Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Escape Game with Augmented Reality

Peter Vataščin

Supervisor: prof. Ing. Jiří Žára, CSc.
May 2024

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Vataščin Peter** Personal ID number: **507285**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Specialisation: **Computer Games and Graphics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Escape Game with Augmented Reality

Bachelor’s thesis title in Czech:

Úniková hra s rozšířenou realitou

Guidelines:

1) Make research on existing augmented reality systems in escape rooms/games. Describe their main features, specify a list of requirements, and compare the systems accordingly.
2) Based on the output of the previous item, choose suitable tools/libraries and verify the correctness of your choice on partial tasks (augmented reality, network communication).
3) Design at least 3 scenarios that require the cooperation of several players, e.g., one of them uses mobile as a virtual light source or pointer, and the other sees on his mobile only those virtual objects that are "illuminated" by the first mobile.
4) After agreement with the supervisor, implement the selected scenarios into a simple but complete game. Concentrate on robustness and user-friendliness. Compare your solution with systems from item 1). Test the final application with at least five users.
5) Create clear instructions and recommendations for potential escape game creators who would like to use the developed application in practice.

Bibliography / sources:

1) Warmelink H. et al.: AMELIO: evaluating the team-building potential of a mixed reality escape room game. Extended abstracts publication of the annual symposium on computer-human interaction in play. ACM, Amsterdam, Netherlands, pp. 111–123, 2017.
2) Plecher D. et al.: Designing an AR-Escape-Room with Competitive and Cooperative Mode. GI VR / AR Workshop, DOI: 10.18420/vrar2020_30, pp. 24.-25, 2020.

Name and workplace of bachelor's thesis supervisor:

prof. Ing. Jiří Žára, CSc. Department of Computer Graphics and Interaction FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **24.01.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

prof. Ing. Jiří Žára, CSc. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my gratitude to prof. Ing. Jiří Žára, CSc. for his invaluable guidance, insightful feedback, and support throughout the development of this thesis.

I am also thankful to all those who participated in testing the game prototype developed for this research. Your thoughtful feedback and constructive criticism have been immensely valuable.

Declaration

I declare that this thesis is a work of my own and that all the sources used are cited and referenced.

Prague, 25 May 2024

Abstract

This bachelor's thesis describes the process of creating an augmented reality multiplayer escape room mobile game. First, it compares existing AR games with similar features. Then it analyses tools for creating AR games and network multiplayer games. The development process of the game is then described in detail. The game is tested on eleven people and the results are analysed. Finally, this thesis includes a manual on how to build on top of the game in the Unity game engine.

Keywords: Augmented reality, Escape Rooms, Escape games, AR Foundation, Unity, Multiplayer Game, Netcode for GameObjects

Supervisor: prof. Ing. Jiří Žára, CSc. Department of Computer Graphics and Interaction

Abstrakt

V této bakalářské práci je popsán postup vytváření únikové hry pro více hráčů v rozšířené realitě. Nejprve jsou porovnány podobné existující hry. Dále jsou analyzovány vývojářské nástroje pro tvorbu her s rozšířenou realitou a her pro více hráčů po síti. Dále je detailně popsán vývoj této hry. Hra je testována na jedenácti lidech a výsledky jsou analyzovány. Nakonec tato práce obsahuje manuál, jak rozšířit tuto hru v herním enginu Unity.

Klíčová slova: Rozšířená realita, Únikové hry, AR Foundation, Unity, Hra pro více hráčů, Netcode for GameObjects

Překlad názvu: Úniková hra s rozšířenou realitou

Contents

Figures

Tables

Chapter 1

Introduction

Escape rooms are a very popular pastime nowadays. The demand for such games is high and there are several dozen of them in Prague alone with various themes. However, the physical world is limited and therefore many variations are being created, using modern technology, to extend the experience into the virtual world.

The most common adaptation of escape rooms is computer games, which make this genre of entertainment accessible to audiences who cannot experience it for financial reasons or because of its unavailability in a given location. Such games often utilize the fact that there are no physical constraints in the virtual world giving them more creative freedom to entertain the user. As seen in the popular game series Portal, where the players encounter all sorts of puzzles impossible to be built in the real world, most notably having portals connecting two places as if there was a doorway between them. These games often have a deeper story and longer playtime, thanks to not being time-limited by customer capacity. Despite this, a computer game feels significantly different since the player is detached from the game world, only following it through the computer's monitor. What's missing is being locked in a room, trying to solve all sorts of puzzles with your friends, and being under stress because the clock is ticking and you have just one try.

A way to have both the immersion of a real-life escape room, and the freedom from physical constraints is using virtual reality, or VR for short. It is a technology that captures the movement of a player and synchronises it with the virtual world of a computer game. This is most often done using a VR headset, which tracks the motion of the player's head and projects the image from the game onto two screens right in front of each of the player's eyes. This creates the illusion of the player being inside a computer game. A VR headset is often accompanied by two controllers which the player holds so they can interact with the virtual world as if they were using their own hands. The players can either all be in one room or they can play, using the network, from different places, in which case VR can make them feel as if they were in one room. The main disadvantage of using VR is the high price. A casual user would more likely visit a dedicated place where they can use the headsets once, instead of having to buy one.

This can still be too pricey or unavailable in one's location. To combat

this while retaining some of the advantages of VR once may use AR, which is short for augmented reality. It is a technology that seemingly connects the real world with a virtual one. An image from a virtual scene is layered on top of an image from a device's camera. By tracking the position of a device in the real world, the virtual scene can be synchronised, creating the illusion as if the virtual objects were placed in the real world. Since this technology is supported by most modern smartphones, it is accessible to the general public. There are disadvantages to using AR instead of VR. The main one is that using your phone with a tiny screen is much less immersive than using a VR headset. AR technology is usually less stable than VR since it is not used on a dedicated game device, but on a phone, whose main purpose is not AR. Furthermore, it has to process each image the camera captures which can be computationally demanding. On the other hand, by using AR, one can utilize the surrounding area instead of having to create a whole game scene virtually.

I like the idea of escape rooms being more accessible and creating new kinds of interactions that aren't possible in the real world. I also like the idea of real-world escape rooms using AR to enhance the player experience. Since one of the most entertaining aspects of an escape room is cooperation, I will focus on multiplayer AR escape games in this thesis. Firstly, I will analyse existing AR games that are relevant to this topic. Based on the results, I will propose an AR multiplayer escape room mobile game which I will implement for this thesis. Then, I will analyse existing tools for creating AR, and network multiplayer applications and choose which ones to use for my implementation. A detailed description of the development process and a description of the final game will follow. Lastly, I will analyse the results of the testing I have performed.

Chapter 2

Analysis of existing applications

In this part of my bachelor's thesis, I compare different augmented reality applications that have similar elements to my assignment. These are escape room games or multiplayer augmented reality games. I am not comparing any games that do not use augmented reality, as it is the main and most important element of my application. The table 2.1 shows a simple comparison of my chosen applications based on the most important attributes. First, I compare which platform the games are designed for, then which development tool they are made with, if they are made for multiplayer, and if there is any player interaction via augmented reality.

Escape AR Room: Pandemic[21], Scriptum[22], ARia's legacy[27] and Prison Break AR[29] are functionally identical mobile games. All of them work on the principle of scanning the floor and creating a virtual room in AR around the player. The only interaction with the real world is the same orientation, i.e. the movement of the mobile is reflected by the movement of the game camera. Games therefore behave like virtual reality, only instead of glasses use mobile phones, and do not use the physical surroundings. In these games, one can click the screen on various in-game items, store them in an inventory, and then use them in combination with other items to solve puzzles. It is thus conceptually a single-player computer game escape room, augmented by the fact that it is not controlled with a mouse, but by moving the phone and clicking on the screen. I have personally tried playing these games on my OnePlus Nord 2T CPH2399 5G 128GB 8GB RAM mobile phone and an older Samsung Galaxy A7 (2018). There wasn't much difference in how the app ran between the mobiles although one is four years newer. Unfortunately, with these games, it often happened that the app would lose track of the mobile's location and the entire room would desynchronise, which resulted in the virtual camera appearing far away from the room and the game being unplayable and usually having to be restarted. These games are created using the AR Foundation and use built-in AR Plane Manager, which recognises flat surfaces in the real world and places them in the virtual world, as the main part of the game synchronisation. Because of these issues, I decided not to rely on this technology in my implementation and rather focus on image recognition and tracking. Prison Break AR also states that it includes networked multiplayer gameplay, which would be important for the further

progression of my thesis. Unfortunately, this mode was inaccessible, and I did not receive a response after contacting the game's authors for further details. Escape The Room: AR is, according to my available sources[28], a similar game, unfortunately, it is only available for iOS, so I was not able to try it out. I also don't think it would be beneficial for my further progress. According to the description, it doesn't contain any new or desired features.

I found Escape Team[26] while searching for AR escape games, unfortunately, after testing it I concluded that the game is not using AR. The game consists of printed puzzles that several people can work on simultaneously and a mobile phone that narrates the story and counts down the time to find out the numerical codes and enter them into the app. I decided that this app would not be beneficial for my thesis.

The Walking Dead: Our World[30] is an AR game that looked the most promising for my thesis because it features multiplayer, which isn't very common in AR games. The game stopped being supported at the beginning of 2023. You could fight zombies with other players. Like previous AR games, it scanned the floor to coordinate the physical and virtual worlds.

AR-Escape[31] is an escape AR game developed at the Technical University of Munich. It recognizes printed images or 3D objects and creates puzzles for the player in a virtual world. Players must solve all puzzles within a time limit to win the game. The game features a cooperative mode, in which a group of people solve puzzles together and their progress is synchronised across all devices, and a competitive mode, in which two groups race against each other to see who can complete the puzzles first; teams can sabotage each other. I couldn't find this app on the internet, so I contacted the authors of the study[31]. Unfortunately, no one replied. So I can't describe in detail how the game works, but I think this game comes closest to the game I'm developing for this thesis. It also lacks direct player interaction in a virtual AR world. The game does synchronise, but as I understand, each player solves the tasks alone. In my application, I want to implement direct interactions in the virtual AR world for which multiple players are necessary.

Finally, I will evaluate Cluetivity. Cluetivity is a company that creates AR-enriched escape games tailored for customers. These customers are escape room companies. As of 2023, it offers 3 games: Operation Mindfall[23], Blackout[24], and Magic Portal[25]. Operation Mindfall and Magic Portal use only one AR device. Operation Mindfall is implemented in London and is open to the public by prior reservation[32]. They are similar to the previous games, but more incorporated into the physical world in specific locations. Blackout is an escape game with hacker themes. It is played in two teams that interact by trying to hack each other and slow down the other team. Unfortunately, there are not many publicly available details about this game, so I tried to contact Cluetivity but to no avail. All of these games have a large portion of gameplay outside of the virtual world, i.e. with detailed physical gadgets. I haven't found any other company that links physical escape games to the AR world.

Title of the game	Targeted platform	Development tools	Multiplayer	Interactions of multiple players in AR
Escape AR Room: Pandemic [21]	Android 12+/iOS 12+	AR Foundation	No	No
Scriptum [22]	Android 7+/iOS 11+	AR Foundation	No	No
Cluetivity: Operation Mindfall [23]	iOS	Unknown	Outside of AR	No
Cluetivity: Blackout [24]	iOS	Unknown	Yes	Unknown*
Cluetivity: Magic Portal [25]	iOS	Unknown	Outside of AR	No
Escape Team [26]	Android 4.4+/iOS 11+	Unknown	Outside of AR	No
ARia's Legacy [27]	Android 7+/iOS 11.3+	AR Foundation	No	No
Escape the Room: AR [28]	iOS 11.3+	Unknown	No	No
Prison Break AR [29]	Android 7+	Unity (specific tool unknown)	Yes - unavailable	Unknown*
The Walking Dead: Our World [30]	Android 7+	Unknown	Yes	Killing zombies
AR-Escape [31]	Android	Vuforia	Yes	Synchronisation of puzzles, sabotaging the enemy team*

Table 2.1: Comparison of the most relevant applications using AR. * the developers have been contacted by e-mail for further details, but I got no response.

Chapter 3

Specifications of my game

A part of my bachelor's thesis was to create an AR multiplayer escape room game with at least three scenarios where the cooperation of players is required. This game would be created for mobile phones using the Android operating system. I decided to call my game "Escape Pierito", because it is an escape room game and because of my online nickname. In this chapter, I will focus on how I designed the game and what technologies I wanted to use. In chapter 4, I will talk about development tools that could be used to create such a game and which tools I chose and why. In chapter 5, I will describe the process of implementing the game and conclude if I chose the right tools for the job.

3.1 Gameplay

When I was designing the three scenarios with player interactions I was thinking of different real-life escape room puzzles that could be enhanced by augmented reality. A common type of puzzle is figuring out a number or letter code and then inputting it into a device to progress. So the first scenario would be a process of figuring out the secret code. The second scenario would be inputting the code into a machine. The last scenario would be a stand-alone puzzle that would unlock after the correct code had been submitted.

My supervisor, prof. Ing. Jiří Žára, CSc, suggested a scenario of one player using their phone as a virtual light source and the second one only seeing the "illuminated" game objects. I built on this idea and came up with a scenario where an object shows a scrambled code for the first player if the second player is looking at a linked game object. When the second player looks away, the first player's object will change and show the order of the letters in the code. This is the first scenario.

The second scenario is a way of inputting the code from the first scenario into a machine. I thought it would be interesting to engage both players. One of the players would be able to click buttons that would circle letters in the other player's code machine but wouldn't know what letters the other player sees. This would force them to cooperate to input the correct code.

For the third scenario, I thought a classic puzzle that could be used was

the sliding puzzle with eight scrambled pieces of an image that need to be placed in their correct spots by sliding them into the one empty space. I thought it would be interesting if one of the players could see the scrambled pieces of the image on top of the puzzle pieces and the other player could move the pieces they see as blank. When creating this puzzle, I didn't realise it would be lengthy and hard for some people to solve, making it even harder when two people need to cooperate.

3.2 Technical side of the game

When trying out the different applications in chapter 2, I noticed the plane recognition technology was not very reliable and had no way of recovery once it got desynchronised. For this reason, I decided to use image tracking. This way there would be different images scattered around the room that could be scanned by the game and the correct game object would be created on it. If the objects got desynchronised, the images could be scanned again to position them back in their intended place.

Since it would be unnecessarily complicated to synchronise virtual AR scenes of multiple devices, I decided to implement multiplayer synchronisation by a central network object that could communicate with the instance of the object of the other player. This object would keep track of the game progress and actions of each player, updating the scene accordingly. For instance, if one player clicks a button, the network object would recognise it and update itself on both players' sides and update the code machine for the other player.

Chapter 4

Development tools

In this chapter, I will talk about different tools for creating augmented reality applications, games supporting multiplayer, and 3D modelling. I will also mention which tools I chose for the implementation of my game and justify why I chose them.

4.1 Tools for augmented reality

First, I'll look at tools for creating AR apps for iOS. These are developed by Apple[1]. The most widely used framework is ARKit[2], which enables the most important aspects of AR apps, such as object recognition, depth perception, motion capture, background modification, face detection, and more. RealityKit[3] is used for creating more detailed scenes, more realistic physics simulations, and interaction with dynamic environments. Reality Composer[4] is a tool for easier implementation of AR applications on ARKit or Reality AR frameworks. Since I don't own any device using iOS, I won't be using these tools in my implementation.

ARCore[5][6] is Google's cross-platform AR application development kit most commonly used on Android devices. ARCore offers more limited features compared to ARKit, but the difference is in specific features that are not very important for my game, such as face recognition and body positioning. Android apps can be created in Android Studio, which is an Android development environment, in Kotlin, Java or C. For an iOS app, the XCode environment and Objective-C and Swift languages can be used. Browser apps can be created using the WebXR interface. A great advantage of ARCore is access to Google Maps, which can be used to create location-dependent applications, such as applications with historical sights of a city, where the current location and interaction with maps are necessary.

OpenXR[7] is Khronos' open standard for building extended reality, a term used for augmented and virtual reality, XR for short, applications. It is currently used by all major companies in the development and creation of tools for XR, such as HTC, Meta, MagicLeap, Steam, Microsoft, and more.

Unity[8] is the most widely used game engine and can be used to create almost any type of game. There are several ways to build AR apps in Unity, from built-in libraries to the unified framework to community plugins. I have

experience with Unity and find it intuitive to develop games in, plus it has a large user base, so it's easier to find solutions to problems on the internet.

AR Foundation[9] is a framework that connects multiple platforms and simplifies development by having unified functionalities and converting all calls to platform specific calls in the background. If any new functionality is added to a platform, it will be added to the AR Foundation. AR Foundation uses ARKit for iOS, and ARCore for Android, and can be used to create apps for Magic Leap[11] and HoloLens[10]. These two products are AR glasses, that project AR elements directly onto the glass lenses to convey a more immersive experience. These glasses can be used in medicine to help doctors during surgeries. I mention them here because applications for them are programmed in Unity. Currently supported features (as of January 2024) of each platform are visible in figure 4.1. It also supports all the features of Unity, so it's easy to combine multiple different concepts into one, such as AR and networked multiplayer in my case. The simplicity of and familiarity with Unity and the C# programming language, and the speed of AR Foundation are the reasons why I chose it to develop my application. Because AR Foundation is developed by Unity itself, it is going to be supported in the future and I expect better compatibility.

Unity's AR Foundation
Supported Features

Functionality	ARCore	ARKit	Magic Leap	HoloLens
Device tracking	✓	✓	✓	✓
Plane tracking	✓	✓	✓	
Point clouds	✓	✓		
Anchors	✓	✓	✓	✓
Light estimation	✓	✓		
Environment probes	✓	✓		
Face tracking	✓	✓		
Meshing			✓	✓
2D Image tracking	✓	✓		
Raycast	✓	✓	✓	
Pass-through video	✓	✓		
Session management	✓	✓	✓	✓

Figure 4.1: Table of supported features from the official AR Foundation website[9] (January 2024).

Like AR Foundation, Unity MARS[12] is a built-in framework from Unity. MARS is more focused on complex dynamic scenes, detailed object recogni-

tion, and physically realistic, and photo-realistic scenes. Compared to AR Foundation, it is not free to use, is more complex in approach, and needs more detailed configurations and settings for specific applications. The advantage is the ability to use it alongside AR Foundation to create games that require the lightness of AR Foundation and Unity MARS advanced technology.

Unity can also be used just as an application development library[13]. This means that the application does not need to be rebuilt in AR Foundation in Unity, but only specific AR Foundation features can be used while the application is developed in another environment.

The XR Interaction Tool[14] is a tool for creating applications in Unity without code, just by using the visual environment. It offers limited interaction options such as click events on virtual objects, dragging, adding, and removing objects from the scene.

Vuforia[15] is a plugin suite for developing AR applications in Unity. This means that it is not a direct part of Unity and is maintained by a third party, so there is no guarantee of compatibility and up-to-dateness. Before the AR Foundation, Vuforia was the most widely used way to create AR apps for Unity and is still popular today. It adds enhanced computer vision, image and object recognition, and more. Vuforia supports apps on Android, iOS, Lumin, and UWP.

Unreal Engine[16][17][18] is the second most widely used game engine after Unity. Like Unity, it includes a built-in framework for building apps that is limited to iOS ARKit and Android ARCore. It is slower to update and does not include as many features as Unity's AR Foundation. This difference is negligible in my project. Unreal Engine uses C++ and is more focused on graphical programming using so-called blueprints. Compared to Unity, it is significantly harder to create XR multiplayer games in Unreal Engine, there are no built-in libraries to convey this functionality, and plugins tend to be harder to use due to poorer documentation and less clarity. Additionally, the Unreal Engine contains fewer plugins due to its smaller user base.

4.2 Tools for network multiplayer

Since I decided to build my application in Unity, I only considered Unity's networking tools. Unity used to have its framework, UNet, which was discarded, and until recently there was nothing directly from Unity to support network play. Unity has taken the MLAPI framework and made it into its own framework, Netcode for GameObjects[19][20]. Today there is also Netcode for Entities, which is a similar framework to Netcode for GameObjects but is more oriented towards games with high quantities of players over the internet at once, while Netcode for GameObjects is more intended for local networked multiplayer. The table in figure 4.2 from 2020 summarizes the capabilities of the frameworks for networked play well. Here is still MLAPI, but the description still fits Netcode for GameObjects. I chose to develop my application using Netcode for GameObjects because it is free, it is supported by Unity, Unity represents it as a future standard, it will be maintained and

improved in the future, and my game is intended to be played by two players using a local network.

	Stability/ support	Ease-of-use	Performance	Scalability	Feature breadth	Cost*	Customers recommend for
MLAPI	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★★	Free	Most client-server games for up to ~64 players that want a stable breadth of mid-level features
DarkRift 2	★★★★★	★★★☆☆	★★★★★	★★★★★	★★★☆☆	$100 for source	Games with high perf/ scale requirements that want to build on a stable LL layer
Photon PUN	★★★★☆	★★★★★	★★★☆☆	★☆☆☆☆	★★★★☆	$0.30/PCU	Simple and small (2–8 players) mesh-topology games
Photon Quantum 2.0	★★★★☆	★★★★☆	★★★★★	★★☆☆☆	★★★★☆	$1000/mo + $0.50/PCU	Games desiring deterministic roll-back, like MOBA games, for up to 32 players
Mirror	★★★★★	★★★★★	★★★☆☆	★★★★☆	★★★★☆	Free	Stable and proven client-server solution, loved best for its community and ease-of-use

* Note that Photon pricing provides access to the networking libraries and services, whereas other solutions are standalone networking libraries, and the cost of services is separate.

Figure 4.2: A table of tools enabling network multiplayer for Unity (2020)[19].

4.3 3D modelling tools

Since 3D modelling isn't the main focus of my bachelor's thesis I was deciding whether I should create my own 3D models or use some from the internet. I decided to create my own so I could create exactly what I needed for my game as opposed to having pretty-looking models but having to compromise on my game design, so the models would fit.

Because I wanted to create simple 3D models to be used in Unity, I used the tool I'm most familiar with, Blender 3.6. I knew beforehand how to create models in Blender and how to use them in Unity thanks to the course Creating Graphic Content (VGO). I didn't consider any other options, because it would take too much time to learn how to use, and with Blender, I could focus on augmented reality and network synchronisation instead. If I didn't have experience with Blender beforehand I would consider using Maya or Cinema4D as well.

Chapter 5

Game implementation

In this chapter, I will focus on the implementation of the game described in chapter 3. First, I will talk about my first simple game scenes in Unity that showcase and test the functionality of the final game without specific puzzles and visual design. Then, I will describe the process of creating the 3D models, implementing the puzzles, and finishing the whole game. All code and models developed for this game are available in the digital appendix. The digital appendix is described in the appendix C.

5.1 Basic AR scene

To start working on the app, I first had to create a new project in Unity and prepare it for AR support. I worked in Unity 2022.3,13f1 with support for Android and iOS, which wasn't necessary since my app is Android-only. I built my project on Unity's empty template AR scene. This makes it easier to work with, as almost everything in the template scene is pre-made for running an AR app. The only thing that needs to be added is a specification of what device and what work-frame the app is for. In my case, that's ARCore for Android. The correct packages are selected for the AR app in the Unity template scene, namely the AR Foundation, Apple ARKit XR plugin, Google ARCore XR plugin, Magic Leap XR plugin, Apple ARKit FaceTracking XR plugin, and OpenXR Plugin. Since my app is Android-only, it could do without the ARKit packages. Furthermore, the AR Session, AR Session Origin, and AR Camera objects are already created in the scene. AR Session handles the communication with the player, handles the input and sets what to track, the rotation, position, both or nothing. AR Session Origin takes care of all the necessities for the AR app to function, including specialised technologies like finding and tracking images in the real world, finding flat surfaces in the real world, and casting rays from the AR Camera. The AR Camera works similarly to a normal camera in the Unity scene, only it reacts to external inputs such as the position and rotation of the device. The game scene and some packages are described in more detail in appendix A.

After building and running this app on a mobile device, nothing significant happens because the scene is empty. The user will only see the image from the camera, and gradually, places where the app thinks are flat surfaces in the

physical world will be highlighted. This is because of an AR Session Origin component called AR Plane Manager, which, when a flat surface is found, creates a virtual representation of it in its place.

I then placed a cube in the empty scene and observed how the application would behave. Once launched, the cube was robustly positioned in the air and responded correctly to any movements I made. In the image 5.1 you can see the position of the cube in the air and the dotted surfaces show the recognized flat surfaces. You can also see that the camera is angled and the cube in the scene is still oriented correctly according to the surrounding space.

Figure 5.1: A simple AR scene with a floating cube and dots representing found flat surfaces.

5.2 AR scene with ray casting

AR sync now works in such an application, but the user can't interact with anything. The main method of interaction in my app between the player and the virtual world is ray casting. This works on the principle of sending a ray from a specific point on the screen through the scene, which acts when it collides with the first object. The ray would be cast by the player tapping the screen. To test this out, I implemented it in two ways. The first one creates an instance of an object consisting of a cube and a sphere on a recognized surface, the second one changes the colour of the cube after touching it on the screen.

The first interaction is inspired by the internet[41] and is implemented using the AR Raycast Manager, which is an AR Session Origin component. This component handles the casting of AR Rays, which can collide with AR-specific objects such as detected surfaces. When the first collision with a surface is found, it creates an instance of an object at the collision coordinates. AR Rays can't detect collisions with non-AR objects in the scene so I decided not to use AR Plane Manager. I wanted to use non-AR Rays that are a part of base Unity. It is also important to know that rays can only collide with objects that have a collider component in Unity. A collider is a component that approximates the shape of an object. All collisions are calculated on the shape of a collider and not the 3D model itself.

All code for input handling from this point on is made only by me referencing official Unity manuals. AR Camera has a "public Ray ScreenPointToRay(Vector3 pos)" method that takes a position in screen coordinates and emits a ray from that position into the virtual scene[42]. This is how I implemented the second interaction. The method that triggers after the player touches the screen is visible in figure 5.2. When the ray collides with an object with the tag "TestCube" it will trigger a method of that object called "Change" with no parameters. This method changes the material of the object.

So my script would cast two beams of different types when touching the screen, both interactions can always happen at the same time. This can be seen in the image 5.3, where one cube changed colour from red to green, but at the same time a new cube was created near it. The recognized flat planes are shown in yellow and bordered in black. Also, notice the cubes are halfway inside the plane. This is because the prefab for the cubes has their origin in the centre. To make them appear on the surface, rather than inside it, the prefab's local position needs to be changed.

During the testing I encountered numerous issues with AR Plane Manager, confirming I had chosen well to not use it for the sake of stability. In the figure 5.4, you can see that during testing of the input system, the scene got so desynchronised, that flat surfaces that used to be around me were barely visible in the distance (the image is zoomed in).

```
2 references
private void FingerDown(EnhancedTouch.Finger finger)
{
    // disable multiple touch
    if (finger.index != 0) return;
    Ray ray = mainCamera.ScreenPointToRay(finger.screenPosition);
    RaycastHit hit;

    if (Physics.Raycast(ray, out hit))
    {
        if (hit.transform.CompareTag("TestCube"))
        {
            ChangeMaterial tcs = hit.transform.GetComponent<ChangeMaterial>();
            tcs.Change();
        }
```

Figure 5.2: A part of the method that is called when the player touches the screen which casts a non-AR ray which triggers the method Change if it collides with an object with the tag "TestCube". This is a part of the "InputManager.cs" file.

5.3 AR scene with multiplayer

I focused on synchronizing the multiplayer scene using Netcode for GameObjects next. I added the necessary package to my project and created a NetworkManager object with Network Manager and Unity Transport components which take care of the communication over the network. Netcode for

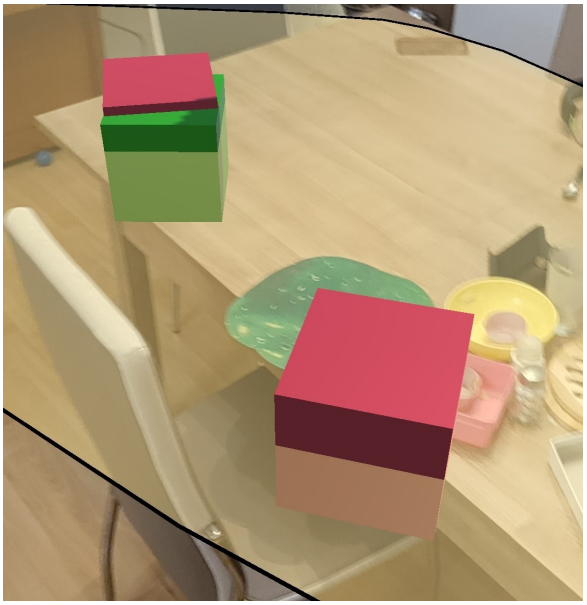


Figure 5.3: Cubes being created on the detected plane by touching the screen. Their material changes when touching the screen as well.

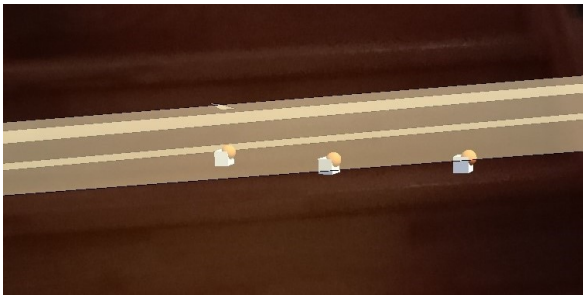


Figure 5.4: Flat planes detected on the ground around me which are now barely visible in the distance due to desynchronisation. The scene was originally the same as the one in figure 5.3 except for different models for the cubes.

GameObjects recognizes three kinds of users: server, host and client. The server has the most privileges and takes care of running the network elements but does not participate in the game. The client is always connected to the server and can't directly change network variables. The host is both the client and the server. I created a simple GUI that allowed the user to connect as a host or a client and specify what local IP address to try to connect to. Every Script in Unity inherits from the MonoBehaviour class. For an object to be shared or communicate over the internet, it must implement the NetworkBehaviour specified in Unity.Netcode. This abstract class makes it easy to manipulate network data, for example, it is easy to see if the owner of a script is a host or a client and what its user ID is. To test network communication without building an Android app, I used the ParrelSync package[33]. This creates an exact copy of the current project and opens it in a new editor. It is then possible to test the communication on one machine

and only in the editor. This way I could first test if the communication works on the computer and only then try it out on phones.

For the first test, I used one cube in the scene that changed colour when you touched it on the screen. The screen touch is not sensed on my computer, so I had to create a simple keyboard control. If there is no support for AR, the camera is static in the position it is initially placed in the scene, which solves the problems of long testing on mobile devices. If the colour of the cube is changed by the host, the change is sent to the client as well. For the client to do the same, a call to ServerRPC[44], which is a request for a method to be called on the server side, must be used. Next, I needed to think about who owns the object. For example, I didn't want one player to control other players' characters when trying to move their own. I solved this with a simple condition on the owner of the object instance. At this stage, all changes were reflected on both devices. I created a simple script that includes a network variable which is a construct that lets one variable be shared and changed by instances of the same object on different devices. This script asks in the "FixedUpdate" function what the value is and changes the material of the cube based on that. This is rather inefficient because it keeps asking for a value of the network variable. It would be more efficient if each instance notified all others if the value changed in its instance, but the main purpose of this was to try out how network variables, and other kinds of network synchronisation work. In figure 5.5, you can see the code for declaring a network variable and the content of the FixedUpdate method. The network variable is just a container and to access its stored value one needs to access the "value" attribute. As visible in the code snippet, only the owner can change the network variable value.

```
private NetworkVariable<bool> first = new NetworkVariable<bool>(true, NetworkVariableReadPermission.Everyone, NetworkVariableWritePermission.Owner);

Unity Message | 0 references
private void FixedUpdate()
{
    if (_first != first.Value)
    {
        if (!first.Value) GetComponent<Renderer>().material = mat2;
        else GetComponent<Renderer>().material = mat1;
        _first = !_first;
    }
}
```

Figure 5.5: A snippet of a code that synchronises the material colour of a cube that can be changed from both the host and client side.

The next step was a more complex synchronization. I created a simple capsule-shaped object representing the player character and created a simple control script[45] using the Joystick from the Joystick Pack from Fenerax Studios on the Unity Asset Store[34]. The NetworkManager in Netcode for GameObjects is made for games, so it can be used to set up a prefab for the player character. It also takes care of making sure each player has their own and synchronizes their movement. Using this, you can move the character in 3D space in an AR application. Each player could now control their own character with the joystick and for the other player, the position of both characters synced correctly. This can be seen in figure 5.6. A script taking care of the player movement and cube material synchronisation is called

"PlayerScript.cs" and is available in the digital appendix. I used this part of the game throughout all of the development to be able to quickly tell whether the network synchronisation works as intended or if there is an issue.

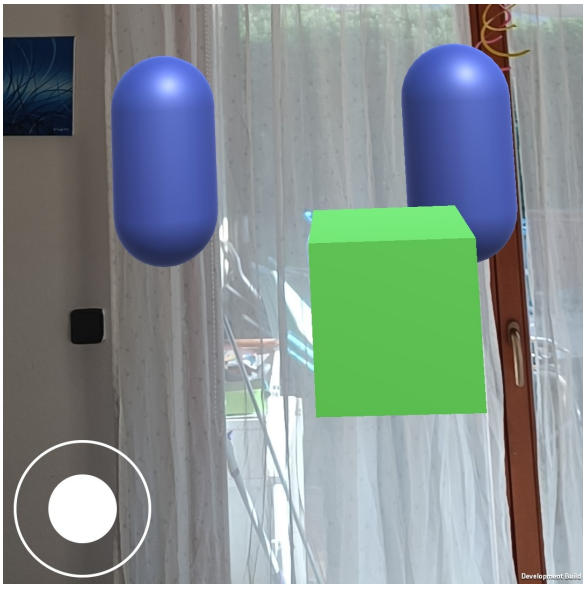


Figure 5.6: An image of a game synchronising the colour of the cube and the position of each capsule, which the players can move using the joystick on the bottom left.

I also created a network script "NetworkUIManager.cs" that synchronises the progress of the game and takes care of UI elements. Each player owns an instance of an object containing this script so these instances can communicate and synchronise game progress. It takes care of creating a host and connecting the client to the given IP address.

In the implementation of my game, one player is always going to be a host and the other is always going to be a client. The game is intended for two players, so instead of calling them, for example, player one and player two, one is always going to be referenced as the host and the other as the client. It only reflects the network state of the game of each player, but I believe it is more convenient and clear to call the players this way.

Working with Netocode for GameObjects seemed intuitive and smooth for local multiplayer. There were no functions I missed and I believe it has been the right choice. I wouldn't have achieved better results if I had chosen a different tool. It also made network communication easier, taking care of all low-level communication. This saved me a lot of time. The disadvantage of this is that it takes away the freedom in taking care of the communication in the exact way one wants. This was not an issue for me since I could focus more on implementing the rest of my game. It is also good for debugging. When I built the game and played it on my phones, one of them would always connect to the Unity editor and print out all debugging logs.

The most significant issue I had with Netcode for GameObjects, which is still a mystery to me, is that sometimes the network synchronisation stopped

working on certain networks (three different ones) for no apparent reason. It stopped working on all already built versions, that used to work before. This could've changed from day to day. One day it was working and the other it would not. There were no error messages. The client was not able to connect to the host as if they were connected to different networks. I tried debugging and changing related settings in Unity but I did not figure out the reason for this issue. I spent a few weeks trying to solve it, but I have not encountered this issue again since the beginning of April 2024.

5.4 Image recognition and tracking

In Unity, I needed to add an AR Session Origin component called AR Tracked Image Manager, which accepts a "Reference Image Library" containing images that it looks for in the real world through the device's camera. In Unity, one can subscribe to the "trackedImageChanged" event of the component, which means that every time the tracked images change, a function will be called. An object of type "ARTrackedImageChangedEventArgs" is sent to the function, which contains the new, changed, and lost images. My implementation of a function called when this event is triggered builds on the implementation from the internet[53], which loops through all recognised, tracked, and lost images and creates or destroys a game object with the same name as the image. In my script (the part that is fully created by me), this function creates an object on the image depending on whether the user is a host or client. Also, for the functionality of the first scenario, it notifies the local "NetworkUIManager.cs", which notifies over the network when an image named "WatchBox" is being watched (figure 5.7) and when it is lost. The name of an image is set inside the "Reference Image Library" so it's easy to change tracked images when needed, but there should always be one called "WatchBox" for the first scenario to work correctly. When the figure is lost, the object is not destroyed, which adds to robustness because the object remains in place in the scene at all times and only updates its location when the figure is found again.

```
foreach (GameObject prefab in clientPrefabs)
{
    if (string.Compare(prefab.name, name, StringComparison.OrdinalIgnoreCase) == 0)
    {
        if (!watchBoxSeen && string.Compare(name, "WatchBox", StringComparison.OrdinalIgnoreCase) == 0) {
            nuim.SeenChanger(true);
            watchBoxSeen = true;
        }
        GameObject newObject = Instantiate(prefab, image.transform);
        placed[name] = newObject;
    }
}
```

Figure 5.7: A part of my code that loops through game object prefabs for the client and finds one with the same name as the image recognised in the scene, if the recognised image is called "WatchBox" it also calls a method of the "NetworkUIManager.cs".

In the beginning, I had problems with choosing an image that would be

recognised and tracked well. I first tried the images in figure 5.8, thinking it would work well, not knowing how the image recognition technology works. After reading several articles[35][36][50] on the topic of choosing the right image, I found that the quality of the image mainly depends on the image having a wide spectrum of brightness, that it is not repetitive, that it is complex, and that it is well spread out. For example, image 5.8a is too repetitive, and image 5.8b is simple with just a few narrow lines. It also depends on the shape. Square images are the easiest to recognize, and an odd number of pixels in one of the dimensions can detract from the quality of recognition. All images are converted to black and white, so it's the brightness that matters, not the colours. Later on, I tried printing the final chosen images for my game in black and white and there was no difference in recognition quality compared to originals. Furthermore, the app finds the most notable features and remembers their position in the image. Then, it searches for these features in the pictures from the device's camera[49].

(a) : A colour pattern I initially thought would be good for image recognition[46].

(b) : A drawing of a glass made by my mum.

Figure 5.8: Images I first used when trying out the image recognition technology.

Google also has an application called The arcoreimg tool[38] to rate the quality of an image for recognition, which gives images a score of 0 to 100, where 100 is the maximum. On the official site, Google recommends using images with a score of at least 75. I used this tool to grade images representing the good and bad qualities of an image for tracking from one of the articles[50]. These images are seen in figure 5.9 and the results given to them by The arcoreimg tool are seen in table 5.1. The good qualities always got a better grade than the bad ones, one of them even got a perfect score. The grade also depends on how the image is cropped. So I tried running The arcoreimg tool on the same image with a different resolution. The values in the table 5.1 are a mean of those results. I then chose three images seen in figure 5.10. I expected the images 5.10a and 5.10b to perform poorly and the image of a concert 5.10c, although not square-shaped, to do well. The King of Hearts got 60 points, which surprised me because it's repetitive and has few details. The image from a concert got 0 points, which also surprised me because it's

detailed. Perhaps it got no points because of the panoramic resolution and small changes in brightness. The album cover art image in the top right got 90 points. It has notable changes in brightness, some details and a square shape. Because of this, I chose to use this image in my final game.

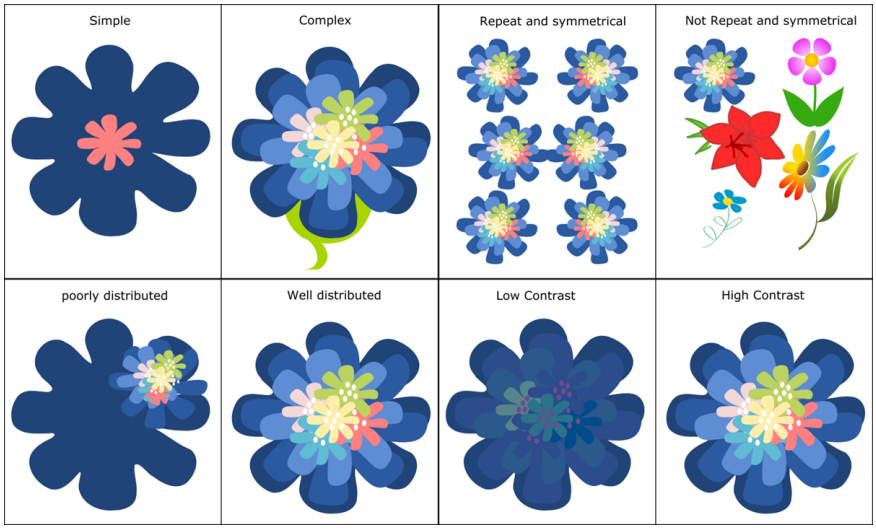


Figure 5.9: Images from an article about how to choose a good image for AR tracking displaying good and bad qualities of an image[50].

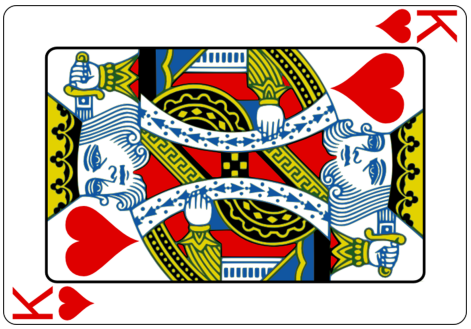
Bad quality	Score	Good quality	Score
Simple	10	Complex	85
Repeat and symmetrical	80	Not repeat and symmetrical	100
Poorly distributed	65	Well distributed	95
Low contrast	15	High contrast	95

Table 5.1: Results of grading the images from figure 5.9 using Google's The arcoreimg tool[38].

The arcoreimg tool is a good approximate metric when choosing an image. One should always try out the images in practice to see how good they are since The arcoreimg tool only uses simplified metrics and doesn't take everything into account. For example, an image of a zebra seen in figure 5.11 got the maximum points in The arcoreimg tool, but when used in Unity, it was recognised much worse than other images, for instance, the King of Hearts image from figure 5.10, which only got 60 points. Textures that don't belong in images, such as watermarks, also add contrast and orientation to images. After testing with The arcoreimg and basic image recognition in Unity, the images in figure 5.12 performed the best. I used these images in my game.

The most unstable part of image recognition is the initial image recognition which can take up to a few seconds based on the lighting, angle between the camera and the image, and size of the image. After the image is found, it is usually steadily being tracked until the image leaves the field of view of the

(a) : A cover art for Judas Priest's album Screaming for Vengeance[51].



(b) : An image of the King of Hearts card.

(c) : An image of a concert[52].

Figure 5.10: Images I first used for image recognition after reading articles about the topic.

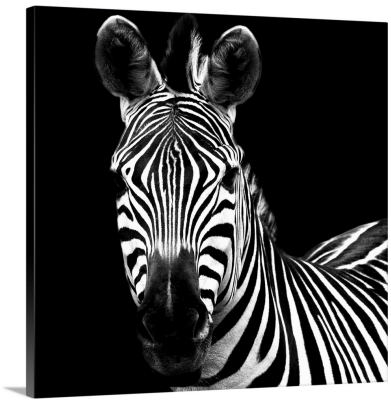
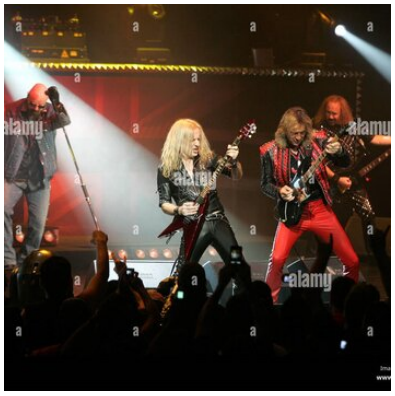


Figure 5.11: An image of a zebra, which got 100 points in The arcoreimg tool, but was not suitable for use in AR image tracking[54].

camera, or becomes too far to be tracked.

I also tested images 5.10a, 5.12a, and 5.12b used in my app and the zebra image from figure 5.11 in different light conditions, by having four simple objects placed in the image's position. This testing was done using the OnePlus Nord 2T CPH2399 5G 128GB 8GB RAM mobile phone. In daylight and in ordinary room light, the image recognition worked perfectly, the images were usually found within a second, except for the zebra which took longer.



(a) : An image of a concert[37].

(b) : An image of a coast with birds[36].

(c) : An image of a cave[47].

(d) : A poster for Metal Fest in Pilsen[48].

Figure 5.12: The images that turned out the best when testing image recognition and tracking.

The recognition time can be significantly longer if there is a shadow partially covering the image. When moving the images, the zebra kept getting lost, while the other three were being tracked all the time and updated according to the movement. When moving away from the light source, the 5.12a and 5.12b images kept tracking to almost complete darkness, with only faint light from another room, where it started having troubles. If I used the phone's torch as a light source, there was barely any difference from tracking in daylight. The three images used in the game were able to recognise an image even in the faint light of another phone's screen. I expected the game to be played in a lit room, so these tests seemed promising for the stability of my game.

All parts of the image recognition seemed to work reliably and fast. Working with image recognition was intuitive with the help of Unity manuals, which were well-documented and clear. If I encountered any issues I could search online for solutions and usually at least some came up, which I believe wouldn't be the case if I used a different tool. I am content with the choice of Unity and AR Foundation for my game.

5.5 3D Modelling

In this section, I will describe the process of creating each of the models used in my game. All 3D models were made in Blender 3.6.

For most of my 3D models, I first created a 3D mesh representing the shape of the object. Then I created materials for different parts of the objects and baked albedo, roughness and metallic textures for the 3D model. I used Gimp 2.10.36 to create details in the textures because it's free, I had experience working in it, and it's easy to perform the tasks I needed in it.

All images of models in this section are taken in Blender, except for the puzzle box in subsection 5.5.5, because it uses transparency in albedo texture to create a see-through window. This means it renders transparent in Unity, so the images are taken in Unity.

5.5.1 Button Box

The first model I created for my game was a box with three buttons labelled from one to three, as seen in figure 5.13. Buttons themselves are separate child objects of the box so they can move when clicked on.

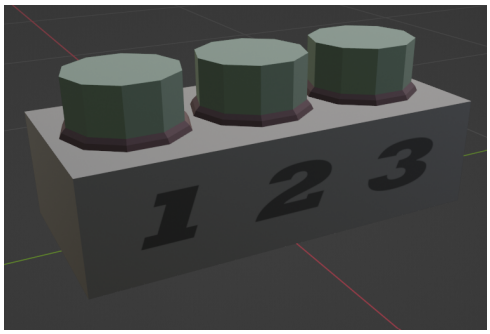


Figure 5.13: An image of the button box model created for my game.

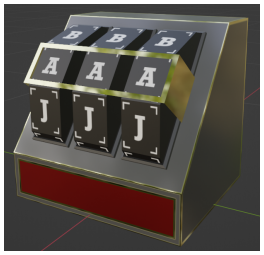
5.5.2 Wheel Box

I called the machine for inputting a code, seen in figure 5.14, the wheel box because it uses letter wheels to show the code, so as not to be confused with the code box, which displays letter and their order in the code.

In this case, I spent more time trying to make the model look appealing compared to the previous one. The details of the letters on the letter wheels and the hint from the back of the machine were made in GIMP. It also has two albedo textures, one for when the code is not correct (image 5.14a) and one for where it is (image 5.14b).

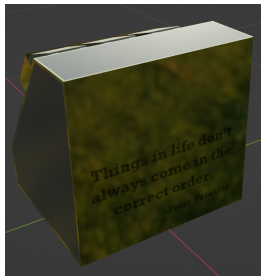
5.5.3 Code Box

The code box displays the scrambled code and the order of the given letters in the correct code. For this, I have created two textures for a box with three



(a) : Wheel box from the front with red, unsolved texture.

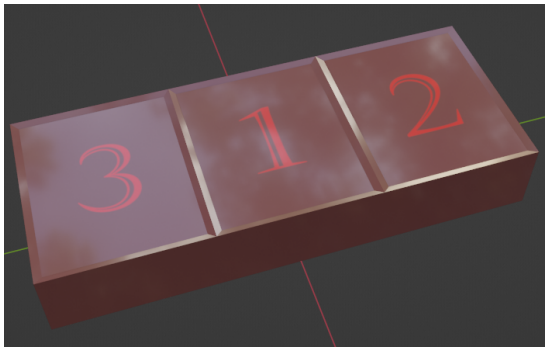
(b) : Wheel box from the front with green, solved texture.



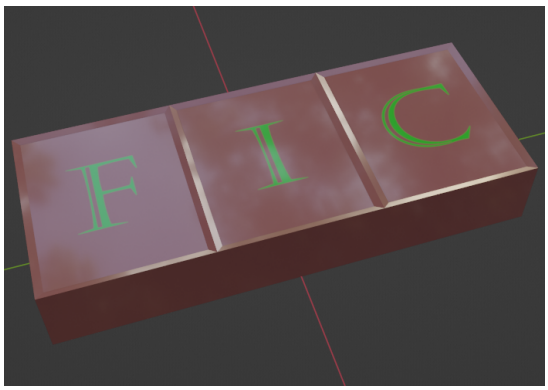
(c) : Wheel box from the back shows a hint for the players.

Figure 5.14: The images of the wheel box.

spaces to display characters. Both of these options are visible in the figure 5.15.



(a) : Code box showing numbers, displaying the correct order of the letters.



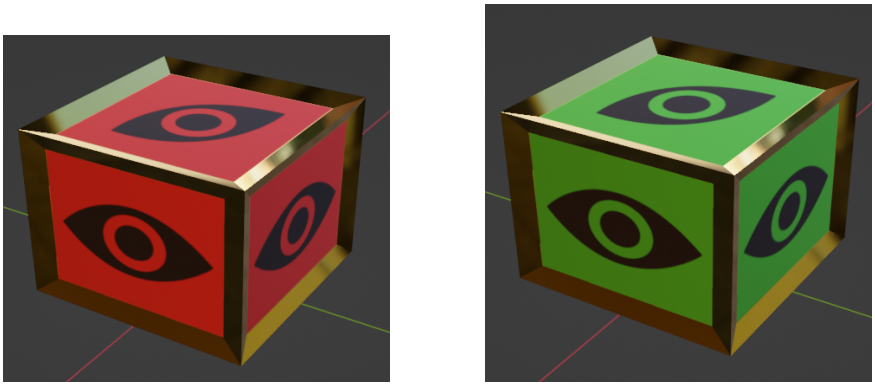
(b) : Code box showing the letters of a scrambled code.

Figure 5.15: Images of the code box.

5.5.4 Watch Box

The watch box, seen in figure 5.16, is an object which one of the players needs to observe for the other one to see the letters on the code box instead of the

numbers. To notify the first player of this change, the watch box is red when not observed and green when it is observed. To indicate that it should be observed, there are simplistic images of eyes on each side.



(a) : Watch box with the red texture. **(b) :** Watch box with the green texture.

Figure 5.16: Images of the watch box.

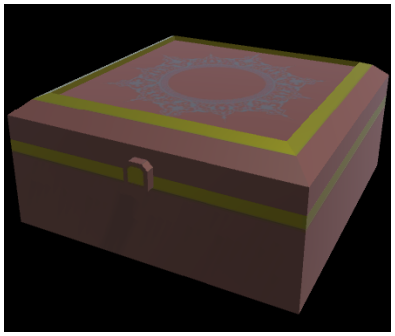
5.5.5 Puzzle Box

The puzzle box is the most complex 3D model. The cover of the box can be solid or see-through with an ornament[39] in the middle. It contains eight pieces of a sliding puzzle, each displaying a piece of an image[40] that is also shown on the bottom of the puzzle box. In the figure 5.17, you can see the closed box from the front (image 5.17a) and back (image 5.17b), and on the bottom how each of the players see the box after all previous tasks have been completed (images 5.17c and 5.17d).

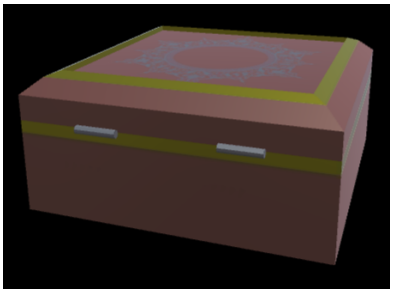
5.5.6 Plant

After creating all the models for the game, I decided to create two dummy models. These models would appear in the game but just as decoration and the players won't interact with them. The first of these models is a model of a plant in a pot. The process of creating the plant model is depicted in figure 5.18.

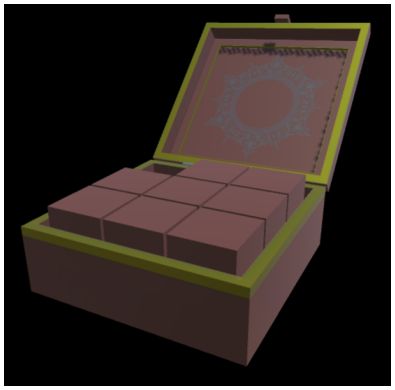
For this model, I decided to take a bit of a different approach and use a normal map. Plants are naturally detailed and wouldn't look good using simple shapes. First, I created a simple model of the plant without details as seen in image 5.18a. Then, I used the Subdivision Surface modifier in Blender to create a high-poly model. Using the sculpting tools I created the details of the crown, trunk, and dirt. The details are seen in the image 5.18b. Afterwards, I tried to bake the normal map using Blender's built-in tools. This turned out poorly without using a cage. I copied the simple model and scaled it down so the normal vectors don't map on the opposite side of the model. This solved the issue and the normal map worked as intended. The bake settings inside Blender are shown in image 5.18d. Lastly, I created materials



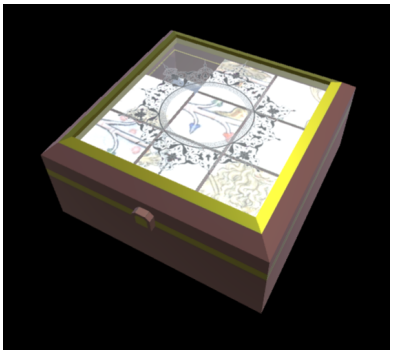
(a) : Closed puzzle box from the front.



(b) : Closed puzzle box from the back.



(c) : Open puzzle box with blank pieces.



(d) : Puzzle box with a transparent cover and puzzle pieces displaying a scrambled image[40].

Figure 5.17: Images of the puzzle box model.

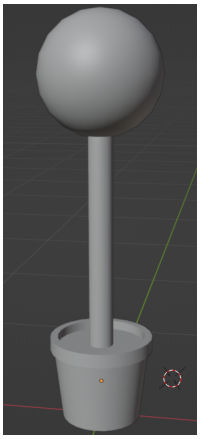
for each part of the model and created albedo and roughness textures (not metallic this time, since the value is always zero). The simple model with all textures and the normal map applied is visible in image 5.18c.

5.5.7 Cube

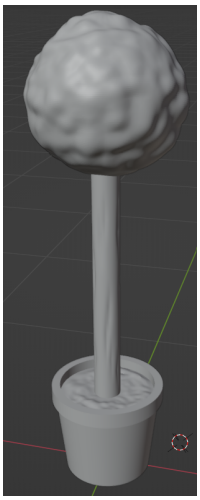
The second dummy model and the last model that I created is a model of a toy cube as seen in figure 5.19. The cube has a logo with my initials and is completely realistic, meaning a real cube could be scrambled into this position. Originally, the cube was intended to be fully functional, but since it is just a dummy object, I didn't want to confuse the players.

5.5.8 Using the models in Unity

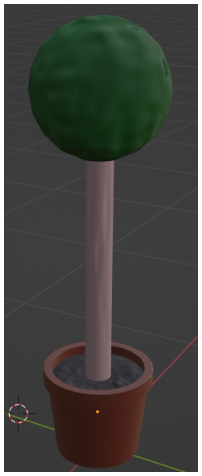
To use the Blender models in my Unity project, I first exported each model as a .fbx file and placed it in the project's folder containing all my 3D models (Assets -> BlenderMeshes). Next, I needed to create a Material for the model. For this purpose I needed to create a metallic-smoothness texture from the metallic and roughness textures, applying inverted roughness as the alpha channel of the metallic texture. I did this in GIMP. Then, I created new materials in Unity and set the correct textures for albedo, metallic-smoothness,



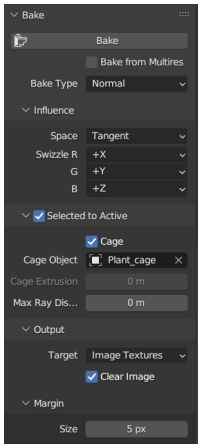
(a) : The simple low-poly mesh of the plant model.



(b) : The detailed high-poly mesh of the plant model.



(c) : The low-poly model with applied normal map and albedo and roughness textures.



(d) : The settings inside Blender I used to bake a normal map from the high-poly model to the low-poly one.

Figure 5.18: Images of the process of creating the plant model.

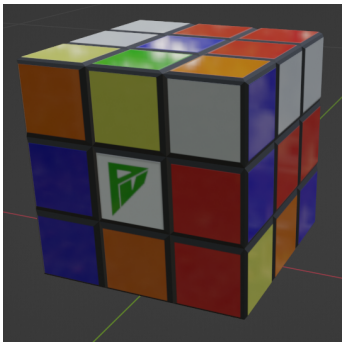


Figure 5.19: An image of the toy cube model taken in Blender.

and normal maps, if needed.

Afterwards, I created prefabs using the models imported to Unity. I added the needed components to the models, so the game works from a technical standpoint. The "ImageRecognitionScript.cs", which I talked about in section 5.4, contains two arrays of game object prefabs. One array is for the host's, and the other is for the client's prefabs. I also needed to make sure that in the Reference Image Library, there is always an image with the same name as each given prefab. A host's and client's prefabs that are supposed to be created on the same tracked image needed to be called the same as well. When this is done, the script takes care of creating the correct prefabs on the correct images in-game. This process is described in more detail in appendix A.

5.6 First scenario

The first scenario, as described in chapter 3, is supposed to be a riddle for the players to find the correct secret code for scenario two. From the gameplay perspective, it should work as follows. If one of the players looks directly at an object with eyes on it (watch box from subsection 5.5.4), it would change from red, as seen in image 5.22c, to green, image 5.22a. Simultaneously, when the first player's object (code box from subsection 5.5.3) turns from red to green, the second player's object in the same place would change from displaying numbers, visible in image 5.22d, to displaying letters, seen in image 5.22b. This might not be obvious at first to the players. The letters show the letters used in the secret code but in the wrong order. The numbers tell the players the correct order of the letters. Specifically, the scrambled code shows the letters "F", "I", and "C", and the letters show numbers 3, 1, and 2. This means that the letter "F" comes in the third position in the code, the letter "I" is the first letter of the code, and the letter "C" is in the middle. The secret code then is "ICF".

5.6.1 Hierarchy of the scripts in the Unity project

I will now mention all eleven C# scripts I developed for this game and summarise their use.

The "NetworkUIManager.cs" is a component of the AR Session Origin (mentioned in section 5.1) that takes care of all the network communication between the host and the client. It also takes care of the UI functionalities, which connect the two players.

The "ImageRecognitionScript.cs", mentioned in section 5.4, is a component of the AR Session Origin, which uses another component of the AR Session Origin called "AR Tracked Image Manager". The "ImageRecognitionScript.cs" responds to images from a Reference Image Library being found and lost in the scene. It keeps an array of game objects for both the host and the client to instantiate on a corresponding image. As I mentioned earlier, the base of the script is inspired by the internet[53], but it is heavily modified by me to fit the needs of my game.

The "InputManager", talked about in section 5.2, is also a component of the AR Session Origin. It subscribes to the "onFingerDown" event which is triggered when the player touches the screen. After that, a method of the AR camera is called which casts a ray (Unity's base ray, not AR Ray) in the game scene, and, if the ray collides with an object, calls appropriate methods. The part of the script subscribing to the event is inspired by the internet[41] but the method called and the ray casting is all done by me.

The "PlayerScript.cs", described in section 5.3, is used to control each user's player object and spawn a cube which would change colour and be synchronised between players. This was done to test the network communication during development and is not a part of the final game. The player movement part of the script is based on a video from the internet[45].

The "WatchBoxScript.cs" takes care of changing the material of an object when a method is called. It is used for the implementation of the first scenario.

The "ChangeMaterial.cs" and "seenCubeScript.cs" are both prototypes of the "WatchBoxScript.cs" and aren't used in the game. They both used the "FixedUpdate" method to continually ask "NetworkUIManager.cs" for a value of a variable. The first script would change material based on the value, and the second one would change which child of the object is active.

The "ButtonBehavior.cs" script takes care of the button clicks and button animations in the second scenario. It is a component of all three buttons of the button box from subsection 5.5.1.

The "WheelBox.cs" is a component of the wheel box from subsection 5.5.2. It takes care of turning the letter wheels one step each time a button is clicked, and of checking, whether a code entered is the correct one.

The "PuzzleBoxScript.cs" is a component of the puzzle box from subsection 5.5.5. It takes care of coordinating the movement of the puzzle pieces and checking whether the puzzle is solved.

The "PuzzlePieceScript.cs" is a component of each of the puzzle box's pieces. It moves the piece and notifies the puzzle box which piece has been clicked on.

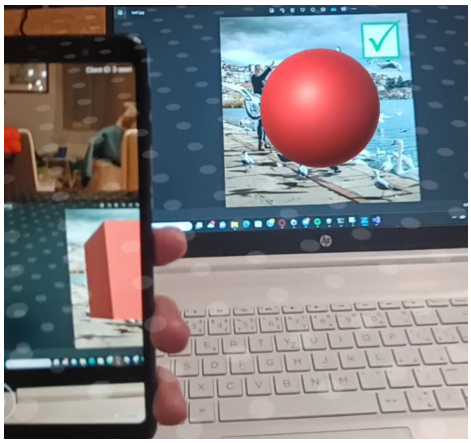
All of these scripts in the C# programming language are available in the digital appendix. The contents of the digital appendix are described in appendix C. These eleven files span around 1300 lines including numerous comments explaining what the codes do for clarity.

5.6.2 Implementation

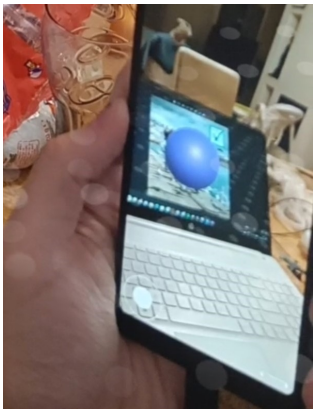
In section 5.4, I mentioned that the "ImageRecognition.cs" script calls a method of "NetworkUIManager.cs" to notify whether the specific image called "WatchBox" is being tracked by the client. This is the main part of the first scenario's technical side and the "NetworkUIManager.cs" takes care of any specific network communication for the scenario to work.

Originally, I implemented a simple representation of this scenario in which I used different objects for the host to represent whether the client's game is tracking the specific image as seen in figure 5.20. Here, the host sees a red cube if the image is being tracked (image 5.20a) and a blue ball if it isn't (image 5.20b). This was originally implemented using a boolean network variable inside the "NetworkUIManager.cs", which would be updated every time the image is recognised or lost. This is implemented in the "seenCubeChanger.cs" script. Then, the specific object on the host's side would ask the "NetworkUIManager.cs" in the "FixedUpdate" method, what the value is.

This approach is inefficient because it keeps continually asking for a value inside a network variable, which can be demanding and slow down the game. My game is quite small so it wouldn't have had a big impact but I didn't like this approach. So I reworked the script inside the "NetworkUIManager.cs" into a simpler form as seen in figure 5.21. In the code snippet, there is a public method called "SeenChanger" that accepts one boolean parameter



(a) : The client is tracking the image on the computer screen so the host (the phone held) sees a red cube on the tracked image.



(b) : The client is not tracking the image on the computer screen so the host (the phone held) sees a blue ball on the tracked image.

Figure 5.20: Screenshots from the client's version of the game showing the prototype of the first scenario. The host's version of the game is visible on the phone held in my hand.

called "isSeen". This method is always called from the client's side of the game when the image called "WatchBox" is recognised or lost in the scene. This method runs locally, always on the client's side. The "isSeen" parameter contains a value of whether the image is being tracked or not. First, it updates its local variable "seen" to the value of "isSeen". This is so that when a new object with the component "WatchBoxScript.cs" is instantiated, it can ask for the variable's value to know which material it should use to render. Next, the method finds the first instance of a "WatchBoxScript.cs" in the game scene, and if one exists, it calls the method "ChangeSeen" with the boolean parameter. The "ChangeSeen" method changes the material of the object based on the boolean variable. The "WatchBoxScript.cs" is a component of objects using the models 5.5.3 and 5.5.4. If the boolean parameter of the "ChangeSeen" method is true, it changes the material to the green version, otherwise, it changes to the red version. If the method in figure 5.21 is analysed further, it calls a method called "ChangeSeenServerRpc". It is a remote procedure call, meaning it is a request for a method to be performed on an instance of the given script owned by a (usually) different user. In this case, it is a server RPC, which means it requests that the method called "ChangeSeenServerRpc" be run on the server side. As I mentioned earlier, the host counts as being both the server and the client, so if a client calls a server RPC, it is performed on the host's instance of the script. The "ChangeSeenServerRpc" method contains an almost identical body to the "ChangeSeen" method.

In the final version of the game, I chose the image from 5.12 on the top right to be the image called "WatchBox", because it performed better than the others during testing. When running the game now, the first scenario

```
3 references
130     public void SeenChanger(bool isSeen)
131     {
132         // always called on the client side
133         seen = isSeen;
134         WatchBoxScript wbox = FindObjectOfType<WatchBoxScript>();
135         if (wbox) wbox.ChangeSeen(isSeen);
136 
137         ChangeSeenServerRpc(isSeen);
138     }
```

Figure 5.21: A code from "NetworkUIManager.cs" taking care of changing a material of models 5.5.3 and 5.5.4, based on whether the client's game is tracking the image called "WatchBox".

works exactly as it should. If the client's game is tracking the image, the objects, which are created in the place, where the image is tracked, change their material. This means that when the image is being tracked by the client, the client sees the green version of the watch box model (image 5.22a) and the host sees the scrambled code on the code box model (image 5.22b). If the image is not tracked by the client, the client sees the red version of the watch box model (image 5.22c) and the host sees the numbers on the code box model (image 5.22d). This also means that if the client is not tracking the image and sees the red model 5.5.4 and the image moves, the model will stay in place. On the other hand, if the image is tracked, the model would move with the image, as if it was physically placed on it. This is because, when a game object is created when an image is recognised, the object's transformation is parented to the image's position. So as long as the image is actively tracked, the object will stay on top of it.

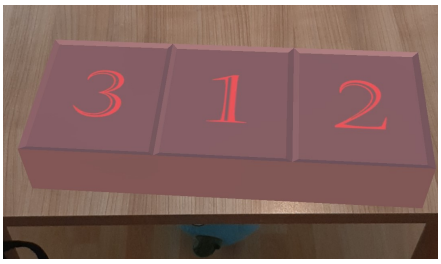
5.7 Second scenario

The second scenario revolves around the players entering the secret code to a code machine seen in image 5.23b. Only the host can see this machine. The client's game displays a machine with buttons numbered from one to three, visible in image 5.23a. The numbers under the buttons show the order of which letter wheel on the code machine they turn. It also references the numbers from the first scenario in image 5.22d. The numbers are needed because the model for the button box is symmetrical so if it was turned around, the players wouldn't notice and would be confused as to why the "left" button turns the "right" letter wheel. When the players enter the right code, the red bar on the bottom of the code machine will turn green as in image 5.23c and the players will not be able to interact with the object from this scenario anymore. There is also a hint for the first scenario on the back of the code machine, as seen in image 5.23d.

(a) : Client's game when the image called "WatchBox" is being tracked.

(b) : Host's game when the image called "WatchBox" is being tracked by the client.

(c) : Client's game when the image called "WatchBox" is not being tracked.



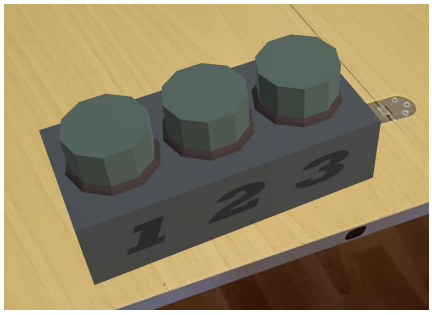
(d) : Host's game when the image called "WatchBox" is not being tracked by the client.

Figure 5.22: Images from the final version of the game of the first scenario from both the host's and the client's side.

5.7.1 Implementation

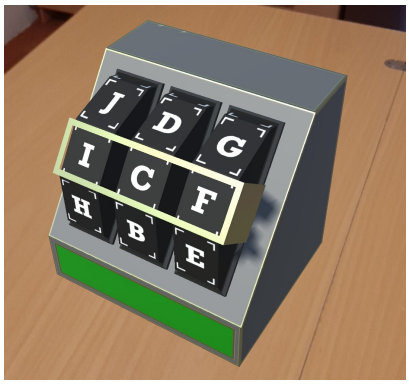
Because in this scenario, one of the players needs to push buttons, I used an implementation from 5.2. There, I explained how I created a way to trigger a specific method of a game object the player touched on the screen. This is implemented in "InputManager.cs". In summary, every time a player touches the screen, the script runs a method which captures the point on the screen the player touched and calculates the coordinates in the game space of that touch. Then, based on the camera rotation, casts a ray from that point. Afterwards, it returns the first game object with the collider component hit.

In this scenario, the game needs to recognise when the ray collides with a button. For this purpose, I added the Box Collider component to all three buttons and created a tag called "Button". Inside the "ButtonBehaviour.cs" script I created a public method called "ButtonClick" (image 5.24b), which is called from "InputManager.cs" script when a ray collides with a button (image 5.24a). This method starts a button click animation and calls a method of the "NetworkUIManager.cs" called "ButtonClick" passing it the button's ID. Since the buttons are always on the client's side, the "NetworkUIManager.cs" calls a server RPC called "ButtonClickServerRpc" (image 5.24c) which then finds the code machine's component "WheelBox.cs" in the game scene and



(a) : Client's game showing the button box.

(b) : Host's game showing the unsolved code machine.



(c) : Host's game showing the solved code machine.

(d) : Host's game showing the back of the code machine with a hint.

Figure 5.23: Images from the final version of the game of the second scenario from both the host's and the client's side.

calls the "RotateWheel" function (image 5.24d).

The "RotateWheel" function takes one argument saying which letter wheel should be turned. It starts an animation of rotating the wheel one letter and checks if the correct code has been entered. If yes, it changes the rendered material, notifies the "NetworkUIManager.cs", and sets a local boolean variable which prevents any further wheel rotations.

5.8 Third scenario

In this scenario, both players see the same 3D object. It is the puzzle box seen in figure 5.25. At first, this box is closed for both players as in image 5.25a. After the second scenario is finished and the correct code is entered into the code machine, the third scenario will become available. For the host, the box will open and they will see eight blank pieces inside, as seen in image 5.25c. If the host clicks on a puzzle piece next to an empty spot, the piece will move to that spot. The client's box won't open but the cover of the box will become transparent, like a window. The pieces inside the box will be

visible and each will display a piece of an image, as seen in image 5.25b. The players need to cooperate to put the pieces in the right spots to form the image. When they are done, the game finishes.

5.8.1 Implementation

The player interaction with the puzzle pieces works similarly to the buttons in scenario two in section 5.7. The "InputManager.cs" will check for the tag "PuzzlePiece". All puzzle pieces have this tag and they all contain the "PuzzlePieceScript.cs". They also all have a Cube Collider component. When a ray collides with a puzzle piece, its method called "Clicked" is called, which calls the method "PieceClicked" of the "PuzzleBoxScript.cs", passing it the ID of the piece. The puzzle box object contains the "PuzzleBoxScript.cs". The "PuzzleBoxScript.cs" keeps track of the state of the game using an array keeping track of each piece's position as seen in image 5.26a. The body of the "PieceClicked" method is visible and described in image 5.26b. In summary, it uses the "IsAdjacent" method, seen in image 5.26c, to find out whether the piece that was clicked by the player is adjacent to an empty spot. If yes, it moves the piece, notifies the "NetworkUIManager.cs", updates the puzzle state, and checks if the puzzle was solved. If yes, it notifies the "NetworkUIManager.cs". All of this only happens on the host's side of the game, because the client can't click the puzzle pieces. When the host's side notifies the "NetworkUIManager.cs" of puzzle piece movement, it then calls the client's side of the "PuzzleBoxScript.cs" to just move a piece. The client's side is not keeping track of the puzzle.

5.9 Final application

My game is designed as a mobile game anyone can play if they print out the correct images. It is easy to start playing but also contains all the needed parts for it to be implemented into a real-world escape room. For example, in appendix A, I explain how an overseer could be implemented, which would track the game's progress. This overseer could communicate with the real-world escape room and trigger real-life events in reaction to in-game events.

When the game is started, the player will see a main menu (image 5.27a). It describes how to connect both players to play the same game. I believe it is fairly intuitive, but it is not foolproof. It was originally not intended to be used by a casual user, but rather to be set up by someone with experience and handed to the players. In future, I would focus on either making the game only for real-life escape rooms and creating a manual for the operators, or on making it easy to play by casual players who download the game and the images from the internet.

When the players finish the sliding puzzle, an ending screen will fade in thanking them for playing the game, as seen in image 5.27b. The full play-through video of the final version of the game called Escape Pierito is

available in the digital appendix. The contents of the digital appendix are described in appendix C.

There was a slight issue with the image 5.12a, which sometimes kept being recognised in a different angle, making it twitch between the correct position and one leaning forward. In the future, I would change the image to a different one.

If I compare my game to the ones described in chapter 2, the most similar one would be AR-Escape[31]. It contains puzzles the players need to solve in AR. It uses image recognition, similar to my game, to create game objects the players can interact with. AR-Escape features a timer which counts down the time players have to solve all puzzles. My game isn't large enough to need a timer. My game is only cooperative but contains puzzles that always need both players to be solved. Unfortunately, I didn't have a chance to try the game out, so I can't compare the two games in more detail.

Escape AR Room: Pandemic[21], Scriptum[22], ARia's legacy[27], Prison Break AR[29] and Escape the Room: AR[28] are all AR escape room games developed in Unity. But they are all single-player only and depend on the AR Plane Manager to find the floor. The games then create the escape room on the floor. This approach is different from my game.

I can't compare my game to any Cluetivity[23][24][25] games because I didn't have access to them and I couldn't tell exactly how the games work from the available sources on the internet.

```
if (hit.transform.CompareTag("Button"))
{
    ButtonBehavior button = hit.transform.GetComponent<ButtonBehavior>();
    button.ButtonClick();
}
```

(a) : A part of the code from the "InputManager.cs" which calls the "ButtonClicked" method if a ray collides with an object with the tag "Button".

```
1 reference
public void ButtonClick()
{
    // if the button animation os running or if the button has been locked, return
    if (isClicking || locked) return;

    // start button animation
    isClicking = true;

    // call NetworkUIManager.cs's method and pass it the button's ID
    nium.ButtonClick(id);
}
```

(b) : The body of the "ButtonClicked" method inside the "ButtonBehaviour.cs" script.

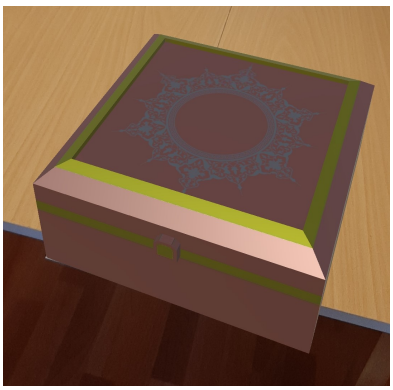
```
[ServerRpc(RequireOwnership = false)]
1 reference
private void ButtonClickServerRpc(int buttonNumber)
{
    WheelBox wbox = FindObjectOfType<WheelBox>();
    if (!wbox) return;

    wbox.RotateWheel(buttonNumber);
}
```

(c) : A server RPC inside the "NetworkUIManager.cs" called after a button was clicked.

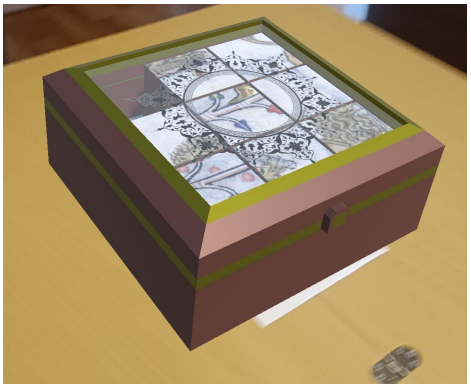
```
// called by NetworkUIManager when a button is clicked
4 references
public void RotateWheel(int wheelNum)
{
    // dont move wheels if the correct code is entered
    if (solved) return;

    // update the entered code and start wheel rotation animation
    switch (wheelNum)
    {
        case 1:
            code1 += 1;
            while (code1 >= MaxInt) code1 = code1 % MaxInt;
            break;
        case 2:
            code2 += 1;
            while (code2 >= MaxInt) code2 = code2 % MaxInt;
            break;
        case 3:
            code3 += 1;
            while (code3 >= MaxInt) code3 = code3 % MaxInt;
            break;
    }

    // if the correct code is entered, mark solved
    if (code1 == correctCode[0] && code2 == correctCode[1] && code3 == correctCode[2])
    {
        Solved();
    }
}
```

(d) : The body of the "RotateWheel" method inside the "WheelBox.cs" rotating one letter wheel one step.

Figure 5.24: Code snippets from the implementation of the second scenario.



(a) : Both players' game showing the closed puzzle box.



(b) : Client's game showing the puzzle box with a transparent cover.

(c) : Host's game showing the open puzzle box.

Figure 5.25: Images from the final version of the game of the second scenario from both the host's and the client's side.

```
// position of each piece gameBoard[0] = position of the empty space, gameBoard[1] = position of the piece 2 etc.
private int[] gameBoard = { 0, 3, 5, 2, 8, 7, 4, 1, 6 };

//                      |1|8|4|    |1|2|3|
// - initial position = |2|7|3| -> |4|5|6| = final position
//                      |9|6|5|    |7|8|9|
```

(a) : An array representing the puzzle state.

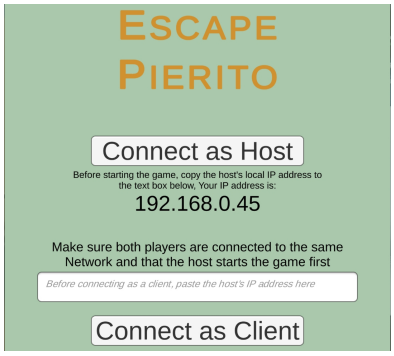
```
// a method called by a puzzle piece when it is clicked passing it the piece's ID
9 references
public void PieceClicked(int stringId)
{
    if (!movable) return;

    // convert id [1-9] to [0-8]
    int id = stringId - 1;
    // check if the piece that was clicked is adjacent to the empty place, if yes, return in which direction the piece can move
    Directions adjacency = IsAdjacent(gameBoard[0], gameBoard[id]);
    // if the piece is not adjacent to an empty space, return
    if (adjacency == Directions.Invalid) return;

    // move piece in the direction of the empty place and notify the NetworkUIManager
    MovePiece(stringId, adjacency);
    nium.PieceMoved(stringId, adjacency);

    // update the local representation of the puzzle
    int tmp = gameBoard[0];
    gameBoard[0] = gameBoard[id];
    gameBoard[id] = tmp;

    // check if the puzzle is solved, if yes stop the pieces from moving again and notify the NetworkUIManager
    if (IsFinished())
    {
        nium.GameWon();
        movable = false;
    }
}
```

(b) : The body of the "PieceClicked" method.

```
// returns the relation between two pieces of the puzzle
1 reference
private Directions IsAdjacent(int empty, int piece)
{
    if (empty - 1 == piece && empty % 3 != 0) return Directions.Right;
    if (empty + 1 == piece && empty % 3 != 2) return Directions.Left;
    if (empty + 3 == piece) return Directions.Up;
    if (empty - 3 == piece) return Directions.Down;
    return Directions.Invalid;
}
```

(c) : A method determining the adjacency of two puzzle pieces.

Figure 5.26: Code snippets from the implementation of the third scenario from the "PuzzleBoxScript.cs".



(a) : An image of the main menu of the game, explaining how to connect.

(b) : The end screen thanking the players for playing.

Figure 5.27: UI parts of the game. Both are cropped from the original phone size, and they both contain my signature at the bottom.

Chapter 6

Testing

Testing an application on users is always an important part of the development process. Not only to understand if there would be demand for such a product but also to reveal any missing functionalities or details that would improve the user experience. During development, I regularly consulted my supervisor prof. Ing. Jiří Žára CSc., but even then I missed a few essential details that came to light during the testing.

6.1 Testing process

The testing was conducted in separate pairs of testers, except for one person who tested the game alone with me replacing the second player. Eleven people participated overall in the course of one week.

Before the testing started I placed images from section 5.4, specifically images 5.12 and 5.10a, on a table and prepared both devices for testing, starting the app, connecting the devices and making sure everything works as it should. When the pair of testers came, I gave an introduction speech. Afterwards, the testers were supposed to play until they solved the first two tasks, ending when they put the correct code in the code machine, which then changed its material to notify the testers. During this time, I only watched them and was taking notes on how they were doing and understanding the game. Occasionally, I would try to push the testers in the correct direction without revealing too much. After finishing the task, I asked the testers to fill out a questionnaire I prepared and printed beforehand. Both the introduction speech and the exact form of the questionnaire are available in the appendix B.

The game was not tested in its final state. It included some of the debugging elements to know as soon as possible, whether the game is running as intended, to not waste time. There were no technical issues with my game during the testing.

6.2 Testing results

All pairs have successfully finished the part of the game, except for one, because the network stopped working during the testing and it would be too lengthy to start over again. Luckily they were close to finishing so they figured out the solution afterwards on their own. It usually took the testers to finish this part of the game around 10 minutes.

Most people first tried clicking the buttons on the button box in the order given by the numbers on the code box (the box displaying numbers when the watch box is not observed and letters of the code otherwise). Afterwards, they tried to input the scrambled code. At this point, some testers thought they solved the puzzle. Not long after, they figured out that the code was scrambled and the numbers showed the correct order of the letters. This is exactly how I intended the players to think when solving the puzzle.

I created a table 6.1 showing the statistics of answers to all closed questions. The answers have been changed to numbers from 1. This means that in the first part the answer "Not familiar" equals 1 and the answer "Expert" equals 4. Similarly, in the second part the answer "Definitely agree" equals 1 and the answer "Definitely disagree" equals 5. All numbers in the table are rounded to two decimal places. There are four possible answers in the first part and five in the second.

Question number	Mean	Mean answer	Variance	Median	Range of answers
1.1	2.09	Somewhat familiar	0.69	Somewhat familiar	1-3
1.2	2.64	Decently familiar	1.25	Decently familiar	1-4
1.3	2.73	Decently familiar	1.22	Somewhat familiar	1-4
1.4	1.91	Somewhat familiar	1.09	Somewhat familiar	1-4
1.5	1.64	Somewhat familiar	0.85	Not familiar	1-4
2.1	2.18	Slightly agree	1.36	Slightly agree	1-4
2.2	1.18	Definitely agree	0.16	Definitely agree	1-2
2.3	1.73	Slightly agree	1.02	Definitely agree	1-4
2.4	1.45	Definitely agree	0.27	Definitely agree	1-2
2.5	4.18	Slightly disagree	0.97	Slightly disagree	2-5
2.6	1.18	Definitely agree	0.16	Definitely agree	1-2

Table 6.1: Statistics of closed questions of the questionnaire with the most important features listed. All numbers are rounded to two decimal places.

This helped immensely when interpreting the answers later on. I could not only tell what the average answer was but also how much they were influenced by outliers and how unanimous the answers were.

6.2.1 Questions on testers' background

The first question of the first part enquires about the tester's familiarity with real-life escape rooms. The question is included because people more familiar with this topic might have more constructive and relevant answers for open questions relating to escape room tasks and puzzles. There was no one identifying as an expert in my group of testers, the group was evenly distributed between the other three answers. This means there are answers

from all kinds of users.

The second question is about familiarity with video games. It would be preferable to have people of all levels of familiarity with video games since my game blends the genre of escape rooms and video games. This was achieved, there were four testers in all categories, except the "Not familiar" which had three.

The next three questions focus on whether the tester is familiar with topics relating to the development of my game. This means that, for example, if a person considering themselves an expert in Augmented Reality and Unity Game Engine answered that there is an implementation issue resulting in unexpected behaviour, I would give more value to that answer, than to that of a person not familiar with the topics. Answers to these questions were polarised, people usually stating they are an "Expert" or "Not/Slightly Familiar". This splits my testing group into casual users and people with a deeper understanding of the topic.

There were a considerable number of experts in programming. There were only a few people familiar with Augmented reality and only one tester who was an expert in Unity Game Engine, while everyone else answered that they are "Not/Somewhat familiar". The same person is also an expert in all other questions of part one, except the real-life escape rooms.

6.2.2 Questions on testers' experience with my game

The second part of the questionnaire starts by asking about the stability of the game. Most people answered they "Definitely agree" or "Slightly agree" that the experience felt stable. The only exceptions were people more familiar with Augmented reality and the Unity Game Engine. This means that the casual players don't mind some level of lability considering the game scene synchronisation with the camera image. While watching the testers play the game I noticed that about half of the time during testing the game worked perfectly fine and the other half it had some issues with stability. In my opinion, this is due to the different lighting. In one case the testing was cut short because one of the phones disconnected from the wireless network.

The answers for whether AR can enrich the escape room experience, whether the task presented was enjoyable, and whether they would like to visit an escape room utilizing augmented reality unanimously agreed with the statements.

Most people disagreed that the AR can make the experience less enjoyable because of its constraints, except for two people. One person stated that there are scenarios which will always be better with physical objects. I agree that sometimes we might create extra steps for the sake of those steps and that one should think about what to use AR for and when it is excessive and might ruin the enjoyment.

Similarly, everyone except two people agreed that the task presented was intuitive. I can confirm that all testers finished the task within a reasonable time and no one got stuck. One person stated that it is similar to real-life

escape rooms with the difference that the two people interacting would usually be standing further away from each other so they have to cooperate.

6.2.3 Open-ended questions on testers' opinions

The most common answer in the third part was that there was no sound in the game. Adding sounds would give players more than just visual feedback. This would help the players orient themselves in the game much better. One respondent stated that they would like to have "some music" in the game. I don't think background music would add much to the game experience and it would add the need to synchronise the music from both devices so it doesn't become an uncomfortable mesh of sounds because of music duplicity. It would make more sense to add background music using another device inside the physical escape room.

Another important answer was about localisation. The tracked images should relate to the theme of the game. I agree with the statement and I would add that the style should be similar inside the game as well, which I didn't manage to do when creating the 3D models. This can be even further emphasised when creating a real-world escape room using AR. The tracked images can be scattered around the room having the same theme as the room itself. In the manual (appendix A), I explain how to change which images the app tracks and how to exchange the 3D models.

Another tester wrote that they would appreciate a hint button. I tried to incorporate hints inside the 3D models, for example at the back of the code machine. I think it is more engaging this way, but some players might appreciate a hint button more. Since only one person pointed this out I don't think it is necessary, but it is a good point to consider.

One person pointed out that the game might need a tutorial for the controls. I didn't notice anyone having any issues with the controls, except one person who tried controlling the game using hand gestures instead of tapping the screen. I would say that tapping the screen is more natural for a casual user, but I agree it is not obvious for everyone and a tutorial is a thing I would consider adding to the game in the future.

6.3 Testing conclusion

In conclusion, the game isn't as stable as I would like it to be, but most people didn't mind and stated it didn't impact their enjoyment. This can be enhanced in a few ways. The first way is to have the tracked images solidly set in the physical room and not move them. Then, it would help to have really good lighting in the room, and a good local network, so the game doesn't desynchronise. The most important thing to add in the future is sounds to make the player more immersed and to indicate to them when the scene changes, since visually it might not be apparent. People liked the idea of the task presented to them and enjoyed the experience using AR. It seems people would like to see AR incorporated more into real-world escape rooms.

Chapter 7

Conclusion

The goal of this bachelor's thesis was to create an AR escape room for multiple players who need to interact in AR to solve puzzles.

For this purpose, I analysed existing AR escape rooms, especially multiplayer ones, and the tools for creating AR games, and games using network communication. Based on this analysis, I chose development tools and created a game design featuring three scenarios in which the players need to cooperate using AR.

I then described the process of developing the game and commented on the choice of development tools. During development, I realised my choice of the third scenario wasn't good from the gameplay and fun perspectives. It is too hard and lengthy and the players will likely lose interest in trying to solve the puzzle as did one pair of testers who wished to play the rest of the game as well.

I have been very satisfied with the choice of Unity's AR Foundation. I ran into problems using Unity's Netcode for GameObjects, but ultimately I was satisfied with the choice as it wasn't hard to create functioning network communication. These tools are the most important for my game as the game's aim is to connect augmented reality and network multiplayer.

When the game was complete, I tested it using eleven people split into pairs. Only the first two scenarios were tested and I got valuable feedback. The most important complaint was that the game is missing sound effects, for example, when the player clicks a button or when a scenario is finished. The image recognition wasn't as stable as I hoped it to be but most people stated it didn't impact their enjoyment of the game. The images used for the image recognition are not related at all to the theme of the game because I focused on choosing good, stable images. In the future, I would focus on fixing these issues.

In the appendix A, I created a manual on how to extend my game in detail. It describes the Unity project in more detail and explains how to progress when implementing new tasks.

Overall, I believe my game showcases how to connect AR and multiplayer escape room games well. It creates unique scenarios which wouldn't be possible in real-life escape rooms. It demonstrates how to create a game anyone can play and how to incorporate AR into real-world escape rooms.

Bibliography

[1] Apple.com (2024). Dive into the world of augmented reality https://developer.apple.com/augmented-reality/ ([online] Accessed: 19. 1. 2024)

[2] Apple.com (2024). More to explore with ARKit 6 https://developer.apple.com/augmented-reality/arkit/ ([online] Accessed: 19. 1. 2024)

[3] Apple.com (2024). RealityKit https://developer.apple.com/augmented-reality/realitykit/ ([online] Accessed: 19. 1. 2024)

[4] Apple.com (2024). Creation tools for spatial apps https://developer.apple.com/augmented-reality/tools/ ([online] Accessed: 19. 1. 2024)

[5] Google.com (2024). Make the world your canvas https://developers.google.com/ar ([online] Accessed: 19. 1. 2024)

[6] Google.com (2024). Overview of ARCore and supported development environments https://developers.google.com/ar/develop ([online] Accessed: 19. 1. 2024)

[7] Khronos.org (2024). OpenXR https://www.khronos.org/openxr/ ([online] Accessed: 19. 1. 2024)

[8] Unity.com (2024). Augmented Reality https://unity.com/unity/features/ar ([online] Accessed: 19. 1. 2024)

[9] Unity.com (2024). AR Foundation https://unity.com/unity/features/arfoundation ([online] Accessed: 19. 1. 2024)

[10] Microsoft.com (2024). Microsoft HoloLens 2 https://www.microsoft.com/en-us/hololens ([online] Accessed: 19. 1. 2024)

[11] Magicleap.com (2024). The most immersive enterprise AR device https://www.magicleap.com/en-us/ ([online] Accessed: 19. 1. 2024)

[12] Unity.com (2024). Unity MARS https://unity.com/products/unity-mars ([online] Accessed: 19. 1. 2024)

[13] Unity.com (2024). Unity as a Library https://unity.com/features/unity-as-a-library ([online] Accessed: 19. 1. 2024)

[14] Unity3D.com (2024). XR Interaction Tool https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.5/manual/index.html ([online] Accessed: 19. 1. 2024)

[15] Vuforia.com (2024). Vuforia Engine https://developer.vuforia.com/library/getting-started/getting-started-vuforia-engine-unity ([online] Accessed: 19. 1. 2024)

[16] XRBootCamp.com (2024). Comparing Unity vs Unreal for VR, MR or AR Development Projects https://xrbootcamp.com/unity-vs-unreal-engine-for-xr-development/ ([online] Accessed: 19. 1. 2024)

[17] VentionTeams.com (2024). How to choose an engine for VR: Unity vs Unreal https://ventionteams.com/blog/unity-vs-unreal-virtual-reality ([online] Accessed: 19. 1. 2024)

[18] UnrealEngine.com (2024). Augmented reality overview https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/XRDevelopment/AR/HandheldAR/AROverview/ ([online] Accessed: 19. 1. 2024)

[19] Unity.com (2024). How to choose the right netcode for your game https://blog.unity.com/games/how-to-choose-the-right-netcode-for-your-game ([online] Accessed: 19. 1. 2024)

[20] Unity3D.com (2024). About Netcode for GameObjects https://docs-multiplayer.unity3d.com/netcode/current/about/ ([online] Accessed: 19. 1. 2024)

[21] Emergereality.com (2024). Escape Room AR http://www.emergereality.com/escaperoom.html ([online] Accessed: 19. 1. 2024)

[22] Brandon Chow (2023, 8. června). Adver2Play - Scriptum AR Escape Room Adventures https://www.escapetheroomers.com/post/adver2play-scriptum-ar-escape-room-adventures ([online] Accessed: 19. 1. 2024)

[23] Cluetivity.com (2024). Become a Secret Agent! https://www.cluetivity.com/operation-mindfall-outdoor-game/ ([online] Accessed: 19. 1. 2024)

[24] Cluetivity.com (2024). Blackout: New Outdoor Game https://www.cluetivity.com/blackout/ ([online] Accessed: 19. 1. 2024)

[25] Cluetivity.com (2024). The Fantasy Epos https://www.cluetivity.com/magic-portal-outdoor-game/ ([online] Accessed: 19. 1. 2024)

[26] Escape-Team.com (2024). Escape Team https://www.escape-team.com ([online] Accessed: 19. 1. 2024)

[27] AriasLegacy.com (2024). Aria's Legacy https://www.ariaslegacy.com ([online] Accessed: 19. 1. 2024)

[28] Apple.com (2024). Escape The Room: AR https://apps.apple.com/us/app/escape-the-room-ar/id1329567068 ([online] Accessed: 19. 1. 2024)

[29] Google.com (2024). Prison Break AR Escape Room https://play.google.com/store/apps/details?id=com.RiddleGames.PrisonBreak&hl=en&gl=US&pli=1 ([online] Accessed: 19. 1. 2024)

[30] Fandom.com (2024). The Walking Dead: Our World https://walkingdead.fandom.com/wiki/The_Walking_Dead:_Our_World ([online] Accessed: 19. 1. 2024)

[31] Plecher, D., Ludl, M., & Klinker, G. (2020). Designing an AR-escape-room with competitive and cooperative mode. In GI VR/AR Workshop. Gesellschaft für Informatik eV.

[32] AIMEscape.com (2024). Augmented Reality Outdoor City Experience https://aimescape.com/ar?gad_source=1&gclid=Cj0KCQiAnrOtBhDIARIsAFsSe509WXUWUjjuT59cjQhm-y-lcO876uEsSYXiKnNxtCex5VcL-WkyjFIaAvehEALw_wcB ([online] Accessed: 19. 1. 2024)

[33] Unity3D.com (2024). Testing multiplayer games locally https://docs-multiplayer.unity3d.com/netcode/current/tutorials/testing/testing_locally/ ([online] Accessed: 19. 1. 2024)

[34] Unity.com (2024). Joystick Pack https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631 ([online] Accessed: 19. 1. 2024)

[35] Onirix.com (2024). Image Tracking https://docs.onirix.com/onirix-studio/projects/image ([online] Accessed: 19. 1. 2024)

[36] Zap.Works (2024). What makes a good tracking image? https://docs.zap.works/general/design/what-makes-good-tracking-image/ ([online] Accessed: 19. 1. 2024)

[37] Alamy.com (2024). Judas Priest performs in concert at the Seminole Hard Rock Hotel and Casino in Hollywood, Florida on August 17, 2009. https://www.alamy.com/judas-priest-performs-in-concert-at-the-seminole-hard-rock-hotel-and-casino-in-hollywood-florida-on-august-17-2009-image220283219.html ([online] Accessed: 19. 1. 2024)

[38] Google.com (2024). The arcoreimg tool https://developers.google.com/ar/develop/augmented-images/arcoreimg#windows ([online] Accessed: 19. 1. 2024)

[39] pngtree.com (2024). Single Color Islamic Ornaments Round Vector PNG Images https://pngtree.com/freepng/single-color-islamic-ornaments-round_5582082.html ([online] Accessed: 19. 5. 2024)

[40] x.com (2024). musical lion, france, 14th century https://x.com/WeirdMedieval/status/1713262385534845384?t=Zw67jC56MYcp7KvQ-GSpjw&s=19 ([online] Accessed: 19. 5. 2024)

[41] YouTube.com (2024). Tap to Place Objects in Unity's AR Foundation - Tutorial 2023 https://www.youtube.com/watch?v=lYDfV-GaKQA&t=3s ([online] Accessed: 19. 5. 2024)

[42] Unity Documentation, Unity3d.com (2024). Camera.ScreenPointToRay https://docs.unity3d.com/ScriptReference/Camera.ScreenPointToRay.html ([online] Accessed: 19. 5. 2024)

[43] Unity3D.com (2024). About Netcode for GameObjects https://docs-multiplayer.unity3d.com/netcode/current/about/ ([online] Accessed: 19. 5. 2024)

[44] Unity3D.com (2024). ServerRPC https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/serverrpc/ ([online] Accessed: 19. 5. 2024)

[45] YouTube.com (2024). Let's Make an AR App in 15 MINUTES!! (Beginner Friendly) https://www.youtube.com/watch?v=GfS72wqKQ_g&t=13s ([online] Accessed: 19. 5. 2024)

[46] Vecteezy.com (2024). abstract organic pattern design background https://www.vecteezy.com/vector-art/8599403-abstract-organic-pattern-design-background ([online] Accessed: 19. 5. 2024)

[47] Craiyon.com (2024). View from inside a cave looking out into a sunny valley with a wooden cross on a mountain https://www.craiyon.com/image/GadKGSufRXOf5Q_enUs_FQ ([online] Accessed: 19. 5. 2024)

[48] Denik.cz (2024). Z Moravy na Metalfest v Plzni https://prostejovsky.denik.cz/volny-cas/z-moravy-na-metalfest-v-plzni-20180529.html ([online] Accessed: 19. 5. 2024)

[49] Nanonets.com (2024). What is image recognition https://nanonets.com/blog/image-recognition/#:~:text=Image%20recognition%20algorithms%20use%20deep,and%20features%20of%20different%20objects. ([online] Accessed: 19. 5. 2024)

[50] Mindar.org (2024). How to Choose a Good Target Image for Tracking in AR - Part 3 https://www.mindar.org/how-to-choose-a-good-target-image-for-tracking-in-ar-part-3/ ([online] Accessed: 19. 5. 2024)

[51] Johnson, Doug (1982). Judas Pries, Screaming for Vengeance [album cover art].

[52] Wikipedia.com (2024). File:Avatar - Rock am Ring 2018-5719.jpg https://en.m.wikipedia.org/wiki/File:Avatar_-_Rock_am_Ring_2018-5719.jpg ([online] Accessed: 19. 5. 2024)

[53] YouTube.com (2024). Augmented Reality (AR) tutorial for beginners using Unity 2022 https://www.youtube.com/watch?v=gpaq5bAjya8&t=1356s ([online] Accessed: 19. 5. 2024)

[54] VanSwearingenPhotography.com (2024) Zebra 2014 https://www.vanswearingenphotography.com/large-multi-view/Animalia/2522688-3-198475/Photography/zebras-i.html ([online] Accessed: 19. 5. 2024)

Appendix A

Manual

A.1 Introduction

This manual explains how to build on my implementation of an Augmented Reality (AR) multiplayer escape room game called Escape Pierito and how to add more features. It may also act as a demonstration of how to create your own game. A general knowledge of Unity and C# programming language is expected. My implementation is built on image recognition and sharing important information between multiple instances of the game; thus, each player may see a bit of a different environment and need to cooperate to achieve progress. This manual shows a project made in Unity 2022.3.13f1, using packages supporting AR and Netcode for GameObjects, which is Unity's framework for network communication. The full list of packages used is visible in figure A.1.

These are the most important packages. ParrelSync lets you create identical clones of your project that are always synchronised. This is very helpful for running two instances of your game on your local computer for fast debugging. Android Logcat shows console messages from any Android phone connected to your computer by a USB cable. AR Foundation is Unity's unified framework for creating AR apps, thanks to which one can create games for Android using Google's ARCore and Apple using ARKit in one project using the same code. Netcode for GameObjects is Unity's framework for network communication. Other packages are either required for already mentioned ones or are not necessary for understanding this manual.

The project creates one executable file that can be run on Android 7.0 'Nougat' and newer. The game is currently intended for two players, but in later stages, I explain how this can be changed to accommodate any number of players, if needed. After starting the application, each player has to choose whether they are the host or the client. The host always has to connect first, and the client has to fill in the host's local IP address before continuing. Note that both players have to be connected to the same network. In theory, multiple clients can connect to the same host, but their game experience is not going to be unique.

The game is based on image recognition. It is essential, before playing the game, to print out all the images used. In the current stage of the game, only

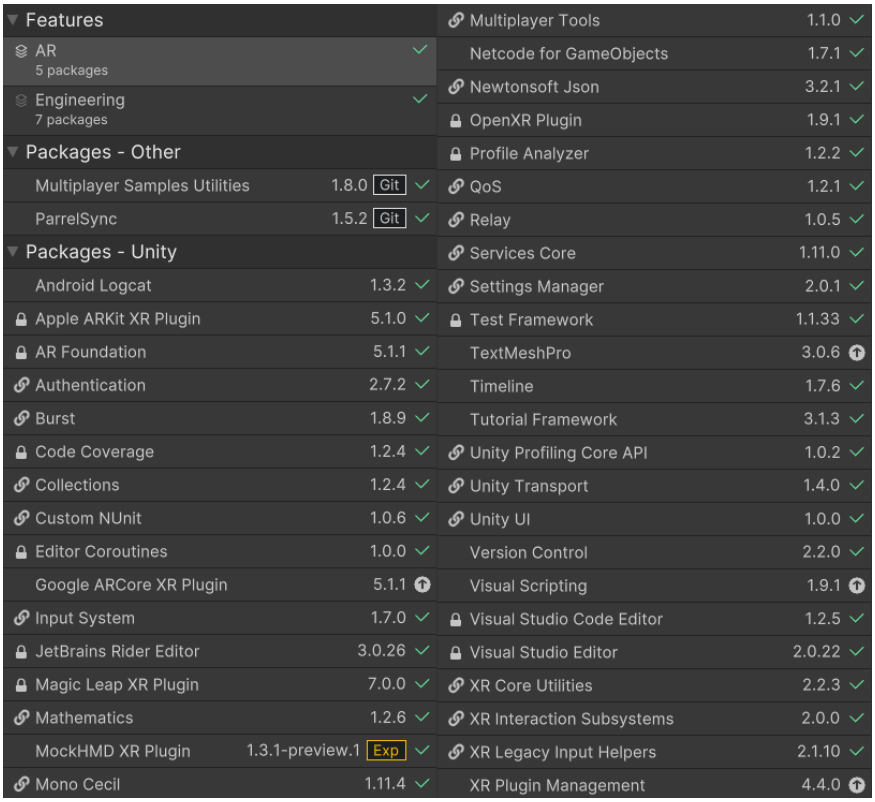


Figure A.1: Screenshot of all used packages (Unity -> Window -> Package Manager) with active versions.

five images are used, so it can be played in a fairly small room. For the best experience, play the game in a well-lit environment, using large static images printed on paper.

A.2 Common Issues

As of the 10th of April 2024, these aren't the most up-to-date versions of the packages, and they may include bugs or deprecated practices. I will try to mention deprecated practices when this manual encounters them. I will list certain bugs I've run into while developing this app. The first issue is in Multiplayer tools. When building the app, some parts of this package may be stripped and cause network multiplayer to not work, throwing the following error, "Error Unity Failed to load type initialization for assembly Unity.Netcode.Runtime". This is fixed in version 2.0.0 and can also be fixed in lower versions when unchecking "Strip Engine Code" and setting "Managed Stripping Level" to "Minimal" in File -> Build Settings... -> Player Settings... -> Settings for Android -> Other Settings. More information about this issue can be found here, `https://forum.unity.com/threads/netcode-error-unity-failed-to-load-type-initialization-for-assemby.1375914/`. The second bug is produced on certain versions of

Android. After starting the app, the screen turns black, and an error occurs on the console stating: "Skipped rendering frame because GfxDevice is in invalid state (device lost)". If the user turns off the screen and locks the phone (pushes the power button) and then turns the screen back on and unlocks the phone (pushes the power button again), the issue is solved after a few seconds. This can be annoying while debugging, but it is resolved in newer versions of Unity. You can read more about this issue here, `https://forum.unity.com/threads/issue-with-android-skipped-rendering-frame-because-gfxdevice-is-in-invalid-state-device-lost.1449145/#post-9136096` and here, `https://issuetracker.unity3d.com/issues/android-black-screen-with-skipped-rendering-frame-and-error-gfxdevice-is-in-invalid-state-device-lost-when-player-is-built-with-google-arcore`.

A.3 Basic Information about my Unity project

In figure A.2, there is the hierarchy of my AR scene in Unity. AR Session, AR Session Origin and AR Camera are responsible for the correct functioning of AR in an application.

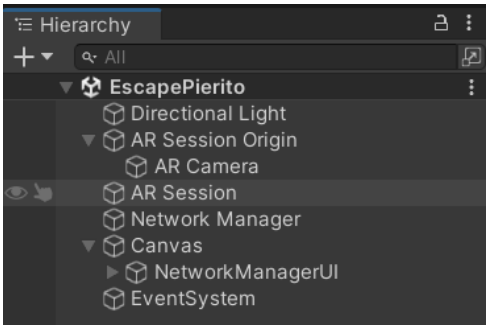


Figure A.2: Screenshot of the hierarchy of objects in my Unity scene.

The AR Session controls all the main AR synchronisation parts and controls the lifecycle of an AR app. More about the AR Session in the Unity documentation, `https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.1/manual/features/session.html`.

The AR Session Origin manages all the smaller and not vital parts of AR, such as plane recognition, AR raycasting, anchoring, image recognition and more. It is also a parent of the AR Camera, creating an origin of AR Session space in Unity 3D space. The AR Session Origin is currently deprecated and is replaced by XR Origin, which serves the same purpose, and since AR and VR have many common features, Unity has decided to merge AR and VR Origins. More on the AR Session Origin, `https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/api/UnityEngine.XR.ARFoundation.ARSessionOrigin.html#:~:text=An%20ARSessionOrigin%20is%20the%20parent,as%20planes%20or%20point%20clouds` and on the XR origin, `https://docs.unity3d.com/Manual/xr-origin.html`.

Network Manager includes “Network Manager” and “Unity Transport” components, which establish communication between multiple instances of an application, determine who is a server or host and which clients connect where, how they communicate and more. As Unity is a game engine, the user can specify a prefab for a player object, which will be instantiated for each client connected to a server or host. More on the Network Manager component, `https://docs-multiplayer.unity3d.com/netcode/current/components/networkmanager/` and the Unity Transport component, `https://docs.unity3d.com/Packages/com.unity.transport@2.0/manual/index.html`.

Under regular Unity Canvas there is NetworkManagerUI. This is an empty object originally intended to only manage UI elements related to networking, for example, connect and disconnect buttons. NetworkManagerUI is a network object and contains my custom component called NetworkUIManager, which inherits NetworkBehavior. This means it is a component that can communicate with equivalent components from other users connected to the same server or host. This can be done using ServerRPCs, requests from clients to run a function on the server or host version of the game, ClientRPCs, requests to run a function on all client instances (be careful, a host is a server and a client in one, so ClientRPCs will also run on the host's side), and Network Variables, which can only be written into and read from by specified users. I am not using Network Variables in my application, but if you want to know how they work, you can read about them here, `https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/`.

ServerRPCs and ClientRPCs are deprecated and are replaced simply by RPCs, in which a user can specify the same attributes as before and needs to specify who the RPC is for. For instance, instead of declaring SeverRPC by writing [ServerRPC] followed by a definition of a function whose name ends in ServerRPC, you would use [RPC(SendTo.Server)] followed by a definition which ends in RPC (if it ended in ServerRPC, it might use the legacy version of ServerRPC). Here you can read more about the ServerRPC, `https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/serverrpc/`, about the ClientRPC, `https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/clientrpc/`, and the new RPCs, `https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/rpc/`.

Next, I will focus on the structure of the Assets folder in my project as seen in figure A.3. I will only mention the parts that are specific to my game, otherwise it is a pretty standard hierarchy of folders. The BlenderMeshes folder contains a folder for each blender mesh I have created and exported into a .fbx file. The References folder contains images for image recognition and the Reference Image Library object, which is required for the AR Tracked Image Manager component in AR Session Origin. ClientPrefabs, HostPrefabs and SharedPrefabs contain prefabs for objects to be generated when a specific image is recognised. Since clients and the host can see different objects, I

divided all prefabs into these three folders.

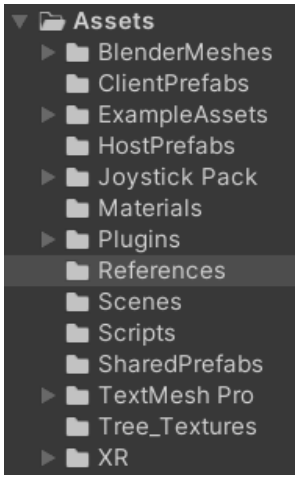


Figure A.3: Contents of the Assets folder of my project.

A.4 Adding new AR objects

There are many ways to add more complex 3D models to Unity. I used the 3D modelling tool Blender and will describe briefly how to use meshes from Blender. If you prefer another way of importing 3D models into Unity, you may skip this paragraph. After creating a model for the game in Blender, you need to export it as a .fbx file (in Blender 3.6 File -> Export -> FBX (.fbx)). It is advised to also bake the albedo, metallic and roughness textures of each model to create a material in Unity closer to the one in Blender. Unity uses albedo and metallic-smoothness for textures. Therefore, you need to first transform metallic and roughness textures into one where the roughness' texture's colours are inverted and applied as an alpha mask to the metallic texture. You can then move these two textures into the "Material" folder in the Assets of my Unity project and preferably create a new folder inside for each model you want to import to Unity. Then you can create a new material, in Unity you will see settings as in figure A.4, and by clicking the dots next to "Albedo" and "Metallic" settings, you can select the newly added textures. If needed, you can also set other properties, for example, a normal map. If you want to create a partly transparent object in Unity, you need to create a texture using the alpha channel for the needed transparency and then set the "Rendering mode" of the material to "Transparent". Then you should create a folder for your 3D model in the "BlenderMeshes" folder and put your .fbx file there.

After that, create a prefab using your imported model with the correct materials applied and paste it into the "ClientPrefabs", "HostPrefabs" or "SharedPrefabs" folder, depending on whom the model is intended for. Then paste the prefab into a component of AR Session Origin called Image Recognition Script under the "Host Prefabs" or "Client Prefabs" as seen in the

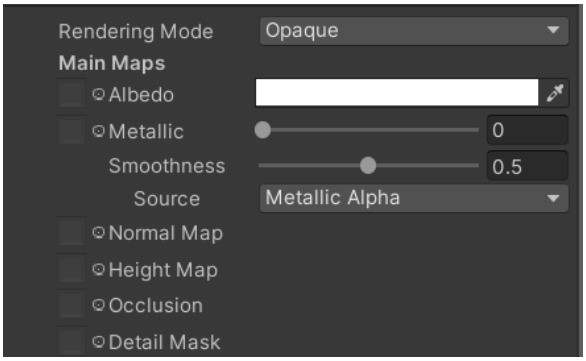


Figure A.4: Material settings in Unity.

figure A.5. This will enable these prefabs to be created when the image linked to the prefab is recognised in the 3D world. Make sure your prefabs are positioned in the centre of the local coordinate system, as it may lead to unwanted positioning of the object in the scene relating to the position of the recognised image. The prefabs which share the same recognition image must be named the same. Paste the image you want to be recognised by the programme in the "References" folder. In the "Reference Image Library" in the same folder, click on "Add Image" (on the bottom) and select your new image. Change the name of that image inside the "Reference Image Library" to the same name as the prefabs that are to be created when this image is recognised.

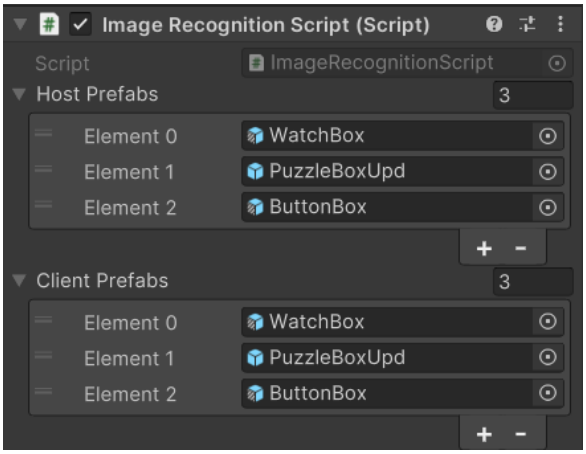


Figure A.5: Lists of prefabs for both players inside the ImageRecognition-Script.cs, as visible in the Unity editor.

For image recognition to work well, you should keep in mind a few rules when choosing an image. First, it is always better to have an image with the same number of pixels in width and height and preferably both the numbers to be even. The more complex the image, the better. Real-life photographs work well because they are imperfect. Make sure that the image stays complex after the colours are changed to brightness. The more the

brightness changes, the better. Try to have the complexity evenly distributed throughout the image, the quality of the image is much lower when one part of the image recognition is complex, but the rest is plain. It is beneficial if the image is in no way symmetrical since the image recognition might decide that the image is in a different place or orientation than it is. This could create a lot of inconsistencies when game objects are created linked to these images. For general orientation, whether your image is good for image recognition, you may use a tool created by Google called The arcoreimg tool, which can grade your image on a scale from 0 to 100. Be careful though, if an image gets a grade of 100, it doesn't mean it will perform well under all circumstances. In my experience, some images with lower grades perform better in praxis. If you want to find out more about this tool, check out this site, https://developers.google.com/ar/develop/augmented-images/arcoreimg#windows. If you want to use your models for my already implemented parts of the game, first make sure the new model is compatible with all that my current model does. For example, my current model for the Code Machine, as visible in the figure A.6, includes three wheels with 10 letters. If you don't want to change my scripts and make sure every detail works, your new model should have the same. Always check the scripts connected to the prefabs, because sometimes there are constraints. For example, the three wheels that are parented to the main object have to be called "Wheel1", "Wheel2" and "Wheel3" for the WheelBox.cs script to work correctly.

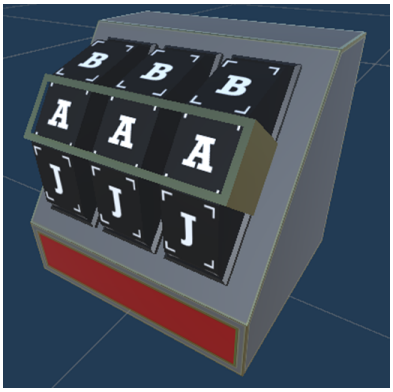


Figure A.6: An image of an asset I created for the game for inputting a number code.

A.5 Creating scripts and network communication

Now, when the application recognises your image, it should create the appropriate object in the 3D game space. But so far, these prefabs don't communicate with the rest of the app. You can treat the object prefabs exactly as any other Unity objects, for example, if you want your button to do a button animation, it is no different from in a non-AR game. One of the

tasks in my game is for the two players to fill in the right code in a machine. The catch is that one of the players controls the buttons and the other one sees the code machine. I will demonstrate some of the implementation techniques in this puzzle.

As seen in A.7, all communication between the two instances of my game goes through the Network UI Manager. Network UI Manager inherits the NetworkBehaviour class, letting the script communicate with its counterpart on the other device. One of the players is a host and the other is a client. This script is separated into different parts for clarity. The "UI declarations" part is for everything related to the UI, mainly UI buttons and text fields. Followed by the "Logic Declarations" part where we declare all our variables needed for the script, for example, a boolean variable "isHost", which tells the script whether the owner is a host or not. Then there is the "Awake" method that performs all tasks needed to be performed when the script is first created. "UI methods" contain all the methods for the UI logic. The "Task synchronisation calls" contain all public methods called from local scripts to communicate with the other player's scripts. Lastly, the "ServerRPCs" and "ClientRPCs". An RPC is a request for a specific method to perform on another instance of the script. In my project, I use a ClientRPC for a method to be performed on the client's side and ServerRPC to be performed on the host's side. It is important to remember that a host is a server and a client at the same time, so we need to make sure that when a host calls ClientRPC, it will not be performed on the host too.

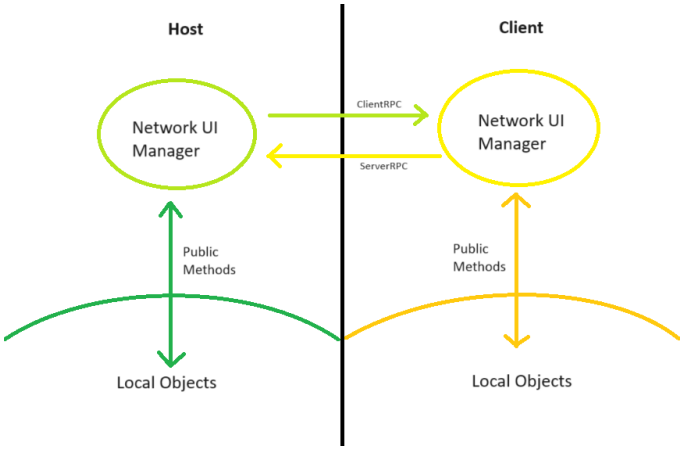


Figure A.7: A diagram showing how the communication between the two players works.

In figures A.8 and A.9, you can see an example of communication where the client's part is informing the host about a button being pressed. If you work on another project in the future that uses the most up-to-date version of Unity's Netcode for GameObjects, this syntax for RPCs is going to be deprecated. Newly, you need to declare that a function whose name ends in "RPC" is an RPC and state who it is for in the arguments. For example, if you wanted to create a ServerRPC, you would write, "[RPC(SendTo.Server)]",

instead of "[ServerRPC]". Furthermore, it is advised not to use functions ending in "ServerRPC" or "ClientRPC" for legacy reasons. As you can see, scripts on either side first call public methods of the Network UI Manager class which then call the RPCs, instead of calling the RPCs directly from other scripts. This is done for safety reasons and because often you would want something else to be done first on one side before sending it to the other, for example, updating local variables. In the figure A.9 you can see the correct syntax for declaring a Server RPC. The argument "RequireOwnership = false" lets any client invoke the method. The ButtonClickServerRPC is called on the client's side but performed on the host's side. First, it finds the correct gameObject. In this case, it is a WheelBox, which is the machine displaying the code as visible in the figure A.6. After checking if such an image was found in the game scene, it calls a "RotateWheel" method, passing it one argument. This method rotates the wheel of the given "buttonNumber" to the next letter.

```
1 reference
public void ButtonClick(int buttonNumber)
{
    //always called on client side
    ButtonClickServerRpc(buttonNumber);
}
```

Figure A.8: A public method in the "Task synchronisation calls" section to notify the host a certain button was clicked.

```
[ServerRpc(RequireOwnership = false)]
1 reference
private void ButtonClickServerRpc(int buttonNumber)
{
    WheelBox wbox = FindObjectOfType<WheelBox>();
    if (!wbox) return;

    wbox.RotateWheel(buttonNumber);
}
```

Figure A.9: A ServerRPC calls all needed methods on the WheelBox object on the host's side.

To ensure communication between the player and the game, you can either create UI elements, for example, buttons, the same way as in any other Unity game, or you can use Ray Casting. The InputManager.cs script I created contains a method that is called every time a player touches the screen. In the figure A.10 you can see the content of the file. OnEnable and OnDisable subscribe and unsubscribe the method FingerDown to the event of a player touching their screen. A single argument containing information about the touch is passed to the method. The first line checks if there are multiple touches at the same time and returns if there are, not letting the player click with multiple fingers at one time. Then it creates a Ray object passing through the point on screen transformed to the game coordinates and going in the direction the camera is facing. Next, the ray is cast, returning true

if it collides with any object and returning info about the closest object hit. After that, it tries to match the tag of the object hit and call a method meant to be called when clicked on, for example, if the ray hits an object with a "Button" tag, it calls the object's method that notifies the button box, which button the player clicked on. You can use any way of checking what kind of object was hit, but I prefer comparing the Tags. You then add lines to call the right method implemented in the given object's script. Make sure you add a collider to the objects you want the player to be able to click on, otherwise, the rays won't detect the object.

```
Unity Message | 0 references
private void OnEnable()
{
    EnhancedTouch.TouchSimulation.Enable();
    EnhancedTouch.EnhancedTouchSupport.Enable();
    EnhancedTouch.Touch.onFingerDown += FingerDown;
}

Unity Message | 0 references
private void OnDisable()
{
    EnhancedTouch.TouchSimulation.Disable();
    EnhancedTouch.EnhancedTouchSupport.Disable();
    EnhancedTouch.Touch.onFingerDown -= FingerDown;
}

2 references
private void FingerDown(EnhancedTouch.Finger finger)
{
    // disable multiple touch
    if (finger.index != 0) return;
    Ray ray = mainCamera.ScreenPointToRay(finger.screenPosition);
    RaycastHit hit;

    if (Physics.Raycast(ray, out hit))
    {
        if (hit.transform.CompareTag("TestCube"))
        {
            ChangeMaterial tcs = hit.transform.GetComponent<ChangeMaterial>();
            tcs.Change();
        }

        if (hit.transform.CompareTag("Button"))
        {
            ButtonBehavior button = hit.transform.GetComponent<ButtonBehavior>();
            button.ButtonClick();
        }

        if (hit.transform.CompareTag("PuzzlePiece"))
        {
            PuzzlePieceScript piece = hit.transform.GetComponent<PuzzlePieceScript>();
            piece.Clicked();
        }
    }

}
```

Figure A.10: Content of the InputManager.cs file controlling ray casting.

If you want to programme an interaction that interacts with image recognition, you will have to adapt the ImageRecognitionScript.cs. One part of

my game is when an image is being recognised by one player, it changes the material of an object for the other player. In the figure A.11, you can see my implementation of a function that is called every time the "AR Tracked Image Manager" component of AR Session Origin changes the status of the images being tracked. The method is passed a single argument of struct ARTrackedImagesChangedEventArgs, which contains three lists of type ARTrackedImage. The "added" list contains images that weren't being tracked before, but now they are. When an image is being tracked but disappears (or is too far from the camera to be actively recognised), it is not yet removed from the tracked images, only its status changes. These images, together with the ones that reappeared in the scene, are in the "updated" list. Lastly, the images that are removed from being actively tracked are in the "removed" list. For this reason, the method contains three loops, each looping through one of the lists. In the loop which loops through the "added" list, there is another loop that loops through all the client or host prefabs given to the script. Their names are compared and if they match the prefab is instantiated, their position being parented to the tracked image's position in the scene. Inside this method, you can perform any kind of task you need for your new implementation. For example, one part of my game requires the client to look closely at one of the images, which then leads to changing the material of an object for both the host and the client. This reveals the code for solving one of the tasks. You can see that when looping the "added" list, if the script's owner is the client and if the name of the tracked image matches "WatchBox", it changes a local boolean variable and calls a method of NetworkUIManager, which performs the material changes. A similar idea is in the other two loops when the image is not visible or becomes visible again.

A good practice is to store the progress of the game locally in both instances of NetworkUIManager. The players may finish a part of the game before the next part is instantiated. Then the players would be stuck. To fix this, each script can call a public method of the NetworkUIManager to acquire the needed information. In the figure A.12, you can see the PuzzleBoxScript.cs script. Specifically, a part of the Awake method that calls NetworkUIManager's (in this script called "nium") public method that returns a boolean value indicating whether the players have finished all previous parts of the game and whether the methods to enable them to start solving this one should be called. In practice, when the players solve the previous task, the puzzle box (object with the PuzzleBoxScript component) should open for one player and make a window in the lid for the other. If the object is not yet instantiated, this ensures that the box behaves as if the previous task was just solved when instantiated.

There is usually a supervisor in real-life escape rooms. If you would like to implement a way for a third party to monitor the progress of the game, the easiest way would be to create another UI button in the main menu, which would connect the user to the host as a client and also set internal variables to notify the app, that the user is just a supervisor and not a player. Then you would create a UI for supervising, showing which tasks have been solved.

```
2 references
private void ImagesChanged(ARTrackedImagesChangedEventArgs obj)
{
    foreach (ARTrackedImage image in obj.added)
    {
        string name = image.referenceImage.name;

        if (isHost)
        {
            foreach (GameObject prefab in hostPrefabs)
            {
                if (string.Compare(prefab.name, name, StringComparison.OrdinalIgnoreCase) == 0)
                {
                    GameObject newObject = Instantiate(prefab, image.transform);
                    placed[name] = newObject;
                }
            }
        } else
        {
            foreach (GameObject prefab in clientPrefabs)
            {
                if (string.Compare(prefab.name, name, StringComparison.OrdinalIgnoreCase) == 0)
                {
                    if (!watchBoxSeen && string.Compare(name, "WatchBox", StringComparison.OrdinalIgnoreCase) == 0) {
                        nuim.SeenChanger(true);
                        watchBoxSeen = true;
                    }
                    GameObject newObject = Instantiate(prefab, image.transform);
                    placed[name] = newObject;
                }
            }
        }
    }

    foreach (ARTrackedImage image in obj.updated)
    {

        if (!isHost && string.Compare(image.referenceImage.name, "WatchBox", StringComparison.OrdinalIgnoreCase) == 0)
        {
            bool isSeen = image.trackingState == TrackingState.Tracking;
            if (isSeen != watchBoxSeen) nuim.SeenChanger(isSeen);
            watchBoxSeen = isSeen;
        }
    }

    foreach (ARTrackedImage image in obj.removed)
    {
        if (!isHost && string.Compare(image.referenceImage.name, "WatchBox", StringComparison.OrdinalIgnoreCase) == 0)
        {
            if (watchBoxSeen) nuim.SeenChanger(false);
            watchBoxSeen = false;
        }
    }
}
}

```

Figure A.11: The ImagesChanged method inside the ImageRecognitionScript.cs, which is called every time the "AR Tracked Image Manager" changes the status of tracked images.

```
if (nium.IsPuzzle3())
{
    if (nium.Host())
    {
        movable = true;
        OpenBox();
    } else
    {
        MakeWindow();
        FillPieces();
    }
}
```

Figure A.12: A part of the Awake method of the PuzzleBoxScript.cs, shows how it gets the information about the game state from NetworkUIManager (usually abbreviated "nium" in my code).

To get the information, simply access the NetworkUIManagers local variables recording the game state.

To add a third unique player to the game, you can follow similar steps as when creating a supervisor. In the game logic, the players will not be differentiated only by one being the host and another being the client. Now there will be a host and two clients. You will have to update all methods in the NetworkUIManager to always call the correct RPCs, so it is called on the correct instances of the game. For example, it is no longer sufficient for one client to send just one ServerRPC, it will need to send a ClientRPC which excludes itself as well. Furthermore, in the ImageRecognitionScript.cs, you will have to create a new list of prefabs containing prefabs for the third player. You will have to add a third loop inside the ImagesChanged method's loop of added AR Images. This new loop will try to match the image's name to the prefab name if the third player is the owner of this script. Similarly, you can add any number of new players.

Currently, the PuzzleBoxScript.cs calls the NetworkUIManager's method GameWon, which shows the ending screen of the game. When you add new tasks, change the script, so it calls a different method of the NetworkUIManager to progress the game to the next task. Similarly, call the GameWon method from the script of the final task to conclude the game.

A.6 Debugging

If the game doesn't work as intended, I take a few general steps to locate the issue. Firstly, if I build the app and run it on my phone and the network communication does not work, I always check the console of the Unity project. Since currently the Log Level in Network Manager is set to Developer, it detects the phone, if it's in the same network, and prints all logs in the editor. Alternatively, there are other ways to connect directly to the phone's console. To build on that, you can create your debugging logs inside the scripts. If that doesn't help, I check if any part of the game is working, or if the whole network synchronisation is down. For example, if all the tasks in the game work and one has some issues, there is probably going to be some little mistake in the script. On the other hand, if suddenly nothing works, there might be a bigger problem.

Let's say I have created a script with local methods, for example, the code machine. It can rotate each wheel to the next letter, it can change material when the correct code is put in, notify the players, and more. It would take a lot of time to keep building the application and testing what doesn't work on my phone, so instead I can use the Invoke or InvokeRepeating methods in the Awake method. These methods call other methods without arguments in a given time, plus InvokeRepeating calls it repeatedly every number of seconds. After that, I can create a prefab in the scene and position it so that the default camera in the editor version of the game can see it. After starting the game in the editor, I can see if all the methods behave the way they should. In the figure A.13, you can see how I tested the WheelBox.cs script. In the Awake method, I call InvokeRepeating, which calls the method given in the first argument from repeatedly after several seconds given by

the second argument every number of seconds given by the third argument. For example, the first call “InvokeRepeating(“Inc1”, 0, 1);” calls the function Inc1() every second starting at 0 seconds from when the Awake method is called. Below the Awake method, I’ve implemented methods that call another method with specific arguments, since Invoke does not pass any arguments to the methods called.

```
Unity Message | 0 references
35  private void Awake()
36  {
37      nium = FindFirstObjectByType<NetworkUIManager>();
38      stepAngle = 360f / (float)MaxInt;
39      wheel1 = transform.Find("Wheel1");
40      wheel2 = transform.Find("Wheel2");
41      wheel3 = transform.Find("Wheel3");
42
43      // TEST
44
45      InvokeRepeating("Inc1", 0, 1);
46      InvokeRepeating("Inc2", 1, 2);
47      InvokeRepeating("Inc3", 0, 5);
48
49      // TEST
50  }
51
52  //TEST
    0 references
53  private void Inc1()
54  {
55      RotateWheel(1);
56  }
57
    0 references
58  private void Inc2()
59  {
60      RotateWheel(2);
61  }
62
    0 references
63  private void Inc3()
64  {
65      RotateWheel(3);
66  }
67  //TEST
68
```

Figure A.13: Usage of the Invoke method for debugging in the WheelBox.cs script.

I can also test if there is an issue with the network communication on the computer. ParrelSync lets me create an exact copy of a project so that it can be run in two separate editors. This lets me test if the communication works or if it’s broken. I could put some simple object in the scene that is changed in each instance of the editor and the change should be reflected in the other one as well. Alternatively, I can create a simple player prefab that can be controlled, put it in the network manager and see if the two editors synchronise correctly.

The same thing can then be done after building the game for the phone. If

it worked on the computer but not on the phone, there could be some issue with the phones, the way they treat the network, or the network treats them. There could also be an issue with the build. As I mentioned earlier, it is necessary to set stripping to minimal in the player settings, because otherwise, the engine will strip important parts of the Multiplayer Tools package and the app won't work on a phone.

Appendix B
Testing

I included the transcriptions of my introductions speech and for the testing below, followed by the questionnaire for the testers made in Google Forms. The speech is in English and Czech, because not all of the testers knew English enough to understand the speech properly. The questionnaire contains simpler English, so it was not translated.

B.1 English introduction speech

Thank you for participating in the testing of a cooperative Augmented Reality escape room game I developed for my bachelor's thesis.

Augmented Reality, AR for short, is a technology using a device's camera to synchronise the image from it and the game world making it seem virtual game objects are a part of the real world. Furthermore, my game uses Image recognition to place the game objects in the right place.

The game is intended for two players and each player may see different objects than their partner.

You will try to solve a part of this game with your partner. You will need to cooperate. It is apparent when the part of the game is finished and I will also be watching you as you play and intervene if something goes wrong or if you need help. The only game objects you have to interact with are the two in the place of these two images (I show the two images). Any other objects, as well as the joystick on the bottom left, are not a part of the testing and are only a debugging tool for convenience. (I give each test subject one phone with the game prepared beforehand and let them play).

B.2 Czech introduction speech

Děkuji, že se účastníte testování mé kooperativní únikové hry využívající rozšířenou realitu, kterou jsem vyvinul pro mou bakalářskou práci.

Rozšířená realita, krátce AR z anglického Augmented Reality, je technologie užívající kameru herního zařízení pro synchronizaci jejího obrazu a herního světa, díky čemuž vytváří iluzi propojení virtuálního a reálného světa. Dále má

hra využívá technologie rozpoznávání obrázků ve scéně (image recognition), díky které jsou herní objekty ukotveny na správném místě.

Hra je zamýšlena pro dva hráče a každý z nich může na své obrazovce vidět něco jiného.

Vy se s Vaším partnerem pokusíte vyřešit část této hry. Je nutné spolu kooperovat. Je evidentní, když je tato část hry vyřešena, a já budu na průběh dohlížet v případě problémů nebo když budete potřebovat pomoc. Jediné objekty, se kterými musíte interagovat, jsou ty dva, které jsou umístěny na místě těchto dvou obrázků (ukážu dva obrázky). Jakékoliv jiné objekty, jakožto i Joystick v levém dolním rohu, nejsou součástí testování a jsou jen pro zjednodušení testování. (Každému hráči předám jeden mobil s předpřipravenou hrou a nechám je hrát).

B.3 Testing questionnaire

The testing questionnaire was created in Google Forms. Then, I made it into a PDF seen below and printed it out to hand out to the testers. Unfortunately, although everything on my computer is set to English, I didn't manage to change the automatically generated text from Czech to English, but I don't think it is an issue since it only notifies the person how to fill out a form and all the testers knew Czech.

Testing - Escape Pierito

1. Please state how familiar you are with the following topics.

Označte jen jednu elipsu na každém řádku.

	Not familiar	Somewhat familiar	Decently familiar	Expert
Real life Escape Rooms				
Video Games				
Programming				
Augmented Reality				
Unity Game Engine				

2. Please state how much you agree with the following statements.

Označte jen jednu elipsu na každém řádku.

	Definitely agree	Slithly agree	Neutral	Slightly disagree	Definitely disagree
The game experience felt stable.	◯	◯	◯	◯	◯
Using AR in Escape Rooms can enrich the experience.	◯	◯	◯	◯	◯
The task was intuitive.	◯	◯	◯	◯	◯
The task presented a unique and enjoyable kind of interaction between players.	◯	◯	◯	◯	◯
AR makes the experience less enjoyable (having to play on a device...).	◯	◯	◯	◯	◯
I would want to visit a full-scale Escape Room using similar AR interactions.	◯	◯	◯	◯	◯

3. Is there anything you would like to add about using AR in Escape Rooms?

4. Is there anything you would like to add about the demo you've been presented?

5. Can you think of any specific Escape Room task that would only be possible using AR? (optional)

Appendix C

Digital appendix

Contents of the online and DVD appendix (all file names start with "F3-BP-Vatascin-Peter-priloha-"):

- drive.txt (online only) — a link for a Google Drive containing the same files as the DVD.
- game_build.zip (online split into two parts) — the final build of my game called "Escape Pierito.
- codes.zip — separate folder containing all the C# scripts I created for this thesis.
- models.zip — contains folders of each 3D model I created and their textures.
- images_for_recognition.pdf — a PDF with the images needed to play my game.
- game_video.mp4 (online split into four parts) — a video of a full playthrough.
- game_source.zip (DVD only) — the complete Unity project I created for this thesis.