Master Test Report

Mike Driessen GD4A

Kaiser

Introduction

1. Introduction
2. The idea / The Final product
3. Production / Grappling hook
4. Multiplayer interaction & Controllers
5. Placing blocks / Testing
6. Objects spawning & UI
7. Raycast Collison check.
8. Particles & Lighting & Tests

The idea

The game itself originated from the game Half + Half. This is a game where you play hide and seek in VR. Our idea was to implement the hide and seek concept into a VR and AR game. The AR player would be in the real office/ environment and the VR players in the virtual environment that looks like the real office.

The AR player would be the seeker and has to shoot the VR players when one is found, when the seeker has shot all the hiders the seeker has won. The way the hiders or VR players can win is by collecting all the collectables randomly spread around the room.

When a hider or VR player has been shot they will be spawned inside the spawn lobby, there they can look into the office and inspect the other players.

The final product

We got two weeks' time to produce our master test. We managed to implement most of the ideas we had. Some were not implemented due to time concerns such as:

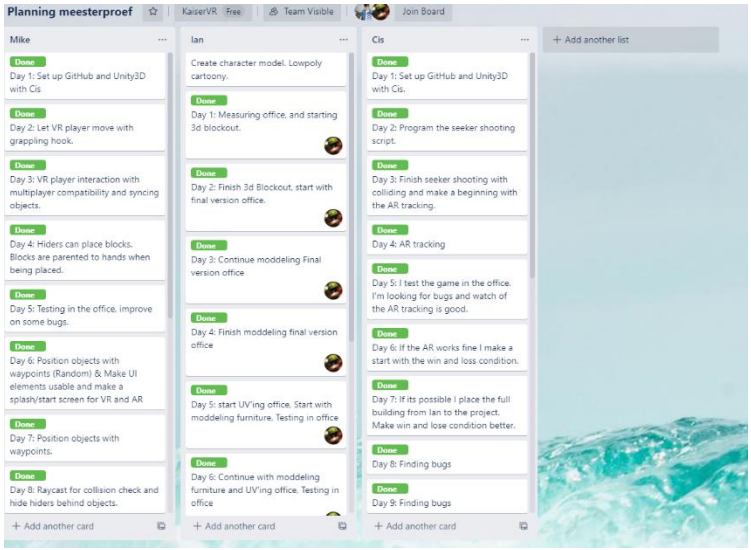
- Build your own hiding places in the beginning of the game. (Not working fully with multiplayer)

- Raycast that had to make the VR player disappear as soon as it was hidden behind an object. (Conflict with multiplayer between AR and VR)

- AR tracking is not quite how we hoped.

Production

We first started getting the idea together and making it so that it was possible to create in 2 weeks' time. All the mechanics and models were laid out an were split. Cis had more experience with multiplayer in unity and implementing AR. I had more experience with creating UI, interaction in VR and different mechanics. So we distributed our tasks and started making a schedule on Trello so it was easy to see what everyone had to do each day.



Grappling hook

The grappling hook is one of the main ways the VR player is able to move through the office. There were a few possibilities but I chose to when the trigger is pressed by the player a Raycast shoots and if a valid grabbable object is hit you will be "reeled in " with a relative speed. Also implemented some ways so it is easy to pick up a grapple for the player. When inside a range they can grab the grapple and it attached to their hand. When attached there is no way to get if off. The grapple draws a line when something valid is hit between the distance of the hit object and the grapple. Also a hook will teleport to the hit position. It still has the rotation of the grapple.

You can also move objects that are dynamic towards you, so if a dynamic hiding block is placed you are able to take one with you and move it around.

Gravity is disabled when reeling in the player and enabled when releasing the trigger. Here is a small example:

```
void ReelInGrapple(float triggerValue) {

    if(validTargetFound && currentGrappleDistance > MinReelDistance) {

        // Move object towards our hand
        if(isDynamic) {
            grappleTargetRigid.isKinematic = false;
            grappleTargetRigid.transform.parent = grappleTargetParent;
            grappleTargetRigid.useGravity = false;
            grappleTargetRigid.AddForce((MuzzleTransform.position - grappleTargetRigid.transform.position) * 0.1f, ForceMode.VelocityChange);

            //r.MovePosition(MuzzleTransform.position * Time.deltaTime);
        }
        // Move character towards Hit Location
        else {
            Vector3 moveDirection = (HitTargetPrefab.position - MuzzleTransform.position) * GrappleReelForce;

            // Turn off gravity before we move
            changeGravity(false);
            characterController.Move( motion: moveDirection * Time.deltaTime * triggerValue);
        }
    }
    else if(validTargetFound && currentGrappleDistance <= MinReelDistance) {

        if (isDynamic) {
            //grappleTargetRigid.useGravity = true;
            grappleTargetRigid.velocity = Vector3.zero;
            grappleTargetRigid.isKinematic = true;
            grappleTargetRigid.transform.parent = transform;
        }

        if(!climbing && !isDynamic) {
            // Add climbable / grabber
            ClimbHelper.transform.localPosition = Vector3.zero;
            bngController.AddClimber(ClimbHelper, thisGrabber);
            climbing = true;
        }

    }
```

Multiplayer interaction

Most multiplayer aspects were handled by Cis, he used the framework beforehand and was able to set up client and server communication. I only had the task of implementing UI elements such as a canvas for starting the game and a small block counter in the left top of the headset which follows the players head position.

Multiplayer

We used photon for the multiplayer in our game. This is a plugin that gives you the tools to set up a multiplayer game. It was easy to set up a multiplayer server online using the free version. There was also a Voice plugin Photon voice, this way you are able to let players speak to each other.

Controllers

Also figuring out how the player controllers can be manipulated, this is handled through a simple bridge which uses oculus SDK to communicate all controller interaction.

You will use Input Bridge in your script to access all the buttons, then just use for example if(input.AButton){Action}. This is an easy way to access all controller buttons.

```
if (input.AButton && HasPressed == 0 && MaxSpawned < 5 && stop == false)
{
    HasPressed = 1;

    gameObject.GetComponent<PhotonView>().RPC( methodName: "OnBButtonPressed", RpcTarget.All);

}
```

Example on input usage.

```
AButton = OVRInput.Get( virtualMask: OVRInput.Button.One, OVRInput.Controller.RTouch);
AButtonDown = OVRInput.GetDown( virtualMask: OVRInput.Button.One, OVRInput.Controller.RTouch);
```

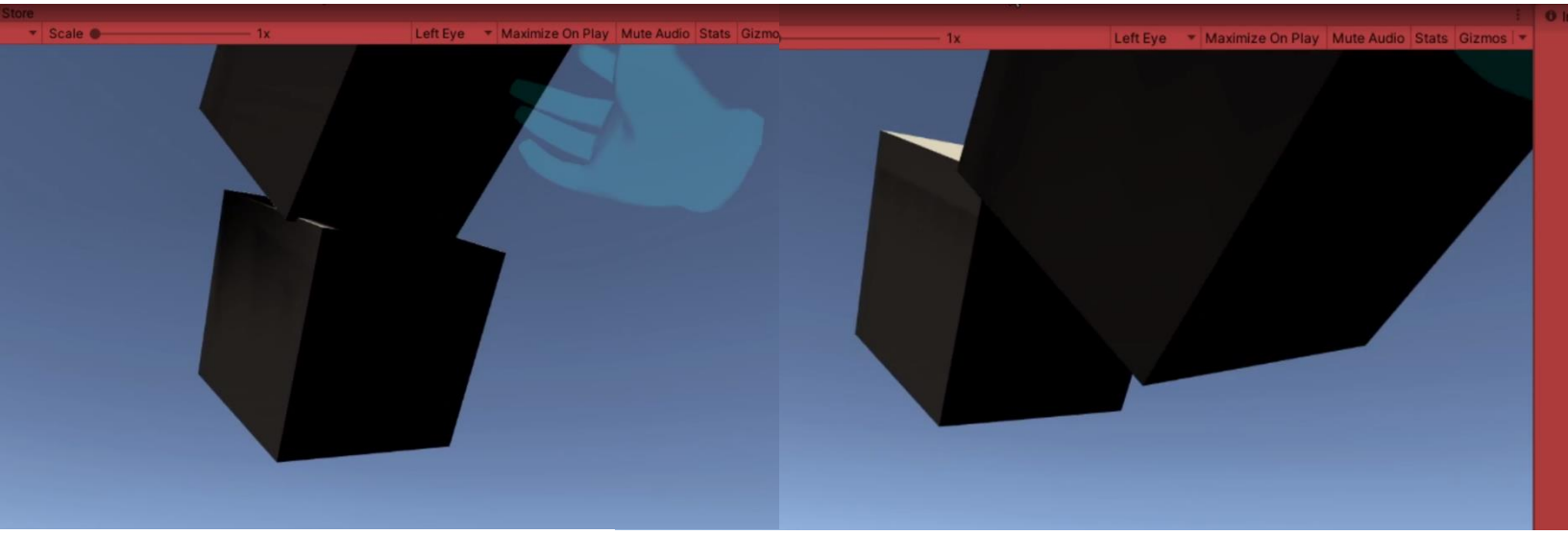
Example how bridge uses OVRInput for oculus controllers.

Placing blocks

For placing the blocks I had to assign 3 different buttons on the VR controllers. Also they had to be attached as a child to the controller parent. When attached Gravity, Collision and rigid body constraints are disabled.

When the player places a block by pressing B the block spawns with collision and physics and is made blue. When the player places a block by pressing X the block spawns without physics and only has collision, the block is then made red.

There is also a count on how many can be placed. Also taking into account that all objects are synced using multiplayer.



Testing

During the testing phase we saw a lot of issues not first thought of such as blocks when spawned were too big, player when shot by AR not spawning in the correct position when using Grappling hook. And some other small things. We tried to fix the major things and got on with the rest. A lot of testing the multiplayer with VR and AR has been done in the meantime.

Objects spawning & UI

This was for implementing the spawning of all collectables.

First off I made a small example for placing the objects randomly through the area by defining an area and letting objects randomly spawn through the office. This way was not the way to do it but we had an example. The objects would spawn inside other objects inside the area and only on level ground.

So I had to do it another way that would not take too much time, so I used “Waypoints” to spawn the objects. You could define all places you want collectables to spawn in an array and those would be taken randomly. Also we did not want to let them spawn in the same spot so I made a list with all locations upon spawning and removing the ones used from the list so there was no way another could spawn on the same spot. Also you could define a maximum.

The UI was mostly pretty easy but we had to figure some small things out. We already had the interaction for the VR player with a canvas. The pointer shows up when both controllers are held and then you can interact with the canvas. So we had a canvas that is always located in the start room and there you can press the play button to start the game.

Here is the menu panel and shows up when menu button is pressed on the controller. Start is not used in this menu.

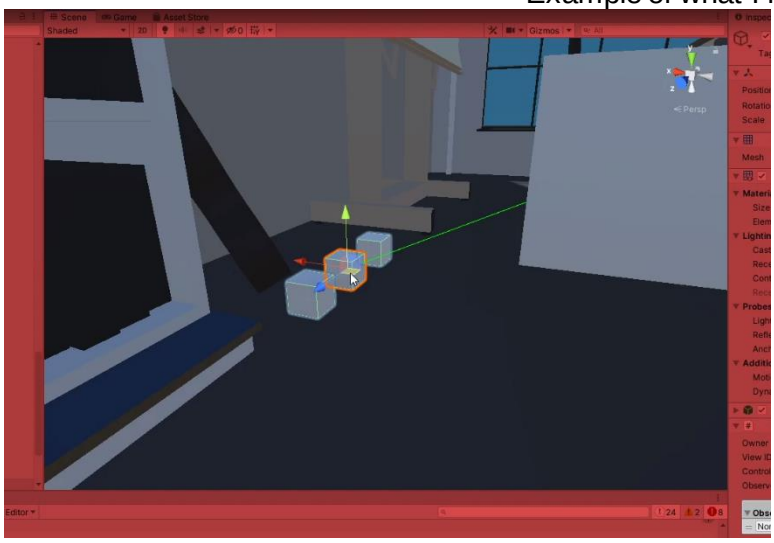
When options button is pressed

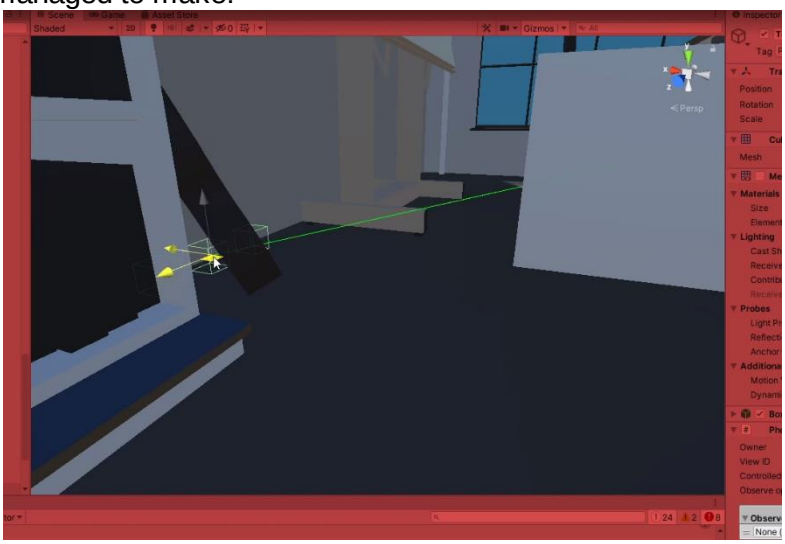
Raycast Collison check.

This was one of the major problems with using the real and virtual world. Not letting the other player be seen in AR whilst they are behind something in the virtual world.

The way I wanted to implement such a fix seemed pretty straight forward. Casting a Raycast to the AR player from the VR player. If the Raycast hits the AR player the VR player could be seen. If the Raycast got obstructed by another object and the Raycast would not hit the AR player anymore the VR player would not be rendered for the AR player. This way it would look just like you hid behind an object in the virtual world.

Example of what I managed to make:





The way I wanted to implement this with our multiplayer framework was by disabling the photon view on the player, so that the player was invisible to all players and was not rendered by the framework. I could disable this through script by finding the photon view ID, but this was not synced to all clients.

Another idea was just to disable all renderers but this was not the way to do it. There was nothing that could be synced to all clients.

Then we came to the conclusion that the framework was not able to send data from disabling objects to the other clients and disabling photon view was not working either.

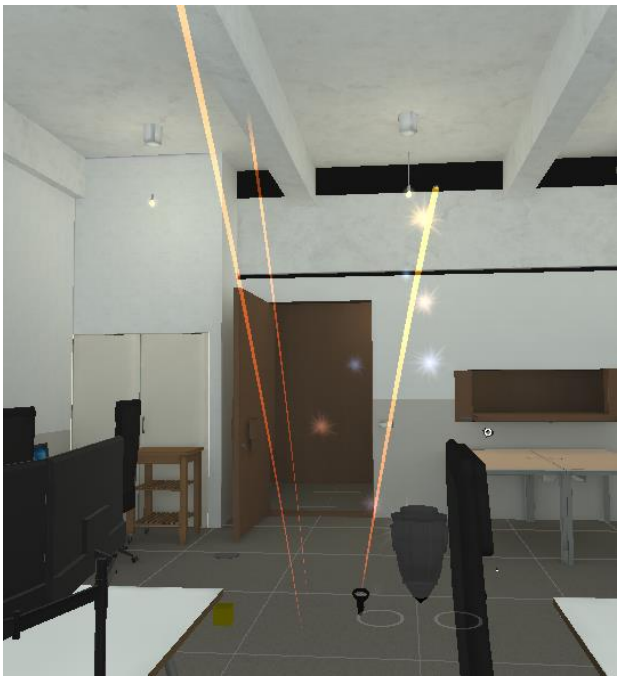
We thought it would be very easy to make something like this but things took a turn for the worst.

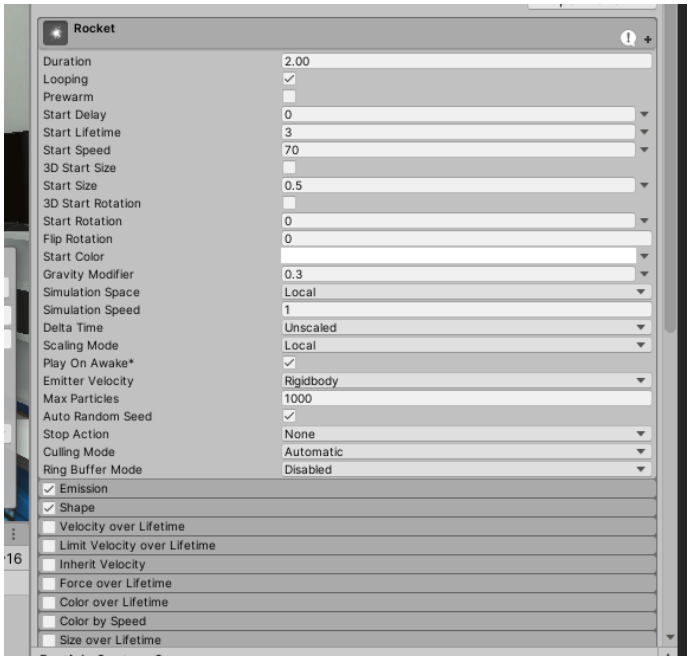
```
[PunRPC]
Frequently called  3 usages  mikedriessen *
void DisableRenderer(int ObjToRemove, bool setActive)
{
    PhotonView Disable = PhotonView.Find(ObjToRemove);
    Disable.transform.gameObject.GetComponent<MeshRenderer>().enabled = setActive;
}
[PunRPC]
Frequently called  3 usages  mikedriessen *
void EnableRenderer(int ObjToEnable, bool setActive)
{
    PhotonView Enable = PhotonView.Find(ObjToEnable);
    Enable.transform.gameObject.GetComponent<MeshRenderer>().enabled = setActive;
}
```

Above you can see how I disabled Photon view using the Photon view ID.

Particles & Lighting & Tests

Having almost finished I started making some winning particles using color gradients and particle effects inside unity. I came up with some small fireworks so that you could see one or the other had won. And you would see these in the middle of the office.





Settings

Lighting

For the lighting in our office we used panel lights at the windows, point lights for the small lights on the ceiling and changed some global light settings for the directional light and shadows. Also the android variant of the game has less demanding light settings baked into the game. Also a lot of fiddling was needed to get it the best we could.

Testing

Testing was the thing we did most. Solving one bug and getting the other and due to time we could not do all the things we wanted. We showed our work and tested together with our Internship instructor. We got feedback form him. This made it easier to improve on some smaller things.