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 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

1 Introduction 1

2 Analysis of existing applications 3

3 Specifications of my game 7
3.1 Gameplay 7
3.2 Technical side of the game 8

4 Development tools 9
4.1 Tools for augmented reality 9
4.2 Tools for network multiplayer . . 11
4.3 3D modelling tools 12

5 Game implementation 13
5.1 Basic AR scene 13
5.2 AR scene with ray casting 14
5.3 AR scene with multiplayer 15
5.4 Image recognition and tracking . 19
5.5 3D Modelling 24
  5.5.1 Button Box 24
  5.5.2 Wheel Box 24
  5.5.3 Code Box 24
  5.5.4 Watch Box 25
  5.5.5 Puzzle Box 26
  5.5.6 Plant 26
  5.5.7 Cube 27
  5.5.8 Using the models in Unity . . 27
5.6 First scenario 29
  5.6.1 Hierarchy of the scripts in the Unity project 29
  5.6.2 Implementation 30
5.7 Second scenario 32
  5.7.1 Implementation 33
5.8 Third scenario 34
  5.8.1 Implementation 35
5.9 Final application 35

6 Testing 41
6.1 Testing process 41
6.2 Testing results 42
  6.2.1 Questions on testers' background 42
  6.2.2 Questions on testers' experience with my game 43
  6.2.3 Open-ended questions on testers' opinions 44
6.3 Testing conclusion 44

7 Conclusion 45

Bibliography 47

A Manual 53
A.1 Introduction 53
A.2 Common Issues 54
A.3 Basic Information about my Unity project 55
A.4 Adding new AR objects 57
A.5 Creating scripts and network communication 59
A.6 Debugging 65

B Testing 69
B.1 English introduction speech 69
B.2 Czech introduction speech 69
B.3 Testing questionnaire 70

C Digital appendix 75

Figures

4.1 Table of supported features from the official AR Foundation website[9] (January 2024). 10
4.2 A table of tools enabling network multiplayer for Unity (2020)[19]. . . 12

5.1 A simple AR scene with a floating cube and dots representing found flat surfaces. 14
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. 15
5.3 Cubes being created on the detected plane by touching the screen. Their material changes when touching the screen as well. 16
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. 16
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. 17
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. 18
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". 19
5.8 Images I first used when trying out the image recognition technology. . 20
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]. 21
5.10 Images I first used for image recognition after reading articles about the topic. 22
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]. 22
5.12 The images that turned out the best when testing image recognition and tracking. 23
5.13 An image of the button box model created for my game. 24
5.14 The images of the wheel box. . . 25
5.15 Images of the code box. 25
5.16 Images of the watch box. 26
5.17 Images of the puzzle box model. 27
5.18 Images of the process of creating the plant model. 28
5.19 An image of the toy cube model taken in Blender. 28
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. . . . 31
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". 32
5.22 Images from the final version of the game of the first scenario from both the host's and the client's side. 33
5.23 Images from the final version of the game of the second scenario from both the host's and the client's side. 34
5.24 Code snippets from the implementation of the second scenario. 37

5.25 Images from the final version of the game of the second scenario from both the host's and the client's side. 38
5.26 Code snippets from the implementation of the third scenario from the "PuzzleBoxScript.cs". 39
5.27 UI parts of the game. Both are cropped from the original phone size, and they both contain my signature at the bottom. 39

A.1 Screenshot of all used packages (Unity -> Window -> Package Manager) with active versions. . . . 54
A.2 Screenshot of the hierarchy of objects in my Unity scene. 55
A.3 Contents of the Assets folder of my project. 57
A.4 Material settings in Unity. 58
A.5 Lists of prefabs for both players inside the ImageRecognitionScript.cs, as visible in the Unity editor. 58
A.6 An image of an asset I created for the game for inputting a number code. 59
A.7 A diagram showing how the communication between the two players works. 60
A.8 A public method in the Task synchronisation calls section to notify the host a certain button was clicked. 61
A.9 A ServerRPC calls all needed methods on the WheelBox object on the host's side. 61
A.10 Content of the InputManager.cs file controlling ray casting. 62
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. 64
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). 64
A.13 Usage of the Invoke method for debugging in the WheelBox.cs script. 66

Tables

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. 5

5.1 Results of grading the images from figure 5.9 using Google's The arcoreimg tool[38]. 21

6.1 Statistics of closed questions of the questionnaire with the most important features listed. All numbers are rounded to two decimal places. 42

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. Ji°í šára, 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.

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 recognition, 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.

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).

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.

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.

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