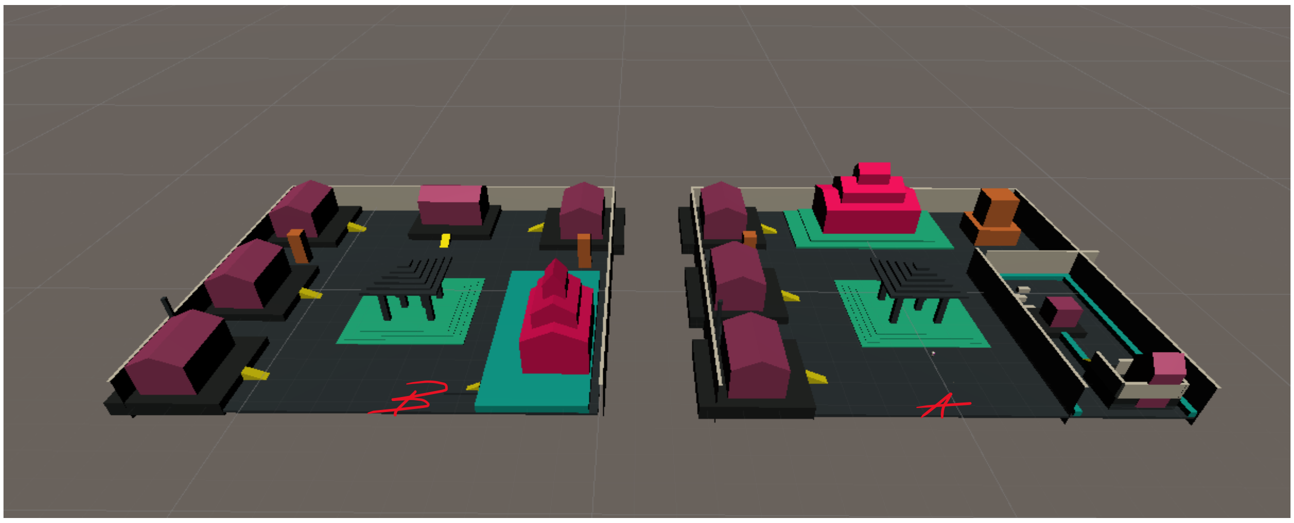
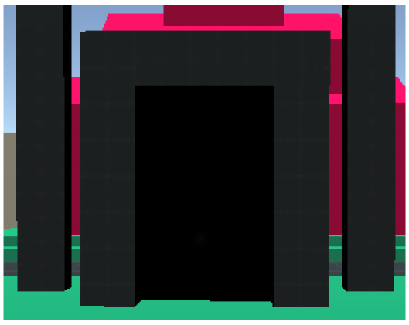
Teleport through Portals in Unity

Part 1 (See through the Portal)

We need to create two worlds side by side (World A and World B)

The first world will contain our player and the main camera, the second world will contain the player's character mimicking his movement.





Create a Portal using three cubes for our temporary graphics.

After create a plane rotating it 90 degrees on the x axis to fit the portal (rename it to render plane), remove the mesh collider of the plane.

Create a material for the render plane (CameraMat_B) (It will display the view from the other world)

Duplicate the players camera and move it to the other world so the camera is positioned in front of the portal (Rename it to Camera_B) and set to untagged.

Create a render texture (CameraTexture_B), drag the render texture to the Target Texture of the Camera_B.

To display the camera view of the World_B we drag the render texture (CameraTexture_B) to the Albedo Channel of the Material CameraMat_B, now the camera view is displayed in our plane.

The texture may appear distorted, to fix the issues we need to change the shader of the material

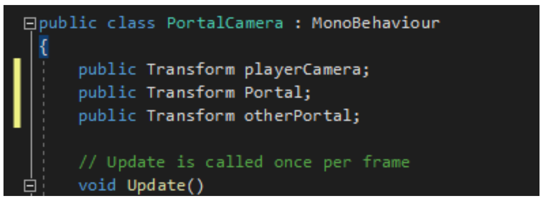
We cannot use the standard shader because we don't need to see the all view of the camera, we only want the part that we see through the portal, for this reason I will be using a Shader provided from the Video Tutorial.

Apply the Unlit ScreenCutoutShader to the Camera_B Material.

Create a Script (PortalCamera)

PortalCamera Script

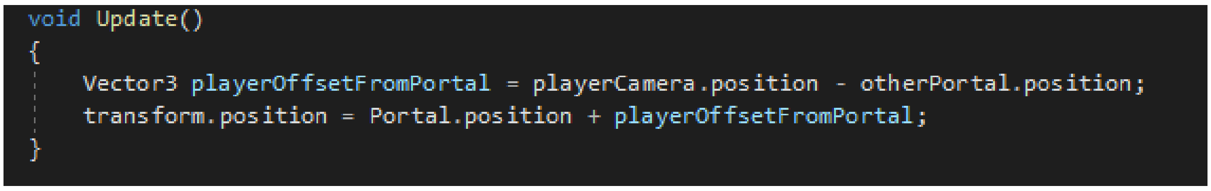
Reference the player's camera and both portals



```
public class PortalCamera : MonoBehaviour
{
    public Transform playerCamera;
    public Transform Portal;
    public Transform otherPortal;

    // Update is called once per frame
    void Update()
```

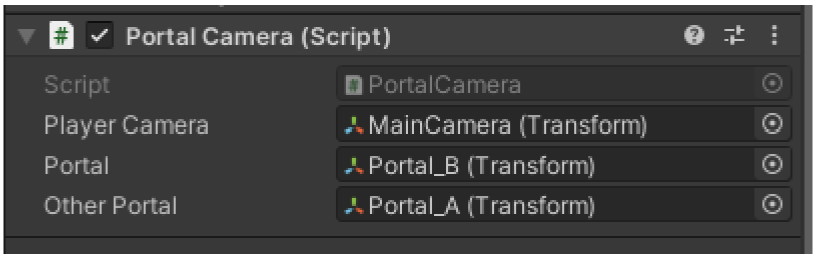
Portal- Portal_B   otherPortal- Portal_A

On the update we need to add the player's offset from his portal and set our current position.



```
void Update()
{
    Vector3 playerOffsetFromPortal = playerCamera.position - otherPortal.position;
    transform.position = Portal.position + playerOffsetFromPortal;
}
```

Assign the Player's camera and the Portals to the script on the inspector menu.

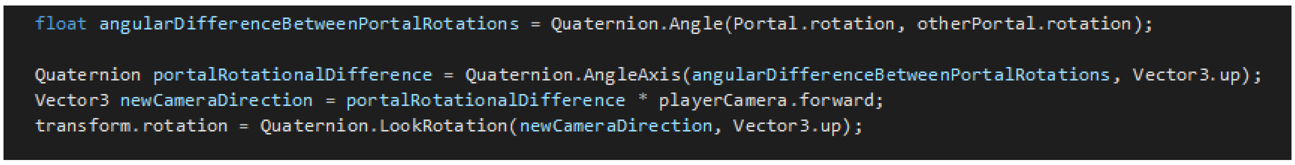


Portal Camera (Script)

Script	PortalCamera
Player Camera	MainCamera (Transform)
Portal	Portal_B (Transform)
Other Portal	Portal_A (Transform)

Now we need to offset the location by setting the angular difference between the portal rotations.

Set the Quaternion portal rotational difference and create a new vector to contain the direction we need to face in and turn it into a rotation.



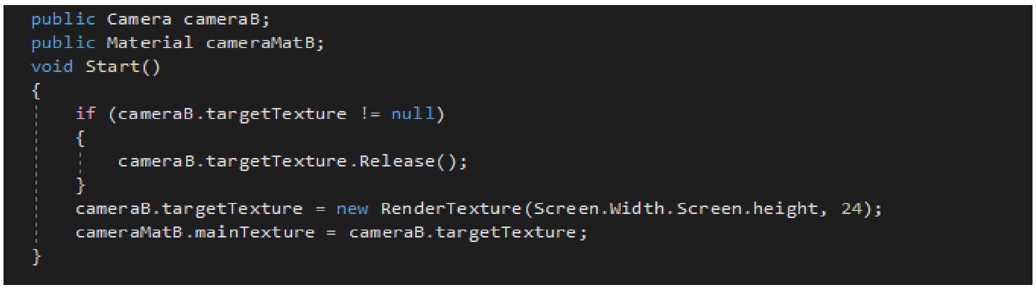
```
float angularDifferenceBetweenPortalRotations = Quaternion.Angle(Portal.rotation, otherPortal.rotation);

Quaternion portalRotationalDifference = Quaternion.AngleAxis(angularDifferenceBetweenPortalRotations, Vector3.up);
Vector3 newCameraDirection = portalRotationalDifference * playerCamera.forward;
transform.rotation = Quaternion.LookRotation(newCameraDirection, Vector3.up);
```

Create a empty object, rename it to GameManager and add a PortalTextureSetup Script

PortalTextureSetup Script

Reference the Camera (cameraB) and the Material(CameraMatB).

When the game starts, we want to see if the cameraB already has a texture assigned, it has a texture we want to remove it. Now that we have the camera free from texture, we want to set a new RenderTexture to the camera, we want the RenderTexture to be the width and height of the screen and the depth to be 24. Now we just need to set the texture to the camera.



```
public Camera cameraB;
public Material cameraMatB;
void Start()
{
    if (cameraB.targetTexture != null)
    {
        cameraB.targetTexture.Release();
    }
    cameraB.targetTexture = new RenderTexture(Screen.Width.Screen.height, 24);
    cameraMatB.mainTexture = cameraB.targetTexture;
}
```

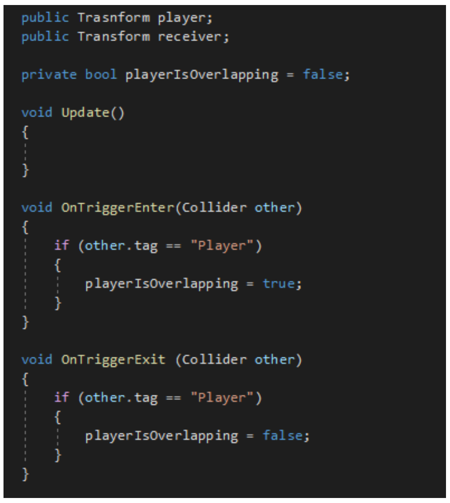
Part 2 (Transport through the Portal)

Select the render plane from world A, we need a collider that triggers when the player enters.

For that we duplicate the RenderPlane(remove the mesh renderer and the plane) rename it to ColliderPlane and add a box collider (resize it to fit the RenderPlane) move the collider slightly forward to the plane and set the collider as is trigger.

Add a PortalTeleporter Script to the ColliderPlane

PortalTeleporter Script



```
public Trasnform player;
public Transform receiver;

private bool playerIsOverlapping = false;

void Update()
{

}

void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
        playerIsOverlapping = true;
    }
}

void OnTriggerExit (Collider other)
{
    if (other.tag == "Player")
    {
        playerIsOverlapping = false;
    }
}
```

Reference the player and the receiving portal.

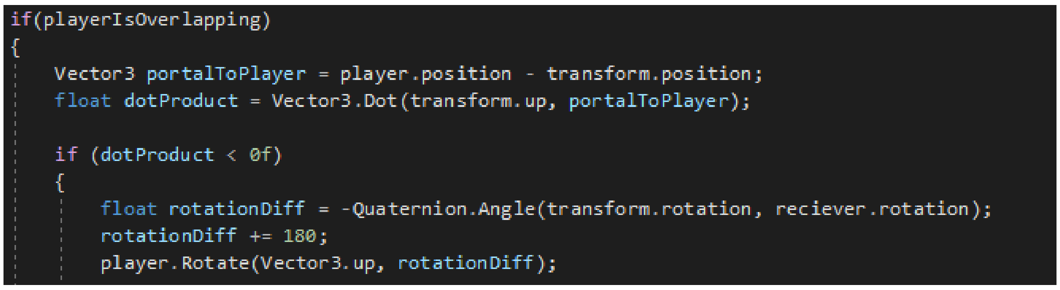
First we need to know if the player is colliding with the portal, if it is then the portal and the player is overlapping and we need to create a private Boolean and set it equal to false.

When the player enters we set the Boolean to true and once it exits the portal we set it to false.

Now we know when the player is overlapping.

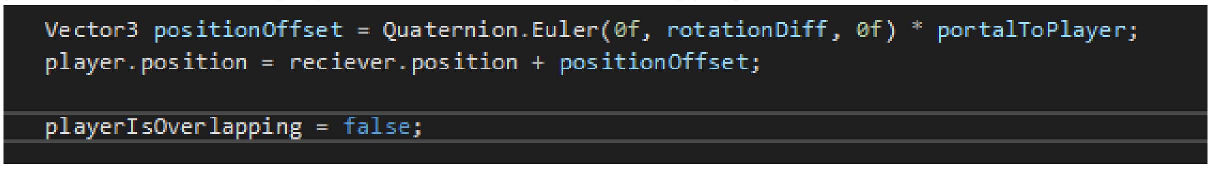
Inside the Update method if the player is overlapping we want to use the dot product(making sure the player enters in the right side of the portal).

If the Dot product is less than 0 (the player has moved across the portal), to teleport the player we need to set the difference between the rotations and add a rotation difference to plus 180 (so the player exits and enters in the same direction from both of the worlds).



```
if(playerIsOverlapping)
{
    Vector3 portalToPlayer = player.position - transform.position;
    float dotProduct = Vector3.Dot(transform.up, portalToPlayer);

    if (dotProduct < 0f)
    {
        float rotationDiff = -Quaternion.Angle(transform.rotation, reciever.rotation);
        rotationDiff += 180;
        player.Rotate(Vector3.up, rotationDiff);
```

Rotate the player according to the rotation difference and move the player by applying the positionOffset, set the players location and set he overlapping to false



```
    Vector3 positionOffset = Quaternion.Euler(0f, rotationDiff, 0f) * portalToPlayer;
    player.position = reciever.position + positionOffset;

    playerIsOverlapping = false;
```

Now we just need to duplicate the RenderPlane and the ColliderPlane from World A to World B and flip the position and rotation(set the position z axis to minus and the rotation z axis to 180).

Set the ColliderPlane from world 2 as a receiver on the ColliderPlane from world two and vice-versa.

Problem faced: The teleport script was not working because the player was untagged.

Portal Camera full script

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

public class PortalCamera : MonoBehaviour
{
    public Transform playerCamera;
    public Transform Portal;
    public Transform otherPortal;

    // Update is called once per frame
    void Update()
    {
        Vector3 playerOffsetFromPortal = playerCamera.position - otherPortal.position;
        transform.position = Portal.position + playerOffsetFromPortal;

        float angularDifferenceBetweenPortalRotations = 
Quaternion.Angle(Portal.rotation, otherPortal.rotation);

        Quaternion portalRotationalDifference = 
Quaternion.AngleAxis(angularDifferenceBetweenPortalRotations, Vector3.up);
        Vector3 newCameraDirection = portalRotationalDifference * 
playerCamera.forward;
        transform.rotation = Quaternion.LookRotation(newCameraDirection, Vector3.up);
    }
}
```

Portal texture setup full script

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

public class PortalTextureSetup : MonoBehaviour
{
    public Camera cameraB;
    public Material cameraMatB;
    void Start()
    {
        if (cameraB.targetTexture != null)
        {
            cameraB.targetTexture.Release();
        }
        cameraB.targetTexture = new RenderTexture(Screen.width, Screen.height, 24);
        cameraMatB.mainTexture = cameraB.targetTexture;
    }
}
```

Portal Teleporter full script

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

public class PortalTeleporter : MonoBehaviour
{
    public Transform player;
    public Transform receiver;
    private bool playerIsOverlapping = false;

    void Update()
    {
        if(playerIsOverlapping)
        {
            Vector3 portalToPlayer = player.position - transform.position;
            float dotProduct = Vector3.Dot(transform.up, portalToPlayer);

            if (dotProduct < 0f)
            {
                float rotationDiff = -Quaternion.Angle(transform.rotation,
receiver.rotation);
                rotationDiff += 180;
                player.Rotate(Vector3.up, rotationDiff);

                Vector3 positionOffset = Quaternion.Euler(0f, rotationDiff, 0f) *
portalToPlayer;
                player.position = receiver.position + positionOffset;

                playerIsOverlapping = false;

            }
        }
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {

            playerIsOverlapping = true;
        }
    }

    void OnTriggerExit (Collider other)
    {
        if (other.tag == "Player")
        {
            playerIsOverlapping = false;
        }
    }
}
```