Development of a Mixed Reality Solution for Emergency Exit Training

Internship report

ISIMA
Ingénieur informatique et modélisation - Système embarqué et Réalité virtuel
Stage - Frédéric RENAUT
ZZ2 - F1

Rémi BOUDRIE
20/08/2025

Abstract

This project was carried out as part of an internship at the UiT Arctic University of Norway in Narvik. Its objective was to design a solution to train users to respond effectively in emergency evacuation situations. To achieve this, we used the Unity 6 game engine and the Meta Quest 3 mixed reality headset.

The system leverages mixed reality to overlay virtual elements onto the real environment. Guiding arrows are displayed to indicate the path and the sequence of doors the user must follow to reach the nearest emergency exit from their current location.

The primary goal of this work was to demonstrate the feasibility of such a project. It serves as a foundation that can be further developed in the future to make the solution more efficient and accessible to a wider audience.

Keywords

Unity, Meta Quest 3, Mixed Reality

Résumé

Ce projet a été réalisé dans le cadre d'un stage à l'UiT Arctic University of Norway à Narvik. Son objectif était de concevoir une solution permettant d'entraîner les utilisateurs à réagir en cas d'évacuation d'urgence. Pour cela, nous avons utilisé le moteur de jeu Unity 6 ainsi que le casque de réalité mixte Meta Quest 3.

Le système exploite la réalité mixte pour superposer des éléments virtuels à l'environnement réel. Des flèches de guidage apparaissent ainsi afin d'indiquer à l'utilisateur le chemin et la succession de portes à emprunter pour rejoindre la sortie d'urgence la plus proche de sa position actuelle.

L'objectif principal de ce travail était de démontrer la faisabilité d'un tel projet. Il constitue une base qui pourra être améliorée à l'avenir afin de rendre la solution plus efficace et accessible au plus grand nombre.

Mots-clés

Unity, Meta Quest 3, Réalité Mixte

Acknowledgement

I would like to express my gratitude to all those who supported me during my internship. I am especially thankful to Frédéric RENAUT that trusted me to go on this internship and also to my school tutor, Mr. GUITTON Alenxendre, for his guidance and support throughout this period. I would also like to thank my colleague Sami for his help during this project. Finally, I would like to thank Mr. YU Hao my tutor at the UiT The Arctic University of Norway for his invaluable advice and support during my internship and it's collegue Mr. WANG Marius that helped us with all our questions and issues we had during the project.

Table of Contents

Table of Figures

Introduction

The internship took place at UiT – The Arctic University of Norway, a multi-campus institution with sites spread across Northern Norway, including one in the city of Narvik. This campus is home to the Department of Industrial Engineering, which combines theoretical knowledge with practical, industry-oriented applications in an international environment. It was within this department that the internship was carried out, providing access it's facilities but to the material that was used during this project.

The project proposed by our tutor, Hao Yu, was to develop a virtual reality application to train students for evacuation in case of an emergency. Since our tutor was not specialized in virtual reality, we were advised to consult Marius Wang for any necessary assistance. With his input, the scope of the project became more clearly defined. It was then decided to use Mixed Reality (MR) in order to avoid recreating the university environment entirely in 3D, and instead overlay virtual informations onto the real world to direct users toward the nearest emergency exit. It was also agreed that the main objective would be to create a foundation that could be further improved in the future, given the complexity of the task and the current state of mixed reality technology.

To carry out this project, UiT provided us with a Meta Quest 3 headset, which we used to develop the mixed reality application. This device is considered a relatively low-cost VR headset that also offers mixed reality capabilities. To create the application for this headset, we chose to use Unity, as I already had prior experience with this game engine for virtual reality usages. For the interaction between Unity and the headset, two options were available; we selected OpenXR for its multi-platform compatibility, ensuring that if a different headset were to be used in the future, the project would not need to be rebuilt from scratch.

At the beginning of this report, we will first define the scope of the project in more detail, providing additional information about UiT, the project objectives, and the materials used to carry it out. The second part will cover the fundamentals of a mixed reality project, followed by the hand gestures that can be used to trigger different functionalities within a mixed reality environment, and finally the debugging features implemented during development to simplify the identification of potential issues. In the final part, I will present my personal contribution to the project: the system that guides the user toward the nearest emergency exit. This section will first focus on the rooms and doors and their implementation, and will then explain the use of the Dijkstra algorithm to determine the shortest path between the user's position and an emergency exit.

1 | Scope of the Project

1.1 UiT The Arctic University of Norway

As mentioned in the introduction, we will first start by talking about the university that hosted our internship. UiT – The Arctic University of Norway [1] is a cluster of multiple universities located across the northern region of Norway, with a total of 11 different campuses, as shown on the map below.

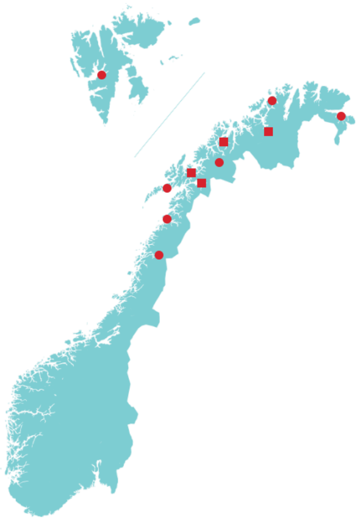


Figure 1: Campus locations of UiT the Arctic University of Norway

Campuses are spread throughout northern Norway to ensure that students across the country can access advanced education not only in larger cities like Tromsø but also in smaller cities such as Svalbard, which hosts the northernmost campus.

Our internship was located more specifically at the Narvik campus, in a modest city. Despite its size, we were warmly welcomed by the university and had access to well-equipped facilities. In the Department of Industrial Engineering, we were introduced to an immersive projection room equipped with multiple video projectors that displayed images on each wall and the floor. This setup allowed users to be immersed in a virtual environment without causing the motion sickness often associated with headsets.

After exploring the campus, its facilities, and the resources available to us, we were ready to focus on the main goal of our internship. The next step was to clearly define the objectives of the project and the approach we would take in its development.

1.2 Objectives of the Project

After our tour of the Department of Industrial Engineering, Hao explained that during our internship we would develop a virtual reality application to help train students in case of emergency. Since he was not specialized in virtual reality, he introduced us to Marius Wang, who became our main contact for the project thanks to his experience in the field. After a brainstorming session with Marius, it was decided that the project would focus on mixed reality, since this field is less well known than virtual reality and would provide us with a valuable opportunity to explore it further.

Following these discussions, Marius provided us with a template that we needed to complete before starting the project. In this template, we defined some milestones and created the preliminary Gantt chart shown below.

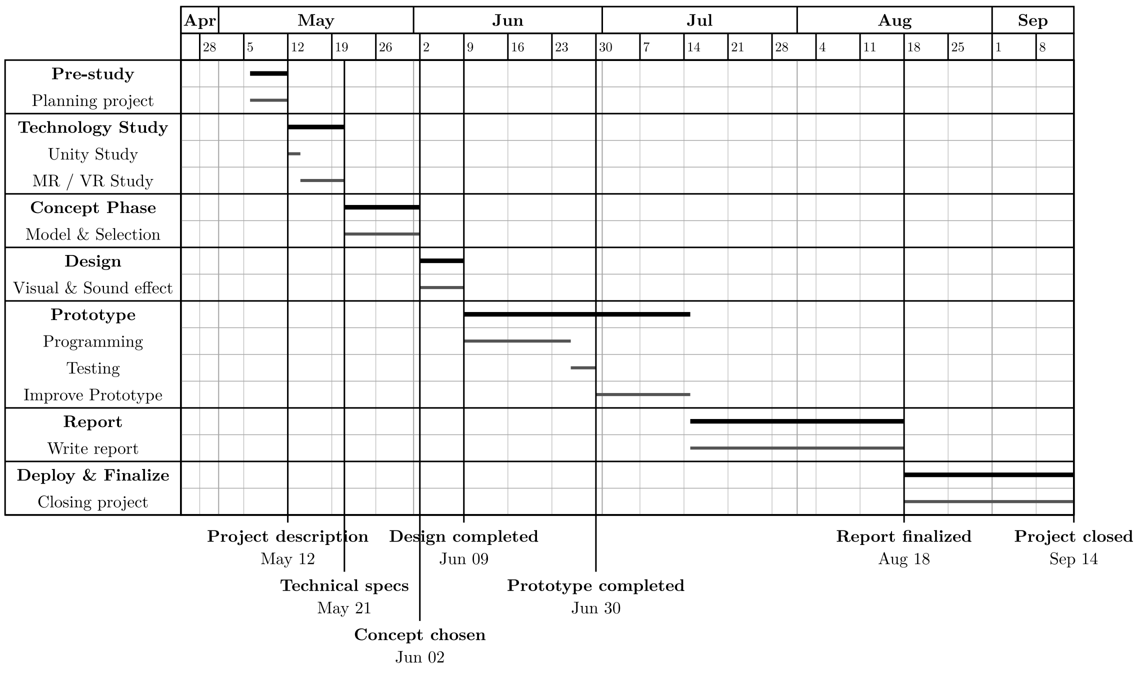


Figure 2: Preliminary Gantt chart for the project

This Gantt chart used generic labels provided in the template, which were not very precise about our actual tasks. These labels were later updated in the final Gantt chart, which is present in my partner's report.

Once these formalities were completed, we had a meeting with Marius, Hao, and other members of the department to present our ideas for the project. This was a good opportunity to receive feedback and confirm that we were heading in the right direction.

In the end, it was decided that our project would use mixed reality to guide students from room to room in the building using virtual visual cues, ultimately leading them to the nearest emergency exit depending on their location. It was also suggested that different scenarios could be added, though this was not a priority and would only be implemented if time and technological limitations allowed. The project is available on my colleague's GitHub [2].

1.3 Project Material

Now that we had a clear idea of what the project would involve, we needed two main things to achieve it: a headset that supported mixed reality and a game engine to develop the application.

1.3.1 Meta Headset

UiT provided us with a Meta Quest 3 headset for the development of our application. This headset supports mixed reality, making it ideal for our project. I was already familiar with this device since I had used it during my studies at ISIMA, though my previous experience was limited to virtual reality.

For the project, the headset needed to meet several requirements. First, it had to support mixed reality at a high enough quality to provide a good user experience. Next, it had to include hand-tracking functionality, as interacting with virtual elements using hands feels more natural than using a controller. Finally, it had to support an open development platform so we could build our application without unnecessary restrictions. The Meta Quest 3 met all these requirements and was therefore the perfect choice for our project.

Figure 3: Meta Quest 3 headset

The headset was an excellent choice since it offered all the necessary features while remaining more affordable than other headsets on the market. It was also readily available at UiT, so we did not need to request a new purchase. This saved us time and allowed us to focus on development rather than hardware procurement. However, as we will see later in this report, this choice also came with some limitations.

1.3.2 Unity

With the headset chosen, the next step was to select a game engine for development. The two main options on the market are Unity and Unreal Engine. Both are powerful tools for creating virtual reality applications, but they differ in certain aspects that made us choose Unity. I already had some experience with Unity from a previous virtual reality project, so I was familiar with its features. Unreal Engine, while more powerful for high-end graphics, is also more complex to use. Since our project focused on interaction rather than visual fidelity, Unity was the best choice.

Unity is a very powerful game engine, but our setup presented a challenge. While UiT provided the Meta Quest 3, we had to use our own laptops for development. Unfortunately, my laptop was not powerful enough to handle the requirements, so we had to rely on my colleague's laptop for the entire project. Sharing one computer for development made the process slightly more complicated, but since we had already worked together on previous projects, we were able to manage effectively.

1.3.3 OpenXR

To develop a mixed reality application with Unity, we needed to install specific packages that would enable the Meta Quest 3 to function properly. Two options were available: the Oculus Integration package and the OpenXR package. The Oculus Integration package, developed by Meta, was more stable and included more features. However, the OpenXR package was more open and allowed future compatibility with other headsets. Since our project aimed to establish a foundation that could be further improved later, we chose OpenXR. This way, future developers could use other headsets without starting from scratch.

To use OpenXR with the Meta Quest 3 in mixed reality, additional packages were required. The first was the OpenXR Plugin package, which enabled OpenXR features in Unity. The second was the Meta Quest XR Plugin, which allowed us to use Meta Quest 3–specific features. Finally, the AR Foundation package enabled mixed reality features in Unity. These packages came with certain limitations, summarized in the table below, taken from the official AR Foundation documentation [3].

Feature	Android	iOS	visionOS	HoloLens	Meta Quest	Android XR	XR Simulation
Session	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Device tracking	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Camera	Yes	Yes			Yes	Yes	Yes
Plane detection	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bounding Box detection		Yes			Yes		
Image tracking	Yes	Yes	Yes				Yes
Object tracking		Yes					
Face tracking	Yes	Yes				Yes	
Body tracking		Yes					
Point clouds	Yes	Yes					Yes
Ray casts	Yes	Yes		Yes	Yes	Yes	Yes
Anchors	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Meshing		Yes	Yes	Yes	Yes		Yes
Environment Probes	Yes	Yes	Yes				Yes
Occlusion	Yes	Yes			Yes	Yes	Yes
Participants		Yes					

Table 1: Limitations of AR Foundation with Meta Quest 3

As shown in this table, the Meta Quest has some limitations compared to smartphones, but among headsets, it offers the widest range of features. For this reason, using the Meta Quest 3 with the OpenXR package was a solid choice for our project.

Now that the headset, game engine, and required packages were selected, the next chapter will introduce the basics of a mixed reality project, including hand gestures and the debug menu created to assist us during development.

2 | Mixed Reality

2.1 Mixed Reality Basis

We are now going to start the development of the Mixed Reality application. To start we need to first create the project. For that it's first better to choose a Universal Render Pipeline (URP) template. This will allow us to have a better performance and more features than the default render pipeline. Then when we enter the project we can start to install the packages that we talked about earlier. That step is also pretty simple since inside Unity there is a package manager that allows us to install the packages we need. There we can also download some samples that will help us to understand how to use the packages and how to implement certain functionalities in our project. Now we can finally begin to add our first step to the Mixed Reality application which is the XR Rig.

2.1.1 XR Rig

The XR Rig is the base of any Virtual and Mixed Reality application. This Game Object will be the one that serve to represent the user of the application. So it will need to contain multiple things. To import this XR Rig we can use the XR Interaction Toolkit package that we installed earlier. This package contains a prefab of the XR Rig that we can use and modify to make it usable in a Mixed Reality application.

The first modification that needs to be made is on the Camera of the XR Rig. The Camera corresponds to the user's view in the application. But right now it is set to be inside a fully virtual environment, or we want to use it in a Mixed Reality application. For that to be possible we need to change the Camera's Background type to "Solid Color" and set the Background color to Black with a transparency of 0. This will allow us to see the real world through the cameras of the device. Though it is not the end of the modifications on the Camera. We also need to had AR Camera Manager Component from the AR Foundation package. This component will allow us to use the cameras of the device to see the real world and to track the position of the device in it. So in the end we will have the following Camera Game Object.

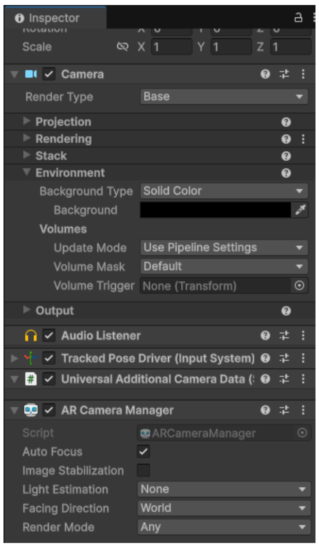

Inspector
Scale X 1 Y 1 Z 1
Camera
Render Type: Base
Projection
Rendering
Stack
Environment
Background Type: Solid Color
Background
Volumes
Update Mode: Use Pipeline Settings
Volume Mask: Default
Volume Trigger: None (Transform)
Output
Audio Listener
Tracked Pose Driver (Input System)
Universal Additional Camera Data (
AR Camera Manager
Script: ARCameraManager
Auto Focus: ✓
Image Stabilization
Light Estimation: None
Facing Direction: World
Render Mode: Any

Figure 4: Camera Game Object in the XR Rig

Now that the Camera is set up the user will be able to see the real world through it. But right now the user doesn't have any way of interacting with anything because the hands aren't set up and neither are the controllers. To be able to set up the hands and the controllers we actually opened a sample from the XR Interaction Toolkit package that is named "Hands Interaction Demo". This sample contains a XR Rig that is already set up to use the hands and the controllers. So we can just copy those elements from the Rig and just add them into ours. We had to remove some elements that were not needed in our case, like the teleportation system since in our project the user will move around in the real world and not in the virtual environment.

Now we can just do one final modification to the XR Rig configuration. On the XR Rig Game Object there is the XR Origin Component. This component is responsible for the positon and rotation of the XR Rig in the real world. But right now it is set to be at the origin of the virtual environment. So we need to change the Tracking Origin Mode to "Floor" so that the XR Rig will be positioned on the floor of the real world. That way the user will be able to move around in the real world and the XR Rig will follow the user's movements. Unfortunately, this is how it works in a virtual reality application but in a mixed reality application like ours this functionality doesn' twork at all and only the Camera is moving around. This will cause two different problems. The first one is that the every Game Object that will be a child of the XR Rig will not follow the player. The second one is that now we don't have a way to know the user's height in the real world. A solution to the first problem will be talked about later in the section about the Debug Menu. As for the second problem, we first need to talk about AR Planes.

2.1.2 AR Planes and Bounding Boxes

So now we are going to talk about the AR Planes and what are their functionalities. The AR Planes are Game Objects that are used to represent the floor, walls and ceiling of the real world. But the problem is that those planes aren't generated because the headset detect the environment around it while the application is running. In reality those data are extracted from the space environment of the Meta headset.

This causes a lot of limitations to the application. The first one is that those space environment data are organized in rooms and the number of rooms is limited to around 15. This means that if we want to use this feature for the project it won't be possible for the project to work around the whole campus of the university. The second limitation is that those data are not savable and there is no way to transfer them between headsets. So if we register multiple room in one headset and then we try the application in another headset it won't work. To solve those problems we decided for our project to just register one single room and use it as a base and then all of their other rooms will be inside our Unity project. This way we can have a lot of rooms and the user will be able to move around in them without any problems. But to see more about this part of the project you can go read the internship report of my college since he was the one that worked on this part of the project.

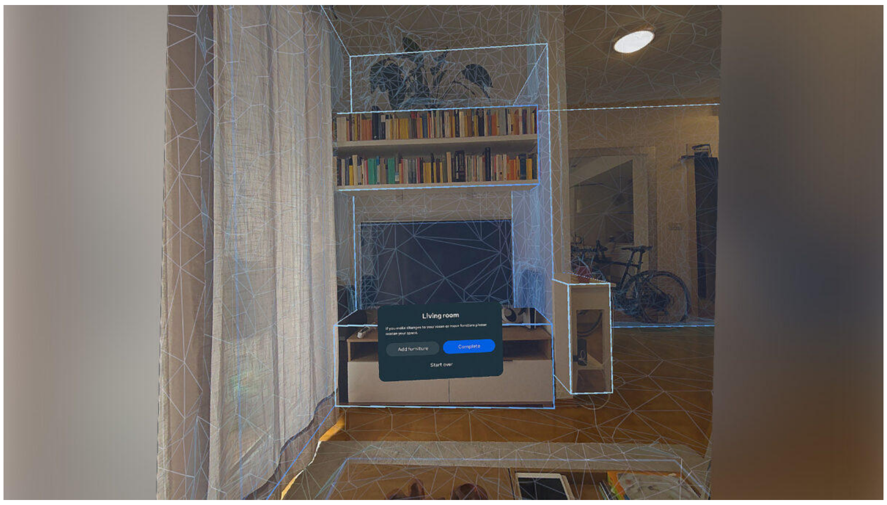

Living room
If you make changes to your room or move furniture please rescan your space.
Add furniture
Complete
Start over

Figure 5: Room Scanned with the Meta Quest 3

Now that we talked about how the AR Planes are generated we can talk about how to use them in our project. The first thing that we need to do is to add the AR Plane Manager Component to the XR Rig Game Object. This component will allow us to generate the AR Planes with the data from the headset. But we also need to add a prefab that will be used to represent the AR Planes in the real world. For that we can just simply use the one that is provided by the XR Interaction Toolkit package. So now when we start the application inside a room that has been scanned previously inside the headset we will be able to see the AR Planes that represent the floor, walls and ceiling of the room.

Since we can represent the floor now we can use it to calculate the user's height in the real world. So on player's camera we added a new object with a script. This script will have one job which is to first detect if the camera is above an AR Floor or not. If it's the case it's going to store this AR Floor and that's it. This information is going to be used for multiple things so that's why we need to store it. Actually since we use the AR Floor to know where the user is and where the other rooms are without this information nothing would work. So this script is really important since it almost start every functionality of that project.

With the information about the AR Floor we can now calculate the user's height in the real world. For that we just need to get the position of the AR Floor and the position of the

Camera. The difference between those two positions will give us the height of the user in the real world. This information is then stored in a variable that can be used later in the project. But this information is only calculated once when the AR Floor has been found. That way even if we aren't above the AR Floor anymore the height will still be the same. We need to use this because if the user changes floor its height will still be the same.

Now that the AR Planes are set up we can also talk about the Bounding Boxes. The Bounding Boxes work just like the AR Planes but instead of being a plane that represents the floor, walls and ceiling of the room they are used to represent the objects that are in the real world by using boxes. The data that are used to generate the Bounding Boxes are also extracted from the space environment of the Meta headset. So the also have the same limitations as the AR Planes. But as the AR Planes that represent the floor, walls and ceiling of the room that aren't really movable object it can cause some problems for the Bounding Box because they are used to represent objects that can be moved around. So if we actually move an object inside a room that has been scanned previously you'll need to scan it again to update any object that has been moved. Since the project doesn't really have any Game Object that would need an interaction with the real world objects and since it seems like a lot of work to do every time some object is moved we decided to not use this feature for our project.

2.1.3 AR Anchors

Now that we talked about the AR Planes and the Bounding Boxes we can talk about the AR Anchors. The AR Anchors are a Component that can be added to any Game Object in the scene. This Component will allow us to anchor the Game Object to a specific position in the real world. This means that if we add an AR Anchor to a Game Object and then we move the headset around, the Game Object will stay at the same position in the real world. Even thought we move in the real world a normal Game Object would actually stay in the place where we left it. Since in that case the AR Anchor doesn't sound really useful since an object without it would still stay in place. But it's the case if we don't use the reset view from the headset, because if we do that the Game Object will be moved to the new position of the headset. So the AR Anchor is really useful in that case since it will allow us to keep the Game Object at the same position in the real world even if we reset the view of the headset.

To be able to use the AR Anchors we first need to add the AR Anchor Manager Component to the XR Rig. This Component will allow us to create and manage the AR Anchors in the scene. Then at the beginning we thought that we could just add the AR Anchor Component to the Game Object that we want to anchor. But we realized that it wasn't recommended and instead we should use a script that will create the AR Anchor using an async method. So we created the following script that will create an anchor the Game Object it's on.

```csharp
using UnityEngine;
using UnityEngine.XR.ARFoundation;

public class ARAnchor_object : MonoBehaviour
{
    [SerializeField]
    private ARAnchorManager _anchorManager;

    void Start()
    {
        Pose pose = new Pose(transform.position, transform.rotation);
        CreateAnchorAsync(pose);
    }

    private async void CreateAnchorAsync(Pose pose)
    {
        var result = await _anchorManager.TryAddAnchorAsync(pose);

        if (result.status.IsSuccess())
        {
            var anchor = result.value as ARAnchor;

            transform.SetParent(anchor.transform);
            transform.localPosition = Vector3.zero;
            transform.localRotation = Quaternion.identity;
        }
    }
}
```

Listing 1: AR Anchor Script

This script will create an AR Anchor at the position and rotation of the Game Object when the application starts. The AR Anchor Manager Component is used to create the AR Anchor and then the Game Object is set as a child of the AR Anchor. This way the Game Object will stay at the same position in the real world even if we reset the view of the headset.

2.2 Hand Gestures

Now that we talked about all the basics of a mixed reality application we can talk about the hand gestures. Those gestures will be used to trigger different functionalities in the application. By default, some gestures are already implemented like the pinch gesture that be used to grab objects and also interact with the UI from afar. Those gestures are implemented in the XR Interaction Toolkit package. But we can also create our own gestures to trigger different functionalities in the application. The hand gesture that we need to add to our application is one to access a pause screen. By default, on the Meat Quest headset left controller there is a button that is usually used to open the pause menu of the application, this is universal across all applications. This button is present on the controller but there is also a default hand gesture that is used to trigger the same functionality. This gesture is the “Pinch” gesture on your left hand when you have your palm facing upward. Even though it’s supposed to trigger the same thing it doesn’t seem that it actually send the same event as the button on the controller. So we need to implement our own gesture to trigger the pause menu ourselves. To be able to create our own gestures they are two things that we need to understand.

The first thing that we need to use is to create a XR Hand Shape. This is used to configure what the shape of the user's hand should look like. You just need to create a new XR Hand Shape in the project and then you can configure it. The configuration is pretty simple, you just need to first choose a finger and then select which shape should be selected in the following list that is available in their official documentation [4].

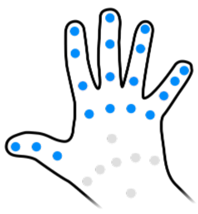
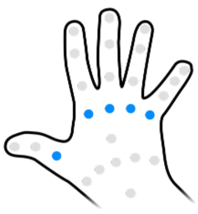
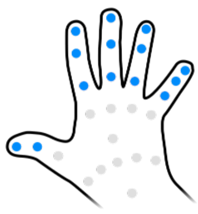
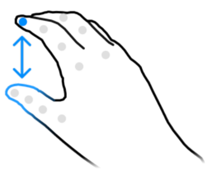
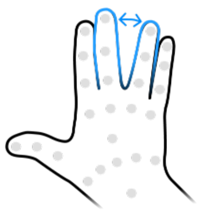
Full Curl	Base Curl	Tip Curl	Pinch	Spread
				

Table 2: XR Hand Shape Configuration

Then on the shape we chose we can tweak the percentage of that shape that we want to use. So for our example the shape that we need is on the index with the shape Pinch. We can just use a percentage of 100% for the Pinch shape. We could also put every other finger with a Full Curl at 0% to have a flat hand but it doesn't really matter for us what the other fingers are doing since we only care about the index finger. So now we have our XR Hand Shape that we can use to detect the hand gesture.

After setting up the XR Hand Shape we need to also set up the XR Hand Pose. This is used to configure the position and rotation of the hand in the real world. The XR Hanhd Pose takes a XR Hand Shape in input and then we can configure the rotation of the hand the real world. We first need to configure the axis that we are going to check. There are the three following axis that can be chosen from.

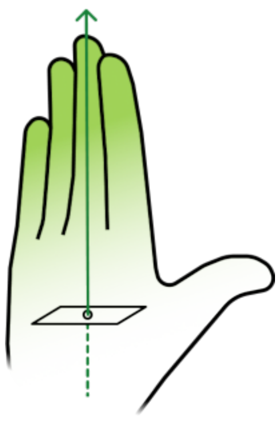
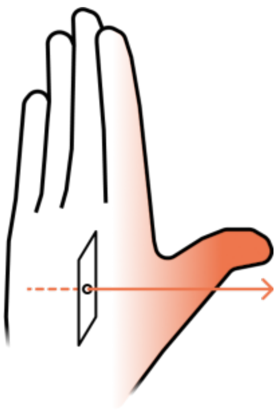
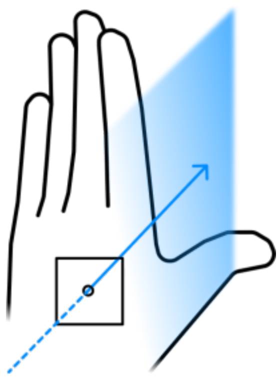
Fingers Extended Direction	Thumb Extended Direction	Palm Direction
		

Table 3: XR Hand Pose Axis Configuration

Since we want the pause menu to launch when the user is pinching their index when the palm is facing upward, we need to use the **Palm Direction** axis. Then after that we need to also choose the reference direction and once again they are multiple options that can be chosen. Those options are available on their official documentation [5] and are the following.

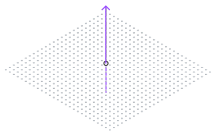
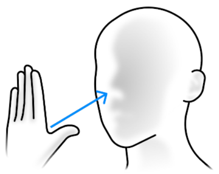
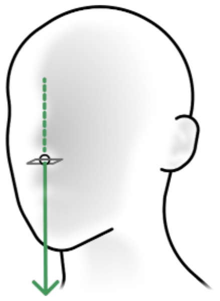
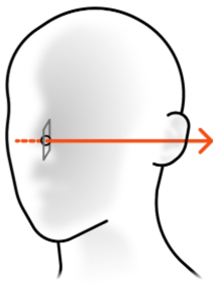
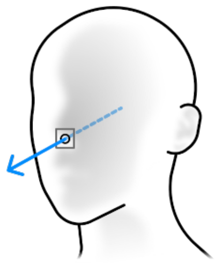
Origin Up	**Hand to Head**	**Chin Direction**	**Ear Direction**	**Nose Direction**
				

Table 4: XR Hand Reference Direction

So for the gesture that we want to add we can choose origin up. Then we can finally set up an angle of tolerance that the gesture will have. Since we don't really need to be that precise we just put a 90° angle so that way even if the hand is being perpendicular to the floor the hand gesture will still be able to be detected.

Now that the gesture that we need is configured we can start to set up away to launch the pause menu with this new gesture. For that we first need to create a Game Object that has the "Static Hand Gesture" script. This script is provided by XR Hands and is used to say what should happen when a certain hand does a certain gesture. For that we just need to put in the "Hand Tracking Events" field the tracking event that correspond to the left hand the one that is tracking the left hand. And in the "Hand Shape or Pose" field we just need to put the XR Hand Pose that we created earlier. Then we can just add a new event to the "On Hand Gesture" field and link it to the function that will launch the pause menu.

2.3 UI Menus

2.3.1 UI following the user

Now that we can open the pause menu with a hand gesture we need to make it. We talked earlier about the fact that the XR Rig doesn't move with the user in a Mixed Reality application. So we need to find a way to be able to have the UI following the user. They are two different ways that the UI could follow the player. The first one is that the UI is just following the position of the Camera without taking the rotation of it into account. The other way is to have the UI following the position and rotation of the Camera. The first is used for UI that are just for information, this is used for the tutorial in our case. The second way is used for UI that are used to interact with the user, like the pause menu. So we just made a Game Object that will have all of our UIs as children and that will follow the Camera without taking the rotation into account. And also the first child of this Game

Object will also take into account the rotation of the Camera. That is what the following script is doing.

```csharp
using UnityEngine;

public class Fix_Position_to_Camera : MonoBehaviour
{
    [SerializeField]
    private Camera _camera;

    [SerializeField]
    [Range(0f, 1f)]
    private float _distance = 0.5f;

    private Transform _rotateObject;

    private bool _tracking = false;

    private Vector3 _position;
    private Vector3 _positionRotation;
    private Quaternion _rotation;

    private void Start()
    {
        _rotateObject = transform.GetChild(0);
        Check_Application_Started.GetEvent().AddListener(StartTracking);
    }

    void Update()
    {
        if (Debuging.IsDebuging() || _tracking)
        {
            TrackingCamera();
        }
    }

    private void TrackingCamera()
    {
        transform.position = _position + _camera.transform.position;

        _rotateObject.rotation = _rotation;
        _rotateObject.position = _positionRotation + _camera.transform.position;
    }

    public void StartTracking()
    {
        GameObject obj = new GameObject("Get Position");
        obj.transform.parent = _camera.transform;
        obj.transform.localPosition = Vector3.zero;
        obj.transform.up = Vector3.up;

        _position = obj.transform.TransformPoint(0, 0, _distance);
        _position -= _camera.transform.position;
        _positionRotation = _camera.transform.forward * _distance;
        _rotation = Quaternion.LookRotation(_positionRotation);

        _tracking = true;
        Destroy(obj);
    }

    public void StopTracking()
    {
        _tracking = false;
    }
}
```

Listing 2: Camera following script

Now that we can have the UI following the Camera we can just create the pause menu.

2.3.2 Pause Menu

The pause menu is a UI that will allow the user to pause the application and access some functionalities. The pause menu is composed of multiple buttons that will allow the user to access different functionalities. The first button is the “Resume” button that will just close the pause menu and resume the application. The second button is “Recalibrate” this one is used to recalibrate the rooms to the AR Floor. This is used because when the user is removing the headset for a reason or another and then put it back on the position of the AR Floor doesn’t change but the virtual environment did so everything is in the wrong place. So this button will just put everything back in it’s place. This is a small explanation of this button but to get more information about it you can go read the internship report of my college since he was the one that worked on this part of the project. The third button is the “Debug” button. This one is used to open the debug menu. We will go into more details of this menu in the future.

We talked about the buttons and what they should do but we didn’t talk about how a button would work. First we looked at the Hand Demo Scene from the XR Interaction Toolkit package. This scene contains a lot of different buttons that are used to interact with the application. So we just took one of those buttons and modified it to fit our needs. The first thing that is noticeable is that the Game Object that has the Canvas Component has a “Graphic Raycaster” Component and a “Tracked Device Graphic Raycaster” Component. Those two Components are used to allow the UI to be interacted with the hands and the controllers. The “Graphic Raycaster” Component is used to allow the UI to be interacted with the controllers and the “Tracked Device Graphic Raycaster” Component is used to allow the UI to be interacted with the hands. So we just need to add those two Components to our Canvas Game Object and then we can just use the buttons that are already present in the scene. So then we can just add all the buttons that we want and use them to trigger the functionalities that we want.

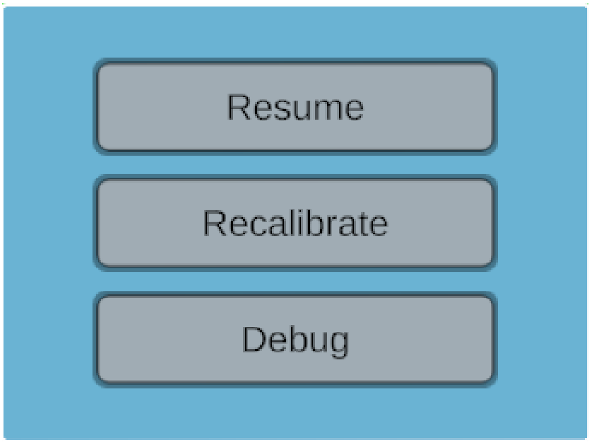


Figure 6: Pause Menu UI

2.3.3 Debug Menu

Now that we have a pause menu we will talk about the debug menu. The debug menu will allow us to access some functionalities that are used for debugging purposes.

At first we have a checkbox that will allow us to Show all the AR Planes present in the scene. Actually even though the AR Planes are generated those planes are set up to not be shown in the scene. That way the user will not be able to see them and will not be confused by them. But for debugging purposes we can just check this box and then all the AR Planes will be shown in the scene. This is really useful to see if the AR Planes are generated correctly and if they are positioned correctly in the real world.

After we have the checkbox “Show All Rooms”, this one correspond to all the rooms that we haded inside our Unity project. This is used to see if the rooms are positioned correctly in the real world.

The last checkbox that we have is “Show Logs”. This one we will enable a new UI that will follow the player’s camera position and orientation. This UI will display all the logs that are generated by the application. This is really useful to see if there are any errors or warnings in the application and to debug them.

Next there is the button “Print Floor Mesh”, when you press it, it will print in the logs the information of the mesh of the AR Floor you are currently above. This is useful because we need to generate inside the Unity project a floor that has the same Mesh as the AR Floor. So this button will allow us to get the information of the AR Floor and then we can use it to generate the floor in the Unity project.

Then the last button is the “Position in world”. This button, when pressed, will print in the logs the position of the Camera but as it would have been before moving virtual environment to correspond to the real world. It is useful to know those coordinates when you want to place a new object in the Unity project. It was used to place the doors in the right position in the virtual environment.

So we get the following debug menu that will allow us to access those functionalities.

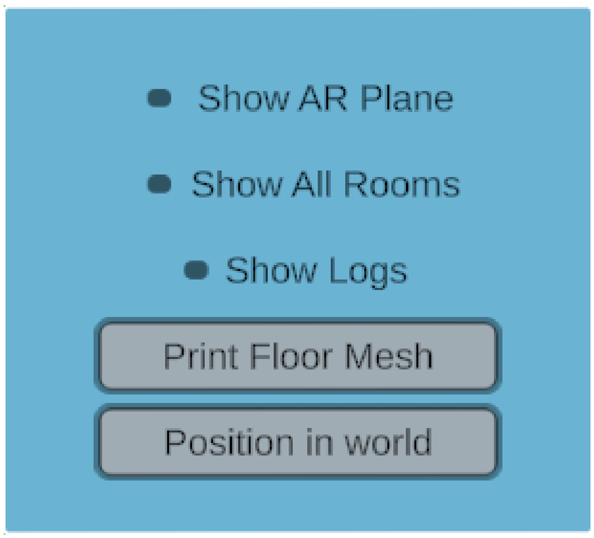


Figure 7: Debug Menu UI

For the UI we talked about checkboxes but we didn’t talk about how to implement them. It’s actually pretty simple since we can just use the Toggle Component that is provided by Unity. This Component will allow us to create a checkbox that can be checked or unchecked. Then in the “On Value Changed” field we can just add a new event and link it to the function that will be called when the checkbox is checked or unchecked. This way we can just use the Toggle Component to create checkboxes that will trigger the functionalities that we want.

Now that we have the debug menu we are going to talk a little bit more about the logs that are displayed in the UI. The logs are generated by the application and they are used to debug the application. The logs are generated by using the Debug class from Unity. This class allows us to generate logs, warnings and errors in the application. Those logs will be then displayed into a custom UI that will follow the player's camera position and orientation. This UI will display all the logs that are generated by the application. The logs are displayed in a scroll view that will allow the user to scroll through the logs and see all the information that is generated by the application. This UI Also has at the top of it the current fps of the application. So we have the following UI that will display the logs and the fps of the application.

Figure 8: Logs UI

3 Exit Pathway

Now that we talked about all the basics and everything that could be used in any mixed reality project we can start to talk about thins are more specific with our project. The project was divided in two big parts. The first one was to make the virtual environment inside the Unity project correspond to the reality. For that we have used the AR Floor and make a copy of it inside Unity and then we had to just make those two overlaps. That way we could easily make the virtual objects appear in the right place. This part was done by my colleague and to read more precisely about it you can read his report. The second big part of this project was to make the rooms and the doors that corresponds to the real ones. Those doors and rooms would be used to guide the user from door to door until he reaches the emergency exit. In this chapter we will talk about how we made those rooms and doors and how we used them to guide the user.

3.1 Rooms

The Rooms that we made are the ones that correspond to the real ones. They are divided in three types: the normal ones, the player rooms and the exit rooms. On the following image you can see the three types of rooms that we made. In yellow it is the player room, in red the exit room and in green the normal rooms. The player room is the one where the user is located at the beginning of the experience and the exit room is the one that he has to reach to finish the experience. The normal rooms are all the other rooms that are not player or exit rooms. This figure is a simplified version of the real world rooms of our project.

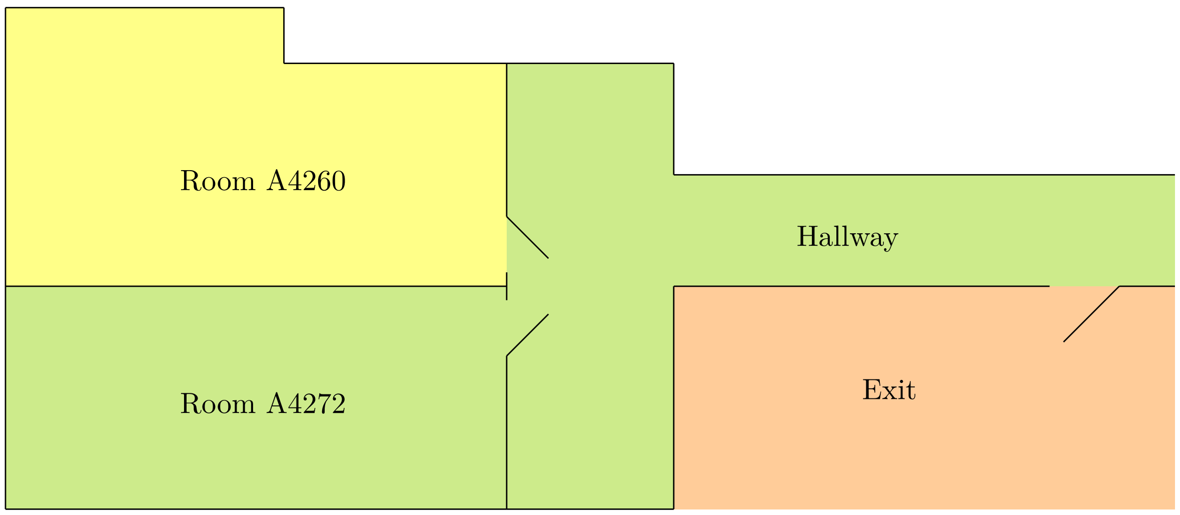


Figure 9: The three types of rooms

The normal rooms are the base of all the other types of rooms. So we are going to start by talking about them.

3.1.1 Normal Room

A normal room is a room that has no special function. The room consist on only a list of doors that connect it to other rooms. We didn't give the rooms any floor or 3D model since it would have taken us a lot of time to make them, and it would not add anything to the experience since the user doesn't have any object that could interact with the room's floor or walls. So a room consist of two simple things: a name and a list of doors. The name is used to identify the room and the list of doors is used to connect it to other rooms. The doors are the ones that will be used to guide the user from one room to another. A room just like that can't do much, all of its core functionalities are attached to the doors.

3.1.2 Player Room

Now we are going to talk about the player room. The player room can be just any room. This room can be any room that is not an exit room. The player room is where the player is currently located. This room always changes when the player goes through a door. The player room is used to keep track of the player's current location and to know which doors are available to the player.

The player room has also another feature, it is that it will change the textures of the doors that are connected to it. This way when the doors are visible when he check the "Show All Rooms" in the debug menu, the doors will have a different texture than the ones that are not supposed to be in the room of the user. This way we were able to verify if the player was in detected in the right room or not. There is also a special texture for the doors that are exit doors. Those are the three types of textures that we have for the doors: the normal doors, the player room doors and the exit doors.

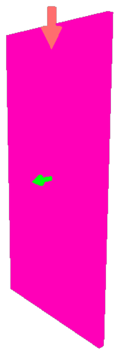
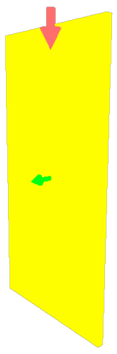
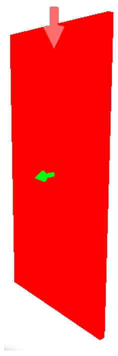
Normal Door	Player Room Door	Exit Door
		

Table 5: The three types of doors

At the beginning of the experience we can't know the player room since we don't even know where the player is. When we use the AR Floor to detect in which room the player is located, we can set the player room to the room that corresponds to the AR Floor. This way we can start the experience in the right room and guide the player from there.

3.1.3 Exit Room

Finally, we have the exit room. This one is more special than the others. The exit room is a room that doesn't really exit in the real world. It is just a room that was created because to have a room that is common between all the other rooms that contains the emergency exit that we want the player to reach. The need for this room to exist is because we want to have a single door at the end that will be linked to all the emergency exits. This door usefulness will be developed in the future when we will talk about the connections between the doors.

3.2 Doors

A door is a connection between two rooms. It is the part that has an element showed to the user and that guides him from one room to another until he reaches the exit. It is composed of multiple elements. First there is the door itself with its collider that is used to detect when the player is going through it. Then there the arrow, this arrow is used to indicate wich room is considered to be the room forward from the room backward. This is just for the person that is placing a door and want to know which room to select as the forward room. Finally, there is the arrow above the door. This arrow is used to indicate that the user need to go through this door and not another one. This arrow is the one that will be visible to the user when he is in the room and will guide him to the next room. This arrow also has an animation that will make it move up and down to attract the user's attention.

3.2.1 Indications

We are going to talk about the animation of the arrow more precisely in this part. The arrow is animated to move up and down to attract the user's attention. To be able to do that we have used the Unity's animation system. For that we have created the following animation clip.

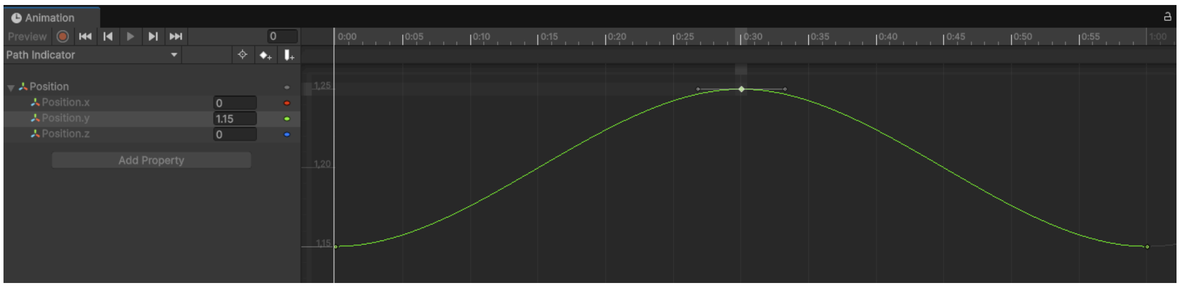


Figure 10: Animation clip of the arrow

This clip as we can see modify the y-axis of the arrow's position to make it move up and down. But this clip alone doesn't do anything. We need to use it in an animation controller to be able to use it in the game. An animation controller is a Unity component that allows us to control the animations of a Game Object. The animation controller is used to define the different states of the animation and how to transition between them. In our case, don't want to have our animation to be played automatically at the beginning. So we have created a state called "Idle" that is the default state of the animation controller. This state is used to keep the arrow in its initial position. Then we have created another state called "Path Indicator" that is used to play the animation of the arrow moving up and down. We don't have to draw any transition to the "Path Indicator" state since we will use a script to change the state of the animation controller to have is play the animation when the player needs to go through this door. We also added a float parameter that we called "Animation speed" that is used to control the speed of the animation. This parameter will have is value changed by the script to make the animation play with a specific speed. The animation controller used is the following one.

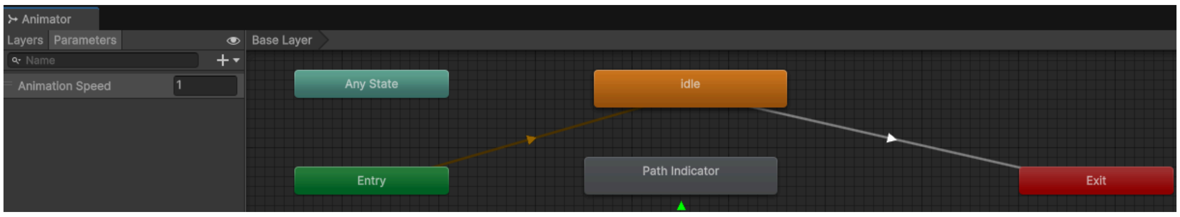


Figure 11: Animator controller of the arrow

Now we just had to do a script that will take in input a float to control the speed of the animation and that will change the state of the animation controller to play or stop the animation of the arrow. This script will be called by forwarded to the door Game Object and it will be called when we want the player to go through this door.

There is also a second thing that is implemented to indicate to the user to go through that door. Because the arrow is above the door it can be difficult to see it when the user is not looking in the direction of the door and it's even more difficult to see it when the door is far away. So to solve that at the feet of the user there is a compass that will indicate the direction of the door. This compass has a simple script use the Unity function named “LookAt” [6] to always look at the door that the user has to go through. Since LookAt also change the X and Z axis of the compass, we had to reset them to 0 so that way the arrow will always be pointing in the right direction but stay flat on the floor.

3.2.2 Going Through

Now that we can indicate the door to the user with the arrow and the compass, we need to be able to detect when the user is going through it. For that we have used the Unity’s physics system.

We have added a collider to the Game Object that represents the door. We also added another collider to the player’s Camera. That way we will be able to detect when the player’s Camera comes in contact with the door. To use that detection we attached to the door a script that will used the Unity’s OnTriggerExit function to detect when the player’s Camera exits the door’s collider. We want to detect when the player exits the door’s collider and not when he enters it because we want to be able to detect when the player is going through the door and not just when he is near it. Even though we could just consider that when the player exits the door’s collider he is always changing room so we could just get the player’s current room and change it to the other one that the door is connected to, that way of doing things can cause some problems. For exemple if in the middle of the door the user remembers that he forgot something in the previous room and goes back to get it, we don’t want to change the player’s room to the previous one since he is not going through the door. To solve that we have to do something a little more complicated. When the player’s camera collider exits the door’s collider, we will get the coordinates of the camera and just look at the angle between the camera and the door. If the angle is between $-90°$ and $90°$ then the player is in the room forward and in the contrary then the player will be in the room backward. This way we can know if the player is going through the door or not and just change the player’s room in consequence. When obtain the following function to do that.

```csharp
public void DoorCollision(Collision collision)
    {

        if (collision.transform.CompareTag("Player"))
        {


            Room newRoom = null;
            var playerRoom = collision.rigidbody.GetComponent<Player_Room>();
            Vector3 directionToTarget = transform.position - collision.transform.parent.position;
            float angle = Vector3.Angle(transform.forward, directionToTarget) % 360f;
            bool nothingToDo = true;
            bool roomForward = false;

            if (angle < 90f && angle > -90f && !(playerRoom.GetLastDoorUsed() == this && playerRoom.GetLastDoorUsedForward() == true))
            {
                Debug.Log("Door behind me, angle: " + angle);
                newRoom = _roomForward;
                roomForward = true;
                nothingToDo = false;
            }
            else if(!(playerRoom.GetLastDoorUsed() == this && playerRoom.GetLastDoorUsedForward() == false))
            {
                Debug.Log("Door in front of me, angle: " + angle);
                newRoom = _roomBackward;
                nothingToDo = false;
            }

            if(newRoom == null && _exit)
            {
                Debug.Log("Launching end of program");
                End_Program.GetStopEvent().Invoke();
            }
            else if(!nothingToDo)
            {
                playerRoom.SetCurrentRoom(newRoom, this, roomForward);
            }
        }
    }
```

Listing 3: Function to detect if the player is going through the door

This function also has a special case when the player is going through the exit door. In that case we just call the End_Program event that will stop the experience and show the user that he has reached the end of the experience. We also verify that the player is not going through the same door that he just used to avoid changing the player's room when he is just going back to get something in the previous room. This way we can detect when the player is going through a door and change his room accordingly.

3.2.3 Connections

Now that everything is set up for the door there is still one last thing that needs to be done with them. Since we will use the Dijkstra algorithm to guide the user from the player room to the exit room, we need to connect the doors to each other to arranged them into a graph. To also use Dijkstra we need to have a weight for each connection. The weight is used to determine the cost of going through a door. In our case, we will use the distance between the two doors as the weight. This way we can use Dijkstra to find the shortest path from the player room to the exit room. With the configuration that we have shown earlier we can have the following graph.

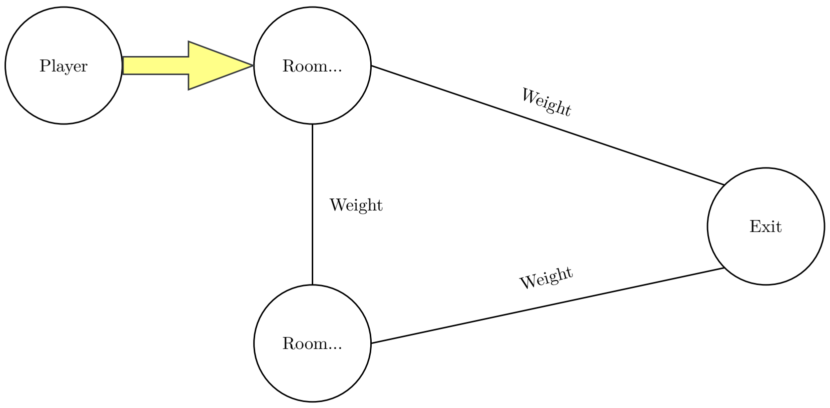


Figure 12: Graph of the doors

To calculate the weight between two nodes of this graph we have made the following script that will be attached to the room Game Object. It first calculates the distance between every doors inside the room and then it will put those into a dictionary that we can use to get the weight between two doors.

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

public class Graph_link : MonoBehaviour
{
    private static float _weightDistanceFactor = 1f;

    [SerializeField]
    private bool _isWeightNull = false;

    private Dictionary<(Door, Door), float> _weight;

    void Start()
    {
        Door[] doors = GetComponent<Room>().GetDoors();
        int size = doors.Length;

        _weight = new Dictionary<(Door, Door), float>();

        for (int i= 0; i < size; i++)
        {
            for(int j = 0;  j < i; j++)
            {
                if (!_isWeightNull)
                {
                    _weight.Add((doors[i], doors[j]), _weightDistanceFactor * Vector3.Distance(doors[i].transform.position,
                    doors[j].transform.position));
                    _weight.Add((doors[j], doors[i]), _weightDistanceFactor * Vector3.Distance(doors[i].transform.position,
                    doors[j].transform.position));
                }
                else
                {
                    Debug.Log("Creating link between: (" + doors[i] + ", " + doors[j] + ")");
                    _weight.Add((doors[i], doors[j]), 0);
                    _weight.Add((doors[j], doors[i]), 0);
                }
            }
        }

    }

    public float GetLinkWeight(Door door1, Door door2)
    {
        return _weight[(door1, door2)];
    }
}
```

Listing 4: Script to calculate the weight between two doors

We also have a special case for this script. If the _isWeightNull variable is set to true, then the weight between two doors will be set to 0. This is used for only one room, the exit room. The exit room is the one that connects every emergency exits together. But to use the Dijkstra algorithm later we need to have one single node as the beginning. But with our current set up if we consider the beginning as the player room then it can have multiple doors connected to it. And if we consider the beginning as the exit room then it can have multiple doors connected to it also. So to solve this problem inside the exit room we added a door that correspond in reality to nothing. This door is just a placeholder that will be used to connect every emergency exits to this one. That way we can have a single node as the beginning of the Dijkstra algorithm. So to have the Dijkstra algorithm work we need to have the weight between every door of the exit room to be 0. This way we can have a single node as the beginning of the Dijkstra algorithm, and it will be able to find the shortest path from the player room to the exit room. So in reality the graph of the doors will look like this.

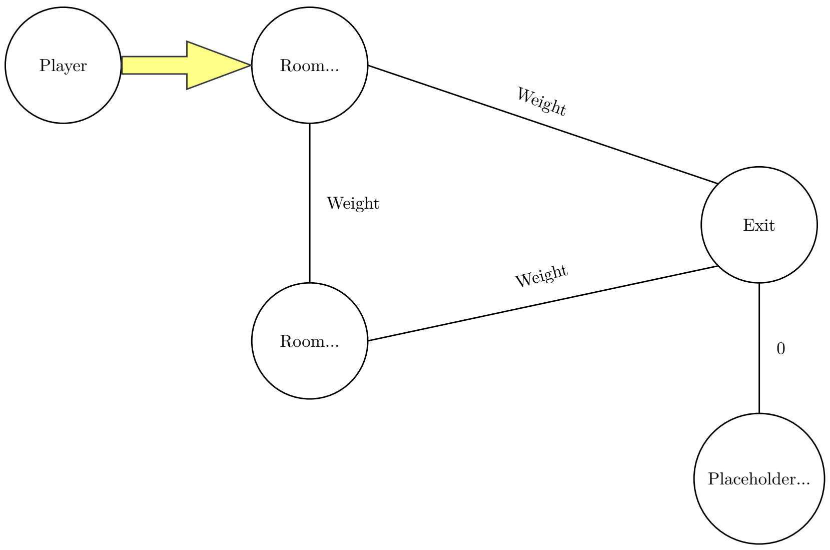


Figure 13: Graph of the doors with the exit room

3.3 Dijkstra

Now that we have all the rooms and doors set up to make a graph, we will be able to use the Dijkstra algorithm to find the shortest path from the player room to the exit room. The Dijkstra algorithm is a graph search algorithm that finds the shortest path between two nodes in a graph.

We will first explain how this algorithm works and then we will show how we implemented it in our project. To explain each step we will use the following graph as an example.

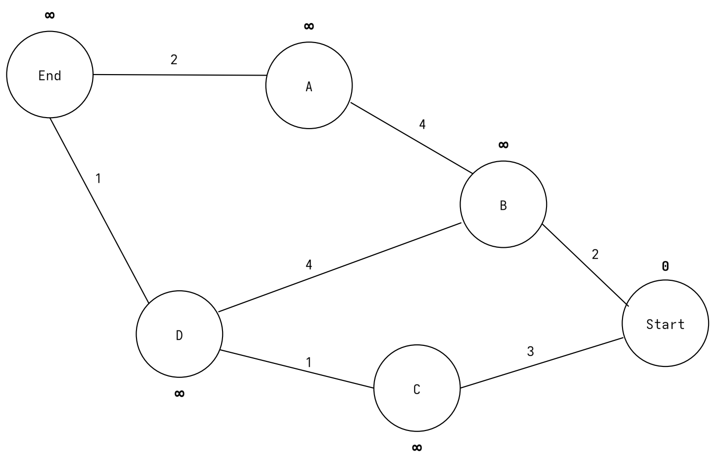


Figure 14: Graph example for the Dijkstra algorithm (Initial Step)

First each node of the graph is initialized with a distance of infinity, except for the starting node which is initialized with a distance of 0. After that we look at the neighbors of the starting node and we update their distance if the distance to the starting node plus the weight of the edge between the starting node and the neighbor is less than the current distance of the neighbor. We will also update the previous node of the neighbor to the starting node. This way we can keep track of the path that we took to reach the neighbor node. After that we will mark the starting node as visited and we will look for the next node to visit. So we finally obtain the following graph.

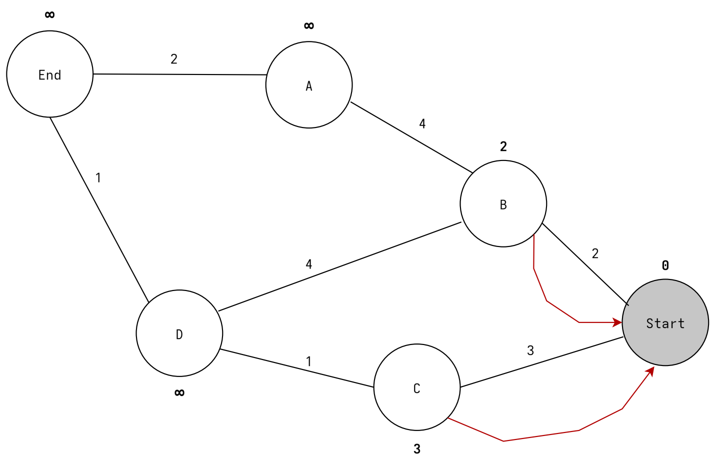


Figure 15: Graph example for the Dijkstra algorithm (Step 1)

Now we will look for the next node to visit, which is the one with the smallest distance that is not visited yet. In our case, it is node B. We will then repeat the same process as before, looking at the neighbors of node B and updating their distance if necessary. We will also mark node B as visited. After that we will have the following graph.

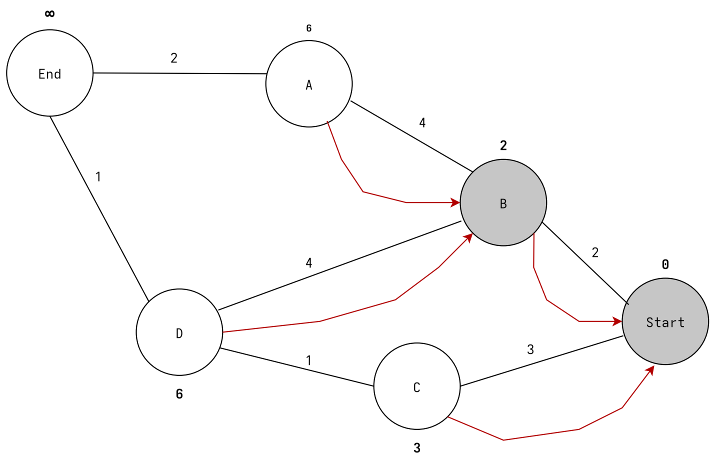


Figure 16: Graph example for the Dijkstra algorithm (Step 2)

We will then look for the next node to visit, which is node C. We will repeat the same process as before, looking at the neighbors of node C and updating their distance if necessary. We will also mark node C as visited. After that we will have the following graph.

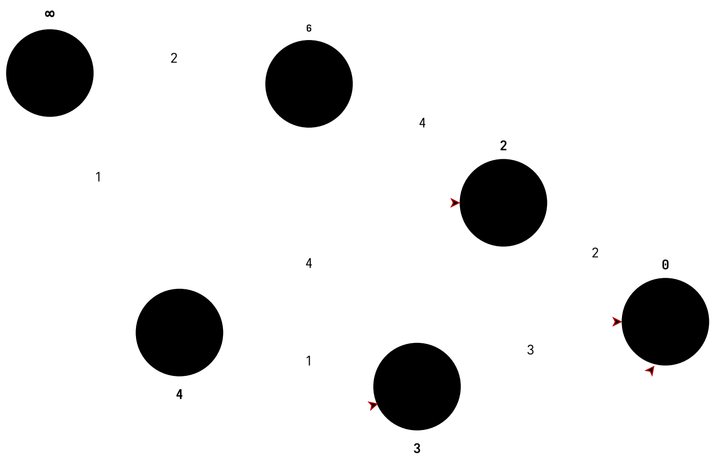


Figure 17: Graph example for the Dijkstra algorithm (Step 3)

We can observe that the node D has been updated with a new distance. We will then look for the next node to visit, which is node D. We will repeat the same process as before, looking at the neighbors of node D and updating their distance if necessary. We will also mark node D as visited. After that we will have the following graph.

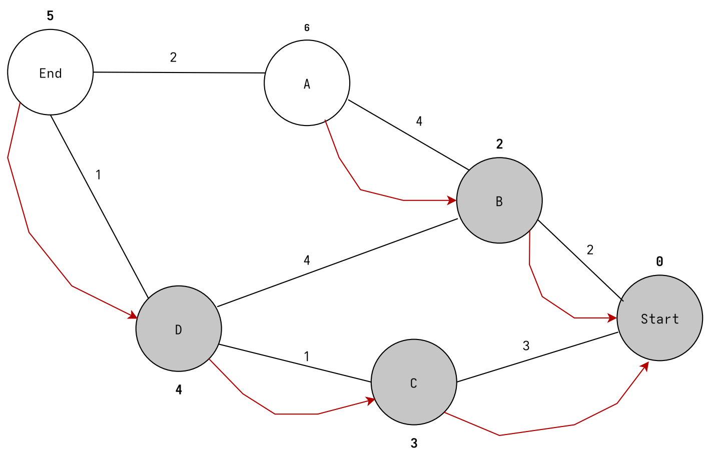


Figure 18: Graph example for the Dijkstra algorithm (Step 4)

We can see that now the end node has been updated with a distance. Even though we could stop here, the algorithm will continue until every node has been visited. We will skip the next steps since they are not really important for the understanding of the algorithm. But we will show the final graph after all nodes have been visited.

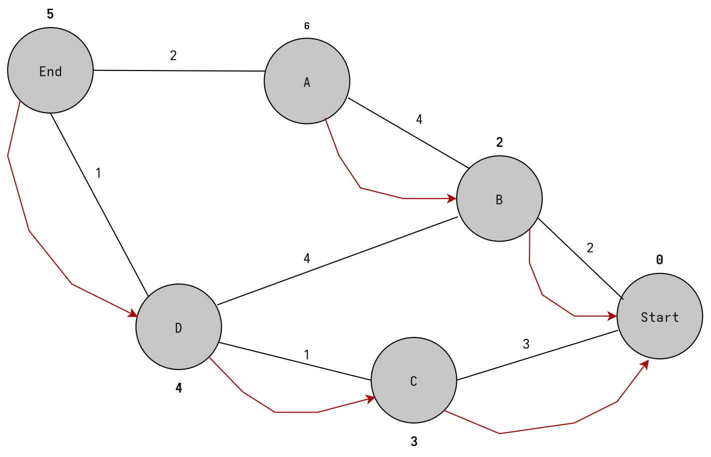


Figure 19: Graph example for the Dijkstra algorithm (Final Step)

Now that we have the final graph, we can see that we have the shortest path from the starting node to the end node. We can also see that we have the previous node of each node, which will allow us to reconstruct the path that we took to reach the end node. To reconstruct the path, we will start from the end node and we will follow the previous nodes until we reach the starting node. This way we can obtain the path that we took to reach the end node.

Now that we have explained how the Dijkstra algorithm works the implementation of it in our project is the same. There is just one slight difference. The end node doesn't really exist in our project. They are actually multiple end nodes that are available. Those corresponds to all of the doors that are accessible to the user inside their current room. And the starting node is the placeholder exit that we had talked about earlier.

Once we have the path from the starting node to the end node, we can use it to guide the user from the player room to the exit room. We will store the information of the path in a list of doors that will be used to guide the user. Then when the user go through a door, we will check if the door is the next one in the path. If it is, then we will update the path to the next door in the list. If it is not, then we will verify if the door is the previous one in the path if their can be a previous one. If it is, then we will just go back to the previous door in the path. And finally if the door is not the next one or the previous one in the path we will consider that the user is lost and we will rerun the Dijkstra algorithm to find a new path from the player room to the exit room. This way we can guide the user from the player room to the exit room and make sure that he is always on the right path. Every time that we know that we need to go through a door we will call the function to make the compass point to the door and the arrow above it will start to animate. This way we can guide the user to the next door that he has to go through.

Conclusion

This internship at UiT – The Arctic University of Norway has provided a valuable opportunity to explore the field of mixed reality. I was familiar with virtual reality, but this project allowed me to improve my skills in the virtual reality field, but also learn about mixed reality. The project was a significant challenge, especially considering the limited time available to complete it.

The project, will need to be further developed in the future since the current implementation is only a foundation. In the current state multiple things can be improved and the most important one is the environment detection. Right now the application use only the position of one predetermined room to place the whole environment. While it works it cause some issues when the user gets away and away from the origin point. In the future we might hope for a headset that coud maybe store internally hundreds of rooms and ther position in the real world. Or maybe there will be a development in the Open XR that will allow the headset to detect the environment while the application is running. This would allow the application to adapt to the user's environment and provide a more immersive experience without needing to recreate the whole environment inside the project.

The project has also provided a solid foundation for future development, particularly in the area of user interaction and navigation. The implementation of hand gestures for interaction is a significant step towards creating a more intuitive user experience in mixed reality. The debugging features added during development will also facilitate future improvements and help identify potential issues more easily.

Overall, this internship has not only enhanced my technical skills but also deepened my understanding of mixed reality applications. I look forward to seeing how this project evolves and contributes to the field of mixed reality in the future.

A | Glossary

1.a Concepts

MR - Mixed Reality

Mixed Reality is a blend of physical and digital worlds, allowing users to interact with both simultaneously. It encompasses technologies like augmented reality (AR) and virtual reality (VR), providing immersive experiences that enhance real-world environments with digital information and objects.

1.b Unity

Component

A Component in Unity is a piece of functionality that can be attached to a Game Object. Components define the behavior and properties of Game Objects, such as rendering, physics, and user interactions. Examples include Mesh Renderer, Rigidbody, and Collider.

Game Object

In Unity, a Game Object is the fundamental entity in a scene. It can represent anything from a character to an environmental element, and it can have various components attached to it, such as meshes, colliders, and scripts, which define its behavior and appearance.

B | Bibliography

[1] UiT The Arctic University of Norway, "UiT The Arctic University of Norway." [Online]. Available: https://en.uit.no/startsida

[2] R. BOUDRIE & S. ABDUL-SALAM, "Emergency Evacuation." [Online]. Available: https://github.com/SamiLumine/Emergency-Evacuation

[3] AR Foundation, "AR Foundation Documentaion." [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@6.1/manual/index.html

[4] XR Hands, "Finger shape." [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.hands@1.4/manual/gestures/finger-shapes.html

[5] XR Hands, "Hand orientation." [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.hands@1.4/manual/gestures/hand-orientation.html

[6] Unity, "Transfrom.LookAt." [Online]. Available: https://docs.unity3d.com/ScriptReference/Transform.LookAt.html