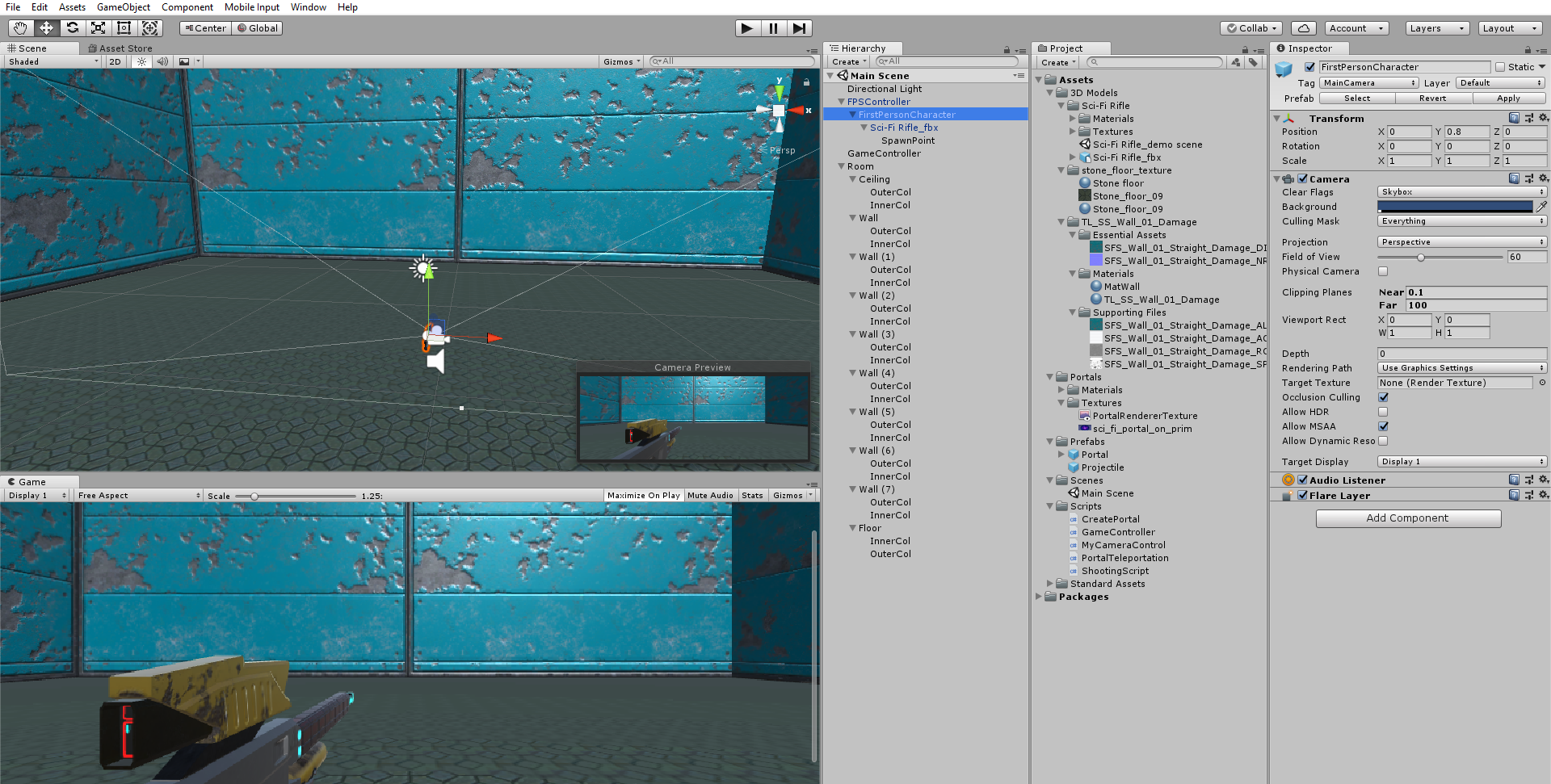
Advanced features:

1. Portal implementation (part 1): Visual effects

2. Portal implementation (part 2): game mechanics

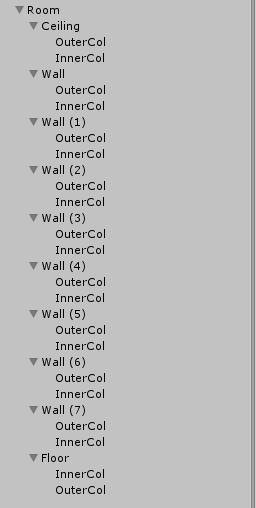
3. Hazards(Timing activated walls)

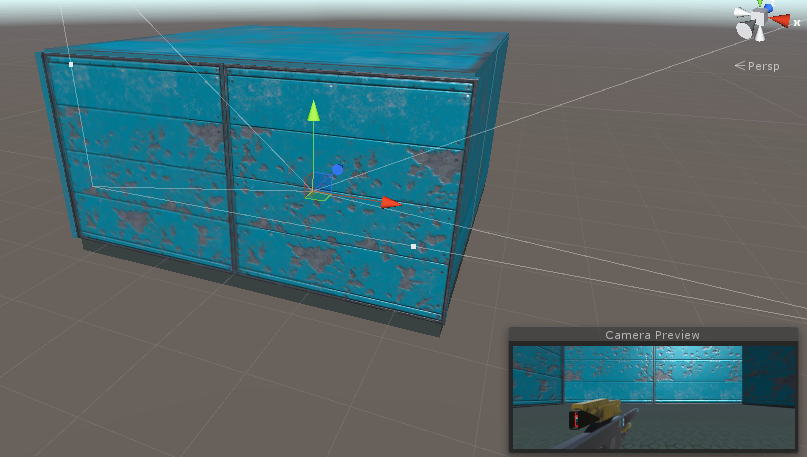
**Current Progress:**



**Figure 1: Current File Layout**

Currently the game is roughly 45% complete. As it stands, The main portal mechanic and firing mechanics have been implemented and the user is able to teleport between portals. The portals also are limited to two portals as intended. Prep work has also been done for the advanced feature of portal visual effects, as you can see with the render texture and cameras. The first thing that I set up was the FPS controller, as seen in the file structure. The FPS controller also comes with extra things such as audio for walking, and its own rigidbody in order to replicate a player character. The second thing I did was set up the scene.





**Figure 2: Current Scene**

The scene is currently very basic with four walls encompassing the main camera and a weapon underneath the main camera. Each wall has a pair of game objects attached to it referred to as “InnerCol” and “OuterCol”. These game objects have colliders attached to them in order to determine where a portal should be able to spawn and shouldn’t be able to spawn. Basically, if a portal is too close to a wall, the portal will hit the “OuterCol” and not spawn. In front of the weapon is an empty game object called SpawnPoint that is used later to give the position where the projectiles will be instantiated. The current setup of the walls is 50 feet by 50 feet, and the scene is designed to be temporary until the full level is designed. The next item I worked on was scripting of the weapon in order to fire projectiles. The weapon has a script attached to it called “Shooting Script” in order to allow the weapon to fire spheres with force out from the gun barrel.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ShootingScript : MonoBehaviour

{

public GameObject projectilePrefab = null; // Reference variable to the cannon ball prefab.

public Transform spawnLocation = null; // Reference variable to the Transform component of the spawn location.

public float projectileVelocity = 25.0F;

// Use this for initialization

void Start ()

{

}

// Update is called once per frame

void Update ()

{

if(Input.GetMouseButtonDown(0))

{

GameObject projectileClone = null;

Vector3 forceDir = spawnLocation.TransformDirection(Vector3.forward);

Quaternion rot = Quaternion.FromToRotation(Vector3.forward, forceDir);

projectileClone = Instantiate(projectilePrefab, spawnLocation.position, rot);

//Set Game object to active

projectileClone.SetActive(true);

Rigidbody projectileRigidbody = projectileClone.GetComponent<Rigidbody>();

projectileRigidbody.velocity = projectileRigidbody.transform.forward \* projectileVelocity;

Destroy(projectileClone, 5.0f);

}

}

}

**Figure 3: Shooting Script**

As you can see, the script activates once the user clicks the left mouse button. Once this is done, a projectile prefab is instantiated into the spawn location that is declared beforehand in front of the weapon with the appropriate rotation. Once this is done, the projectiles rigidbody is obtained and a forward force is applied to it in order to “shoot” the projectile. The projectile is then despawned after 5 seconds. The next script I worked on was the create portal script.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CreatePortal : MonoBehaviour

{

ContactPoint contact; //The location the projectile collided with

public GameObject portalObject;

public static GameObject portalClone;

public static bool portalSpawn = false;

public float portalOffset = 0.01f;

// Use this for initialization

void Start ()

{

}

// Update is called once per frame

void Update ()

{

}

void OnCollisionEnter(Collision collision)

{

if(collision.gameObject.tag == "InnerCol")

{

contact = collision.contacts[0]; //Grab contact point and store it in the collision structure

Quaternion rot = Quaternion.FromToRotation(Vector3.back, contact.normal); //Assign a rotation to it using the normal vector

//Vector3 pos = contact.point; //Assign a vector3 position

Vector3 pos = contact.point + (contact.normal \* portalOffset);

portalClone = Instantiate(portalObject, pos, rot);//Create object

//Destroy(this.gameObject); //Destroy the bullet

portalSpawn = true;

}

Destroy(this.gameObject); //Destroy the bullet

}

void OnTriggerEnter(Collider col)

{

if (col.name == "Portal(Clone)")

{

Destroy(this.gameObject);

}

}

}

**Figure 4: CreatePortal Script**

This script is attached to the projectile and is used to create a portal in the location that the projectile collides with. As you can see, the script first determines if the object collided with is an inner collider or something else. If the projectile hits the inner collider game object, we continue with spawning the portal. From here, we grab the point of contact of the projectile, add a rotation to it, and create a position vector with an offset in the direction of the normal vector of the contact point. This is in order to prevent the portal from spawning on top of the wall. If the projectile collides with anything else, the projectile is destroyed. We then have a second collision detection where we determine if the projectile hits the portals collider, we destroy the projectile. This is in order to prevent overlapping portals. The next script I worked on was the GameController script.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameController : MonoBehaviour

{

public static GameObject portal1;

public static GameObject portal2;

public int p1 = 0;

public int p2 = 0;

public static bool twoPortals = false;

// Use this for initialization

void Start ()

{

}

// Update is called once per frame

void Update ()

{

if(CreatePortal.portalSpawn == true)

{

if(p1 == 0)

{

portal1 = CreatePortal.portalClone;

CreatePortal.portalSpawn = false;

p1++;

}

else if(p2 == 0)

{

portal2 = CreatePortal.portalClone;

CreatePortal.portalSpawn = false;

twoPortals = true;

p2++;

}

else

{

Destroy(portal1);

portal1 = portal2;

portal2 = CreatePortal.portalClone;

CreatePortal.portalSpawn = false;

}

}

}

}

**Figure 5: GameController script**

This script is where the balancing of the portals occurs currently, but will later be used for everything that needs to happen during runtime in the game. As you can see, we declare two game objects called portal1 and portal2 that are supposed to mimic the first 2 portals you fire. In the if statement, we run through assigning these game objects to the portals that are spawned using the CreatePortal script as they are spawned. Once both portals have spawned, we execute the else statement where we destroy the first portal, reassign the portal1 game object to the second portal, and reassign portal2 to the new portal that you spawned. This ensures there are always 2 portals on the map and no more, as well as ensuring the oldest portal is the portal that gets despawned. The last script I worked on was the PortalTeleportation script.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PortalTeleportation : MonoBehaviour

{

public GameObject spawnLocation;

public GameObject teleportLocationGameObject;

Rigidbody rb;

public Vector3 contactPoint;

public float offset;

public Vector3 contactNormal;

// Use this for initialization

void Start ()

{

}

// Update is called once per frame

void Update ()

{

}

void OnTriggerEnter(Collider col)

{

if(col.gameObject.name == "FPSController" && GameController.twoPortals == true)

{

rb = col.gameObject.GetComponent<Rigidbody>();

float rbForceMagnitude = rb.velocity.magnitude;

if (this.gameObject == GameController.portal1)

{

teleportLocationGameObject = GameController.portal2;

spawnLocation = teleportLocationGameObject.transform.Find("SpawnLocation").gameObject;

col.gameObject.transform.position = spawnLocation.transform.position;

Vector3 rbForceDirectionWorld (teleportLocationGameObject.transform.forward);

rb.AddForce(rbForceDirectionWorld \* rbForceMagnitude, ForceMode.Force);

}

else

{

teleportLocationGameObject = GameController.portal1;

spawnLocation = teleportLocationGameObject.transform.Find("SpawnLocation").gameObject;

col.gameObject.transform.position = spawnLocation.transform.position;

rb.AddForce(rb.velocity);

}

}

}

}

The PortalTeleportation script does exactly what it says, it handles the teleportation of the player from one location to another. The script is attached to the portals that are spawned. First it does a check to make sure the character is the one colliding with the portal as well as checks that 2 portals have spawned. From here, you grab the rigidbody of the fps camera in order to determine the velocity you are going when you enter the portal. The next check is to determine if you are entering the first portal fired or the second portal fired. Then, the script spawns you in the location in front of the portal(Spawn location) and launches you with a force equivalent to what you jumped into the portal with. The script is undergoing a minor complication where the user is not launched in the direction the second portal is facing.

**Work to be finished before the next due date:**

1. The portals teleportation should work perfectly (without the cutting), and the user should be able to see the other side of the portal.
2. One full level with hazards should be implemented
3. Pause menu and in game menu should be implemented
4. Start working on more advanced features of “Cutting” the objects as they enter a portal slowly.
5. Implement 3D model to represent the player