Expertise
and insight
for the future

Ngoc Nguyen

Developing a multiplayer AR game using AR Foundation and Unity

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

6 June 2020

Author Title Number of Pages Date	Ngoc Nguyen Developing a multiplayer AR game using AR Foundation and Unity 30 pages + 3 appendices 6 June 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Supervisor, Head of ICT Department

In early 2016, knowledge about the augmented reality of the world is limited. Later that year, the release of Pokemon Go and the popularity of it got undeniable attention, as well as the technology underneath. In this thesis, the main focus is to introduce the fundamentals for getting started with the promising and ever-growing technology, namely Augmented Reality. People usually got confused between augmented reality and its sibling, virtual reality, therefore, the differences will be explained as well.

An augmented reality mobile game is developed throughout this report to apply and prove the application of the technology. The game brings an immersive experience to the player designed to simulate playing a real board game in the virtual world, that is no other platforms can be achieved.

The implementation process is explained in chapter 5. However, this thesis aims for the game developers who are already familiar with the Unity game engine and know all the simple terminology in game development. Therefore, a step-by-step tutorial is not provided. Following the powerful and popular game engine, augmented reality development tools are used, including Apple ARKit, Android ARCore, and Unity AR Foundation. Besides, this project is built for Android phones using the Windows machine, building this app for iOS using a macOS machine can be different.

The outcome is the thesis is an augmented reality mobile game with some expectations that are not fulfilled. Challenges and future improvements are pointed out after all. Carrying out this final year project, the author gains a significant amount of grasp about new technologies, and more confident in game development.

Keywords	Augmented Reality, Mobile game, Unity, AR Foundation

Contents

Appendices

List of Abbreviations

VR	Virtual reality
AR	Augmented reality
MR	Mixed reality
PC	Personal computer
SDK	Software development kit
AI	Artificial intelligence
UI	User interface
API	Application program interface
OS	Operating system
AAA	In the video game industry, AAA (pronounced "triple A") or Triple-A is a classification term used for games with the highest development budgets and levels of promotion.
CPU	Central processing unit
GPU	Graphics processing unit
GPS	Global positioning system
QR code	Quick response code

1 Introduction

The game industry is a competitive business. Therefore, new ideas and new technologies are needed to create new games and attract more players. This thesis explores the new technology in making the video game, namely Augmented reality, and provides the process of implementation of an augmented reality mobile game.

This project is carried out using the popular game engine, Unity, with this intention, this thesis is suitable for the familiar developer with Unity game development. The practical implementation chapter presents the most important processes, not a step-by-step tutorial.

Following the Introduction, this thesis contains five more chapters. In chapter 2, a short description of the current state of the video game industry and the author's motivation. Chapter 3 provides knowledge of Augmented reality while chapter 4 is an introduction of software tools used for developing this project. Chapter 5 describes the process of implementation of the mobile game and final words are contained in the last chapter.

2 Background and Motivation

This section briefly describes the current state of the video game industry and following by the reasons of the author making this project.

2.1 Current State of the Game Industry

Decades ago, the first introduced computer was huge and heavy. Many companies specialize in computer hardware that was researched and developed to reduce the size and increase the performance of the computer. After that laptop was invented as well as smartphone, which was the size become smaller, but the performance was slower. While the technology used in PC, laptop, and smartphone continues growing, the gap between PC and smartphone's performance is closing.

As a result of rising smartphone's performance, a mobile phone can handle more heavy tasks included gaming. Therefore, more and more people want to play video games on their mobile phones. Furthermore, mobile phones are light and compact, so the user can play video games anywhere they want.

Making mobile games contains a lower risk than PC or console video games. An AAA PC game made by a large game company is available to release after 7 to 10 years of development. Unfortunately, the game company can not receive profit from an unfinished product. Besides that, the player's concern may not similar to 7 to 10 years ago. Many sizable game projects have been forced to cancel after 5 years of processing. Developing mobile games required less time, which leads to gaining profit faster.

However, the more companies making mobile games, the bigger threat of competition. The potential of the game mobile business was undeniable, that was the reason why many game companies have followed the trend. Besides, a larger amount of game mobile start-ups has been established. Therefore, standing out in the crowded business was not a simple challenge.

Luckily, VR and AR technology were invented, which is provided great benefits for many industries including gaming. While VR still struggles to obtain enough number of

consumers for expanding at the moment, AR promising a remarkable leap forward for the mobile game business. At the moment, AR is still a new technology, bringing a unique AR game to the market is achievable.

2.2 Motivation

The author's first-time playing computer games was through uncle's PC in 2001. After that, the biggest enthusiasm of the author became playing video games. Over the year until now, the author has been using many devices for playing video games, including PCs, consoles, handheld consoles and smartphones. However, PC and console games are remaining the author's favorites for a very long time even when mobile games have become a huge trend worldwide.

Surprisingly, the first experience with VR and AR technology changed the author's mind. These two new technologies opened a new era for the next level of interaction in playing the video game. VR made huge changes for user experience, it represents a completely believable virtual world, which can be different from the real world or the real world in other places. As impressive as VR's sibling, AR technology gained the author's attention to the mobile game. A combination of the real environment with virtual objects in real-time make user feels connected.

In the real world situation, many times the author needs more than a few board games to play with friends. Playing mobile games is a good solution, but not moving for a long time is not a good idea. Therefore, A multiplayer AR game that simulates a board game will be a good idea, players can move around and enjoy their board games without any physical pieces of equipment.

3 Augmented Reality Technology

AR's definition will be introduced in this section, as well as how it works and which ways it can be used. There will be a short comparison between Augmented reality and Virtual reality, which is many people confuse about the similarity.

3.1 Augmented Reality's Definition

In AR, the physical space around us which is captured by image sensors in real-time is layered on with computer-generated graphics. This is usually associated with the augmentation of 3D objects to a live video from the camera of the mobile device, such as a smartphone or tablet [1, p. 8]. More details about how it works on the mobile platform will be discussed later.

AR is an old concept. The term has been accepted and used since the 1990s in research labs, military and other industries. Since then, the development kits for both open source and other platforms have been accessible. The attraction of AR of the consumer has gone further because of the rising of smartphones and tablets [1, p. 8]. Apple with ARKit and Google with ARCore released since late 2017, they constantly evolve and improve their SDK, even now in 2020, statement of Jonathan and Kristian [1, p. 8] about handheld AR has not yet achieved their ultimate form remain correct.

For further precisely about AR's definition, Jonathan and Kristan [1, p. 10] mentioned the definition of two separated words augmented and reality written in the Merriam-Webster dictionary, the word augment's meaning is " to make greater, more numerous, larger, or more intense" [1, p. 10]. And the definition of the word reality is " the quality or state of being real" [1, p. 10]. Then Jonathan and Kristan [1, p. 10] emphasized that "augmented reality, at its core, is about taking what is real and making it greater, more intense, and more useful". Beside word-by-word meaning, the goal of the AR is to enhance human sense in various activities, including directed tasks, learning, communicating, or entertaining.

Underneath the impression of AR is the power of artificial intelligence (AI) in the field of computer vision. AR needs computer vision to observe targets in the user's field of view,

including coded markers, natural feature tracking (NFT), or other methods to observe objects or text. When a target is detected and its position and orientation in the real world are marked, computer graphics that lines up with those real-word transform will be generated on top of real-world imagery [1, p. 10].

Another equally important definition of AR is it is running in real-time, not pre-recorded. For example, the mixing of actual action with computer graphics as cinematic special effects is not accepted as AR. Besides, the computer-generated display must be registered to the real 3D world, not 2D overlays. As an illustration, many head-up displays, such as in Iron Man's mask or even Google Glass, are not AR. From the user's point of view, AR graphics could be seen as real objects landing around them physically in the real-world [1, p. 11].

To summarize, AR has three key concepts, which is the virtual blend in the real, interact in real-time, and register in 3D.[1, p. 11]

3.2 Augmented Reality versus Virtual Reality

Many people usually got confused between virtual reality (VR) and AR. As described earlier, AR adds virtual objects to the real-world, on the other hand, VR puts you in a different environment digitally, which is virtual [1, p. 12].

One of the important purposes of VR headsets is to separate the user's visual to their real-world. Therefore, everything viewed in VR application is designed and created by the developer to maintain the fully immersive VR experience. Unfortunately, VR comes with a unique issue, which is motion sickness. Researches were conducted to point out that rendering the graphic at least 90 frames per second or higher will reduce or even remove the effect of motion sickness while using the VR headset [1, p. 13].

Unlike VR, latency is not a big issue for AR because of the real-world visual cover most of the user's field of view. As a result, users have a small chance to suffer the motion sickness problem [1, p. 13].

Both technologies have different heavy work-load on hardware usage. VR forces the device's CPU and GPU processors maximum power to render 3D graphics for both left and right eyes simultaneously for the entire scene while maintaining very high framerate, as well as running physics, animations, audio, and other processing tasks. On the other hand, AR's main task is to detect and track the targets through image processing pattern recognition in real-time. Depth sensors built-in complex devices are responsible for creating and tracking a scanned 3D model of the real-world in real-time (Simultaneous Localization and Mapping, or SLAM) [1, p. 13].

3.3 Handheld Mobile Augmented Reality

As described earlier, AR is a combination of the real-world and virtual graphic objects, those objects are attached to the physical 3D world, and this operation must be conducted in real-time. There are two popular ways to achieve that, which is using a handheld mobile device such as a smartphone, tablet, or using wearable AR smartglasses. In this thesis, the first technique will be focused, which is the most universal and reachable one. [1, p. 14]

In the first place, handle mobile platform contains unique characteristics to become the most common in the implementation of AR, those are:

- cordless and powered by a battery
- touchscreen display built-in
- cameras
- CPU, GPU, and memory
- motion sensors and gyroscope
- GPS and Wi-Fi.

First of all, mobile devices are cordless, which provide users the freedom to use anywhere without connection to a PC, and a built-in battery for supply power to all components. That is ideal for AR because AR operates in the real world, while users moving around. [1, p. 15]

Second, mobile devices have a color display with resolution and pixel density suitable for handheld viewing distance. Besides, a multitouch input sensor under the screen is an important feature for interaction using fingers. [1, p. 16]

Third, taking real-world video and present it on-screen in real-time required a rear-facing camera. AR application can process video data in real-time because video data is digital. [1, p. 16]

In addition, CPU and GPU included in mobile devices nowadays are powerful, which is crucial for AR to deal with heavy tasks such as detects targets in the video in real-time, handles sensors and user inputs, and process the video combination. Achieving higher performance requires further researches from hardware manufactures. [1, p. 16]

Furthermore, the advance of mobile AR depends on built-in sensors that calibrate motion, orientation, and other conditions. Tracking linear motion along three axes is an accelerometer's responsible, and tracking rotational motion around the three axes is a gyroscope's task. AR application can measure the device's position and orientation in the physical 3D environment based on real-time data from the sensors, then register the 3D graphics accordingly. [1, p. 16]

Last but not least, the user's location on the globe is defined by a GPS sensor. Metadata can be accessed from the internet with Wi-FI connection. [1, p. 16]

3.4 Mobile Augmented Reality's Operation

In a handheld mobile device, the AR application needs the device's camera to take the video of the real world then merge the graphic-generated objects into it. [1, p. 14] Handheld mobile video see-through is depicted in Figure 1.

Figure 1. Handheld mobile AR [1, p. 15]

As illustrated in Figure 1, using the mobile AR app, the user aims the camera at the target in the real world and a 3D virtual graphic will be registered to the target's position and orientation once the app detects the target. Then the AR video is showed on the device's screen, which is the meaning of video see-through. [1, p. 15]

The loop of detecting a target in AR is portrayed in Figure 2.

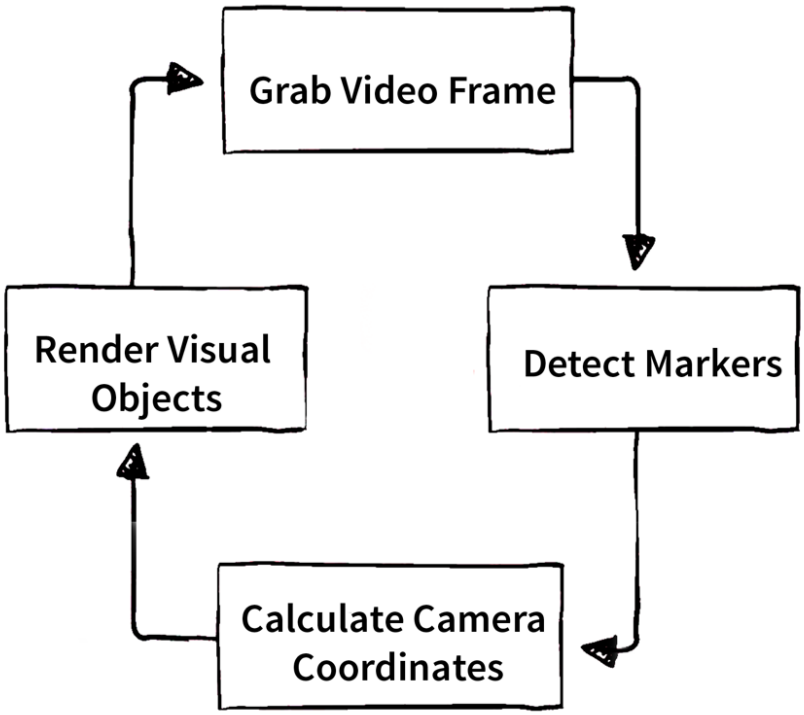

Figure 2. Target detection [1, p. 18]

In Figure 2, each frame of the video, which is taken by the device camera, will be analyzed by the application and use the photogrammetry technique to find a target, such as a pre-programmed marker. The distance, position, and orientation of the marker relative to the camera in 3D is inspected in the detection phase. Then, the virtual objects are rendered based on the value of the camera pose, namely position and orientation, which is established in the 3D world in the previous phase. Finally, the video frame showed to the user is combined with the rendered graphics. [1, p. 18]

Smartphone's display commonly has a 60Hz refresh rate. That is a heavy task for the smartphone to handle that process because the image on the screen is updated 60 times per second. Therefore, many attempts have been carried out in optimizing the software to minimize any unnecessary calculations, reduce redundancy, and other methods to boost efficiency while maintaining a smooth user experience. As an illustration, instead of detecting repeatedly the same target each time, the application will mark and follow its movement from one frame to the next once it has been spotted. [1, p. 19]

Interaction is the mobile screen with computer-generated objects is similar to any mobile app or game. Figure 3 shows the process of input in the mobile AR app.

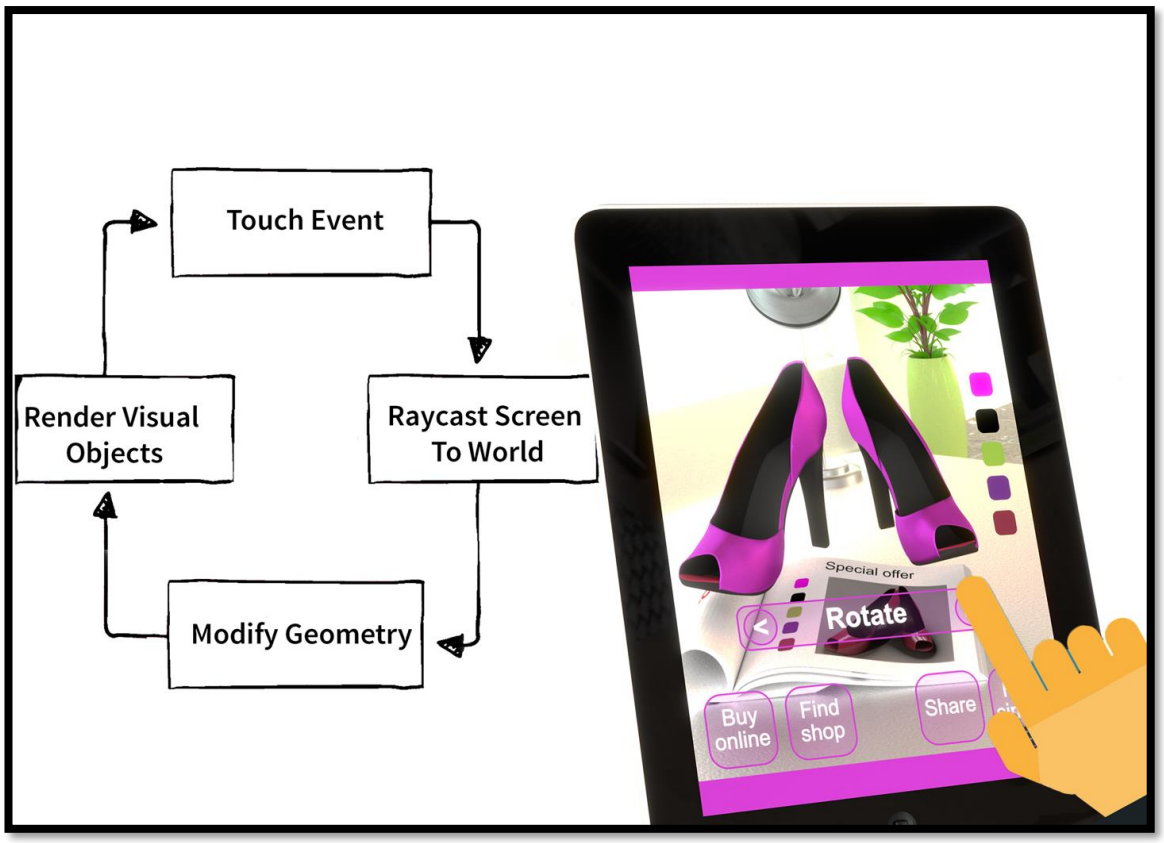

Figure 3. Input process in mobile AR app [1, p. 19]

As depicted in Figure 3, the AR app listens to a touch event. Then it casts a ray from the screen 2D position into 3D space, which uses the current camera pose, to verify which object user expected to tap. The app can react to the tap to move or modify the geometry in case the ray hits a detectable object. These modifications of the object will be rendered visually on the screen in the next updated frame. [1, p. 19]

Applying AR with the selfie camera of the smartphone is more advance, such as Snapchat popular app. The app runs complex AI pattern matching algorithms while analyzes the user's face to determine notable points or nodes, that match the features in the user's face, including eyes, nose, lips, chin, and so on. Then it creates a 3D mesh, which simulates the user's face, to cover the user's matching facial feature with an additional graphic layer and even modify and distort the user's face for entertaining. [1, p. 20]

3.5 Augmented Reality's Applications

As discussed in previous sections about what technique using underneath AR and how it works, AR has colossal potential to become popular as the World Wide Web, the social web. Below are a few examples of the applications that will gain benefit from AR technology.

Business marketing is the first example. A companion app can bring to customers more details about the product, or marketing media when it scans the AR markers printed in the product container. For example, AR business cards are an impressive approach to show information about a person. QR codes are a convenient way to take people to a website for advertisement purposes. Likewise, AR markers in advertising will be popular sooner or later. [1, p. 35]

Education is a field that already receives benefits from AR. Stories in children's books have been augmented to life for years. AR in educational textbooks and media resources are available for older students who study more serious subjects. [1, p. 35]

Industrial training's adoption of AR is another great example. An AR app explains how to fix equipment to increase technical training and lower mistakes. Paper training manuals, PDFs, or web pages are out-dated methods. Instructional videos are somewhat better. However, more interactive 3D graphics, as well as hands-on tutorials augmented on real-world objects brings training to be more intimate. [1, p. 35]

Retail earn a great boost from AR technology. The most compelling evidence is furniture stores, such as IKEA, can provide an AR app to the user for trying new furniture in their home visually before buying decisions. [1, p. 35]

Gaming got undeniable benefits from AR as well. Niantic made AR become a well-known term in the gaming industry with their Pokémon GO game. [1, p. 36] This will be the focus field of AR's application in this thesis.

Many other fields benefit from AR, including engineering, design disciplines, music, cinema, storytelling, journalism, and so on. [1, p. 36]

4 Development Tools

Various game development tools have introduced over the years, from the free version to the high price-tag game engine. Many of them support multi-platform, while some are aiming at a specific platform. Most of the popular game engine build games with heavily scripting, but a small amount of game engine is code-free for fast prototyping and suitable for people who do not have sufficient programming skillset. This chapter provides a brief introduction to one of the most well-known game engines, Unity. Following by the extra introductions of powerful AR development tools, namely ARCore, ARKit, and AR Foundation.

Unity is an industry-level game engine used for multi-platform video game creation. It is suitable for professional game development, as well as beginner game developers. Many tutorials on the internet and books are available to explain in detail about how to start using Unity. For downloading the latest version of Unity and watching tutorials, see https://unity.com/. [2]

Unity consists of a great number of modules for controlling and rendering 3D objects, lighting, physics, animation, audio, and the list goes on. The beauty of the Unity editor is powerful in developing and testing quickly and efficiently. Unity can be used in macOS and Windows. In terms of deployment targets, Unity can deploy to PC, web, mobile that is included iOS and Android, or consoles, and VR and AR platform support recently.[2]

ARCore is a platform made by Google for creating AR apps for Android devices. It uses three key abilities to combine virtual objects with the physical world through the camera:

- Motion tracking allows the phone to track its position based on analyzing the real world.
- Environment understanding allows the phone to discover any surface's size and location, such as horizontal, vertical, and angled surfaces.
- Light estimation allows the phone to evaluate the current lightning status of the surrounding physical space.

A list of supported devices can be found here https://developers.google.com/ar/discover/supported-devices.

ARKit is the framework made by Apple for building AR projects for the iPhone and iPad. Compiling an app on iOS or iPadOS required a macOS device. ARKit consists of features: [3]

- TrueDeph Camera
- Visual Inertial Odometry
- Scene Understanding
- Lighting Estimation
- Rendering Optimizations.

AR Foundation is a framework made by Unity, that allows developing an AR once, then deploy it across multiple mobile platforms and wearable AR devices, such as Android, iOS / iPadOS, Magic Leap, and HoloLens. This package provides an interface for Unity developer to use core features from each platform, as well as unique features from Unity, including photorealistic rendering, physics, device optimization. However, it does not carry out any AR features itself. [4]

5 Practical Implementation

This chapter explains in detail how the final year project is carried out. It also provides an overview of the project, such as the main idea of the project, the requirements to run the project in Unity.

5.1 Project overview

Many time the author visit friends and wanted to play board games, playing frequently a few board games over time leading to losing interest. Unfortunately, the author could not bring a large number of board games for having more options. Therefore, the author wanted to make AR mobile board games for players can switch to other games easily, at the same time player have to move, that encourages players not to sit for too long. On the other hand, an AR game will bring an immersive experience to the player and even more impressive than using a physical board game.

According to the purpose above, this project is based on the idea of creating a multiplayer AR mobile board game, which is an upgrade from the legacy Pong game. The game can run on both iOS and Android devices.

Up to 4 players can enjoy the game together at once or playing with computer enemies. The player controls their paddle to protect the goal and leads the ball to the enemy's goals. Several power-ups respawn randomly in-game, who take it first can use that to take advantage. Every player has 3 stars, each time their goal got hit, they lose 1 star. If a player loses all their 3 stars, they are knocked out of the game. The last player stays, that is the winner.

This project can deploy an app for Android and iOS devices. Building the app for Android devices can carry out either on Windows or macOS machine. However, building the app for iOS devices requires Xcode, which only can run on the macOS machine.

To run this project in Unity, suitable Unity version and extra preview packages are required to prevent any conflict and able to develop AR applications:

- Unity 2019.3.7f1 (later is possible)
- AR Foundation 3.1.0 preview 6
- AR Subsystems 3.1.0 preview 6
- ARCore XR Plugin 3.1.0 preview 6
- ARKit XR Plugin 3.1.0 preview 6.

5.2 Getting started

Before start writing any line of code, setting up all the software and tools needed are important. This project is carried out with Unity in Windows and developed an AR game for Android specifically. Due to the slight difference between developing for iOS and Android, this report may not suitable for iOS development.

First, Unity is essential for the entire process. To install Unity, downloading and installing **Unity Hub** first is strongly recommended, which is a developer can manage multiple Unity versions easily and add more modules conveniently. Choosing a suitable Unity version is crucial, as mention in the previous subsection. Depend on which mobile platform developer aims to, select **Android Build Support** or **iOS Build Support** in the process of installing Unity, or can be added later in Unity Hub.

Once a new project is created, heading to **Package Manager** to install the rest of the required packages, that is mentioned in the previous subsection. There are several configurations needed to be set in the **Build Settings** window. Begin with switching the platform to Android. The rest of the configurations is shown in Table 1.

Table 1. Player Settings configurations

Setting	Value
Player Settings > Other Settings > Rendering	Uncheck **Auto Graphics API** If **Vulkan** is listed under **Graphics APIs**, remove it.
Player Settings > Other Settings > Package Name	Create a unique app ID using a Java package name format. For example, use `com.example.helloAR`
Player Settings > Other Settings > Minimum API Level	Android 7.0 'Nougat' (API Level 24) or higher (For AR Optional apps, the Minimum API level is 14.)
Player Settings > XR Settings > ARCore Supported	Enable

As shown in Table 1, those configurations are in **Player Settings**. Preparing all the settings carefully is important to avoid any extra conflict in further development.

Last but not least, installing the **Android Logcat** package for debugging and analyzing the device is recommended. Enabling **Developer Options** and **USB debugging** on the testing device is necessary as well.

5.3 Initial scene set up

After importing all the required packages in the previous subsection, the next step is creating AR Foundations game objects under the **GameObject > XR** submenu. The initial set up scene of **Hierarchy** and **Project** view can illustrate in Figure 4.

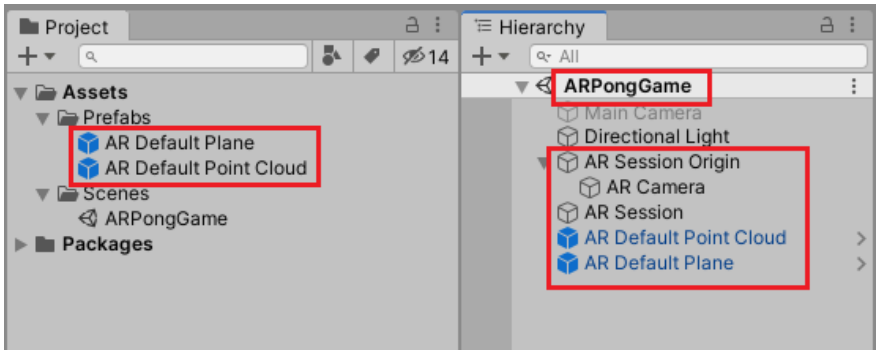

Figure 4. Initial set up of Project and Hierarchy view

As can be seen in Figure 4, the scene's name is changed to **ARPongGame**. Four new game objects are created and shown in the Hierarchy view. The **AR Session** controls the lifecycle of an AR app. The **AR Session Origin**'s mission is to transform trackable features into their final position, orientation, and scale in the Unity Scene. Two last game objects are blue, which indicates they are prefab instances, by dragging into newly created folder **Prefabs** in Project view, then they will be deleted after that.

Several components and prefabs need to add to the **AR Session Origin**, which is shown in Figure 5.

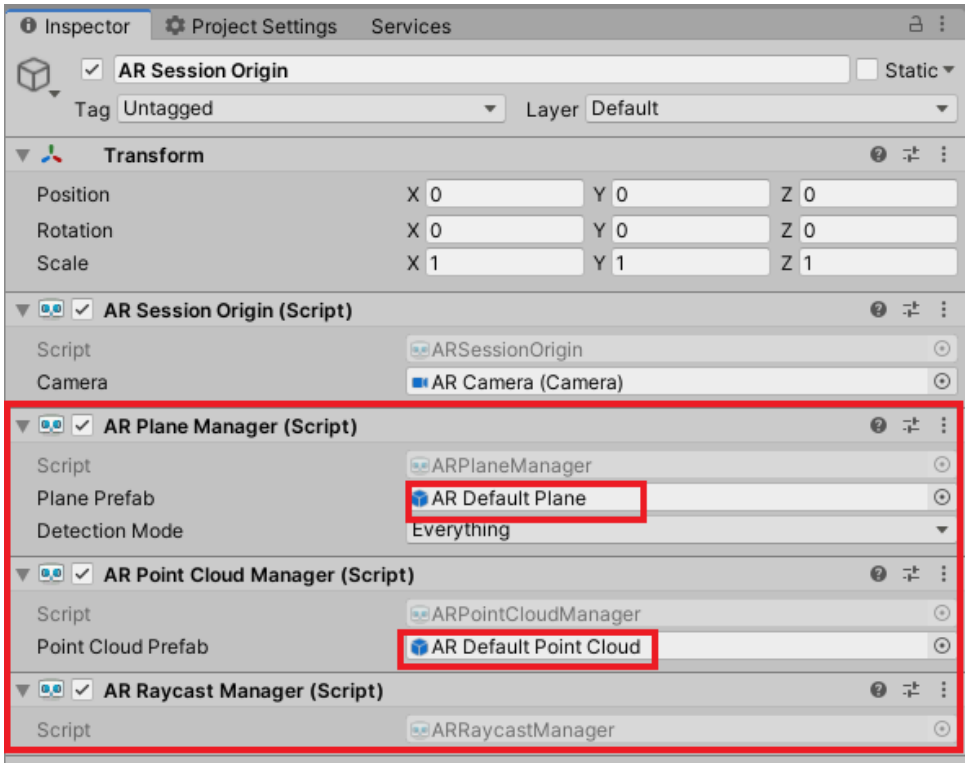

Figure 5. Components added to the AR Session Origin

In Figure 5, three new components are added, namely **AR Plane Manager**, **AR Point Cloud Manager**, and **AR Raycast Manager**. Their purpose is to detect flat surface, detect feature points, and raycast functions respectively. Two prefabs that are created in the previous step are added to new components.

The final step in setting up the initial scene is adding the opening scene to the **Scene in Build** list in **Build Settings** window.

5.4 Board placing

The game board, which contains all the objects of the game, will be selected its position by the player from the beginning of the game. The script handling this function is added to the AR Session Origin game object, therefore it will be executed early when the player opens the app.

Once the app is started, all the AR game objects related, that is mentioned earlier, worked together to analyze the video feed of the real-world, detect feature points then create horizontal or vertical planes. The Update() method for placing a board is shown in Listing 1:

```
// Update is called once per frame
    void Update()
    {
        if (!boardIsPlaced)
        {
            if (RaycastManager.Raycast(centerScreenPos, listHits,
                TrackableType.PlaneWithinPolygon))
            {
                // Raycast hits are sorted by distance, so the first one
                // will be the closest hit.
                hitPose = listHits[0].pose;

                if (spawnedObject == null)
                {
                    spawnedObject = Instantiate(board);
                    SessionOrigin.MakeContentAppearAt(spawnedObject.transform,
                    hitPose.position, hitPose.rotation);
                }
                else
                {
                    SessionOrigin.MakeContentAppearAt(spawnedObject.transform,
                    hitPose.position);
                }
            }
        }
    }
```

Listing 1. Placing the board

In Listing 1, that is the **Update()** method, which is called once per frame. Whether or not the board is placed on the plane will be checked every frame, if not then a ray will be cast from the center of the phone screen and target and available plane in the 3D world. Once the **Raycast** sends out by the **RaycastManger** hits a plane, an image of the board will appear and the player can rotate the phone and move around to select the preferred

position for the board. There are more methods and lines of code contribute to placing the board, the full script can be found in Appendix 1.

5.5 Play Area prefab

The game needs to restart several times while players playing. Usually, using restart the scene method available with Unity is easy and convenient. However, an AR app renders virtual objects on top of the video feed from the phone's camera, using that restart method will black out the screen, which can disrupt the immersive experience of the user. Therefore all components of the game need to be packed in one prefab and reset the prefab without resetting the camera feed. All of the prefab's components and visual is illustrated in Figure 6:

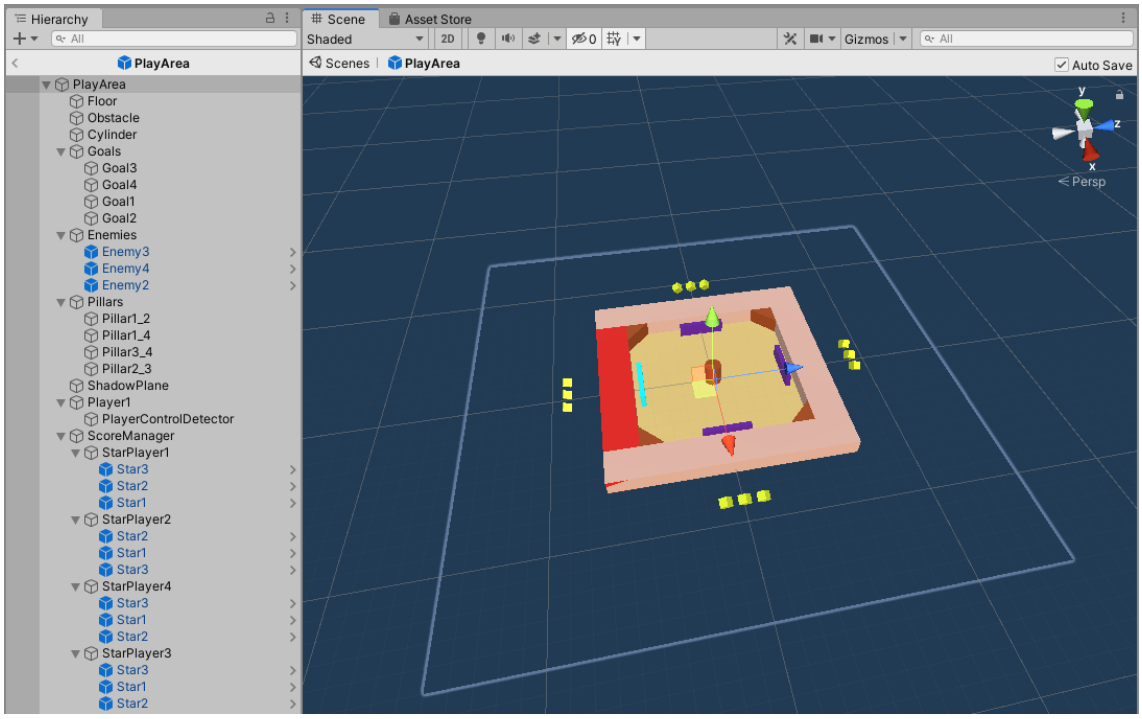

Figure 6. Play Area prefab

As shown in Figure 6, the right side is the **Scene** view, which is displaying the whole play area prefab visually. That is how the play area looks like in the game. On the left side is the **Hierarchy** view, which is a nested list of all the objects. Many objects on the list contain collider for all the physic interaction related. Objects, such as **Goals**, **Enemies**,

Player, and **ScoreManager** include separated script components, which will be explained later in this chapter.

The **ShadowPlane** is a transparent material plane with a custom shader, that allows the transparent plane to receive shadow. Therefore all objects in the **PlayeArea** will be got virtual shadow added to the video feed. Moreover, one script component is attached to the **PlayArea**, that is shown in Listing 2:

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

public class PlayArea : IntEventInvoker
{
    // Start is called before the first frame update
    void Start()
    {
        Time.timeScale = 0;

        // add as invoker for BoardPlacedEvent
        unityEvents.Add(EventName.BoardPlacedEvent, new BoardPlacedEvent());
        EventManager.AddInvoker(EventName.BoardPlacedEvent, this);

        unityEvents[EventName.BoardPlacedEvent].Invoke(0);
    }

    private void OnDestroy()
    {
        gameObject.tag = "Untagged";
        EventManager.RemoveInvoker(EventName.BoardPlacedEvent, this);
    }
}
```

Listing 2. Script component of the PlayArea

In Listing 2, the **Start()** method is called before the first frame update. Its purpose is to pause the game and invoke an event to indicate that the play area is fully loaded and ready to run. The invoke callback belongs to a complex Unity Events system, which will be discussed later.

5.6 The ball

One of the key objects in this game is the ball. The ball needs to convert to a prefab object because that is an efficient technique to respawn a new ball while the game

running. Physical interaction is crucial for the ball in the game, therefore, several settings of components of the ball must setted properly in the **Inspector** as shown in Figure 7:

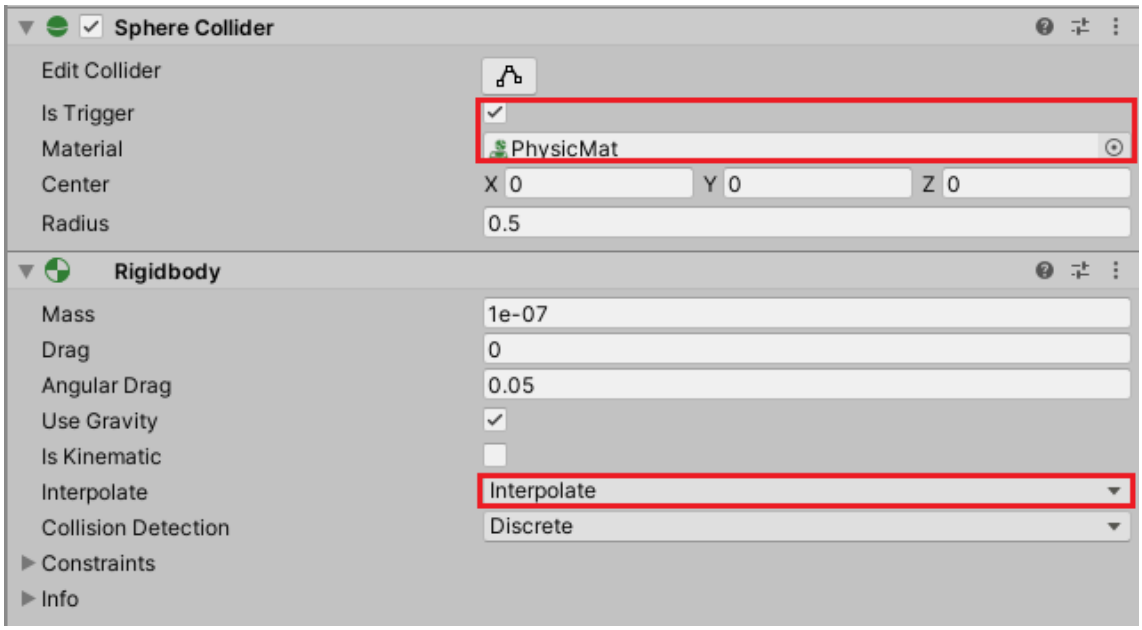

Figure 7. Important components of the ball prefab

As depicted in Figure 7, in **Sphere Collider** component, **Is Trigger** is enabled, and **Material** property contains the **PhysicMat** physic material, which is a custom physic material with the **Bounciness** value set to 1. In the **Rigidbody** component, the **Interpolate** is set to **Interpolate** for the smooth transition of the moving ball when the game enters slow-motion time.

Controlling the movement of the ball is handled by the attached script. Two essential methods are shown in Listing 3:

```
// Update is called once per frame
    void Update()
    {
        if(Time.time >= timeToMove && !ballmoves)
        {
            Move();
        }

        //For constant velocity
        if (ballmoves)
        {
            //Get current speed and direction
            Vector3 direction = rb.velocity;
```

```
            float currentSpeed = direction.magnitude;
            direction.Normalize();

            if (currentSpeed != speed)
            {
                rb.velocity = direction * speed;
            }
        }
    }

    void Move()
    {
        //Determines a random starting direction
        transform.position = new Vector3(0, 0.5f, 0);
        randomDir.x = Random.Range(-1F, 1F);
        randomDir.z = Random.Range(-1F, 1F);
        //randomDir.x = 1;
        randomDir.Normalize();
        rb.velocity = new Vector3(randomDir.x, randomDir.y, randomDir.z) *
speed;

        ballmoves = true;
    }
```

Listing 3. Control ball movement

As illustrated in Listing 3, once per frame, the **Update()** method is called to ensure the suitable condition to allow the ball to start moving. One the other hand, another task of it is maintaining the constant velocity of the ball in any condition, because, at the time of the collision with other physic objects, the velocity of the ball may become unstable. The **Move()** method is responsible for determining a random starting direction for the ball.

5.7 Paddle control

To play the game, every player control one paddle to block the ball from going through the goal. Simply dragging the paddle parallel with the goal. Because this game is an AR game, touching and dragging the in-game object are not similar to a normal mobile game. An important piece of code of the Update() method is shown in Listing 4:

```
if (!TryGetTouchPosition(out Vector2 touchPosition))
        {
            drag = false;
        }
        else
        {
            Ray ray = Camera.main.ScreenPointToRay(touchPosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit))
            {
```

```
                if (hit.collider.name.Equals("PlayerControlDetector"))
                {
                    drag = true;

                    if (drag == true)
                    {
                        float screenDist = Camera.main.WorldToScreen-
                        Point(gameObject.transform.position).z;
                        Vector3 pos_move = Camera.main.ScreenToWorldPoint(new
                        Vector3(touchPosition.x, touchPosition.y, screenDist));
                        transform.position = new Vector3(pos_move.x, trans-
                        form.position.y, transform.position.z);
                    }
                }
            }
        }
```

Listing 4. Control the paddle

As shown in Listing 4, when a touch action is detected, a ray is cast from the touchpoint forward in the 3D world. At the same time, hit testing is established to identify which object got hit by the ray. If the object's name got hit is **PlayerControlDetector**, which is a child object of the paddle, then the player can start dragging the paddle.

5.8 Enemy

At the idea stage, the game is designed to be a multiplayer game. However, due to short of time, computer enemies are created to cover the missing of real players. Begin with creating the enemy paddle prefab for duplicating more enemies faster.

As similar to any object that needs physical interaction, add the **Collider** and **Rigidbody** components to the enemy paddle object. In the **Rigidbody** component, select the **Interpolate** option. **Constraints** field selection slightly differences between enemies paddle based on its position on the board. An example of those selections is illustrated in Figure 8 below:

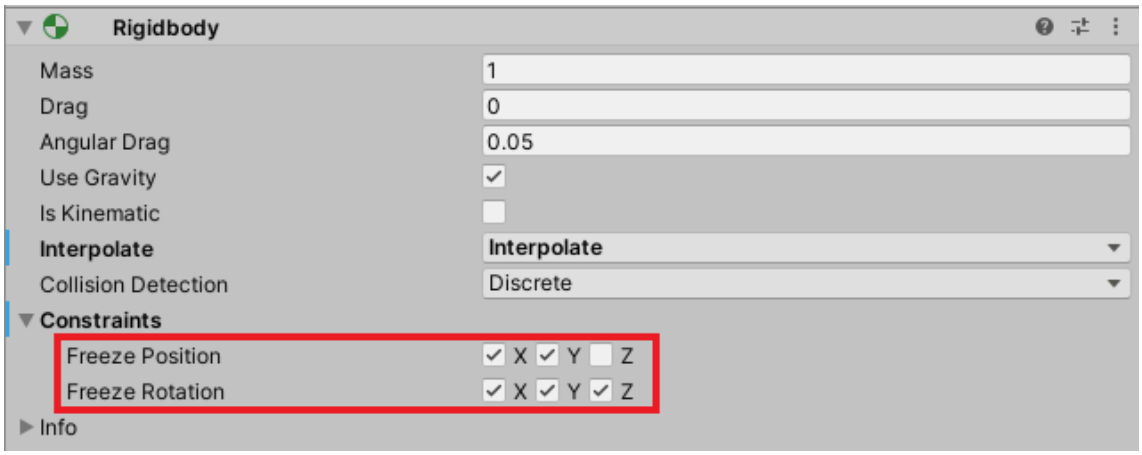

Figure 8. Rigidbody component of enemy paddle

As shown in Figure 8, all the **Freeze Rotation** of 3 axes are selected, this prevents the paddle to rotate in any direction and applied to all enemies paddle object. On the other hand, only the enemy, which its position is opposite to the main player, is **Freeze Position** on **X**-axis, the rest enemies paddle maintain the same selection as shown in Figure 8 above.

Although computer enemies are not implemented with Artificial Intelligent (AI), they are still designed to adjust the difficulty level easily if needed. One parent object control all the enemies paddle, the controlling method is similar for all the enemies, which is shown in Listing 5:

```
//Reading the ball's coordinates
        if (ball != null)
        {
            ballCoord = ball.transform.position;
            i = 0;

            //Tracking the ball and moving enemies
            if (move && ball.GetComponent<Ball>().ballmoves)
            {
                //Enemy 1
                if (!knockedPlayers[3] && !isFreeze3)
                {
                    if (enemy3.transform.position.x <= ballCoord.x && en-
                        emy3.transform.position.x <= 2.5F)
                    {
                        enemy3.transform.position = new Vector3(
                            enemy3.transform.position.x + movAmount,
                            enemy3.transform.position.y,
                            enemy3.transform.position.z);
                    }
```

```
        if (enemy3.transform.position.x >= ballCoord.x && en-
            emy3.transform.position.x >= -2.5F)
        {
            enemy3.transform.position = new Vector3(
                enemy3.transform.position.x - movAmount,
                enemy3.transform.position.y,
                enemy3.transform.position.z);
        }
    }
```

Listing 5. Control the enemy paddle

In Listing 5, this is a part of the Update() method in the script attached to the enemy controller object. Every frame, the ball's coordinate is tracked and the paddle is moved based on that data. Several booleans are added to prevent the null reference error. The full script to control enemies' movement can be found in Appendix 2.

5.9 User Interface

User Interface (UI) is important for the user to interact easily with the game. Creating any new UI object leads to new Canvas created if there is not one. An overview of UI for this project is depicted in Figure 9:

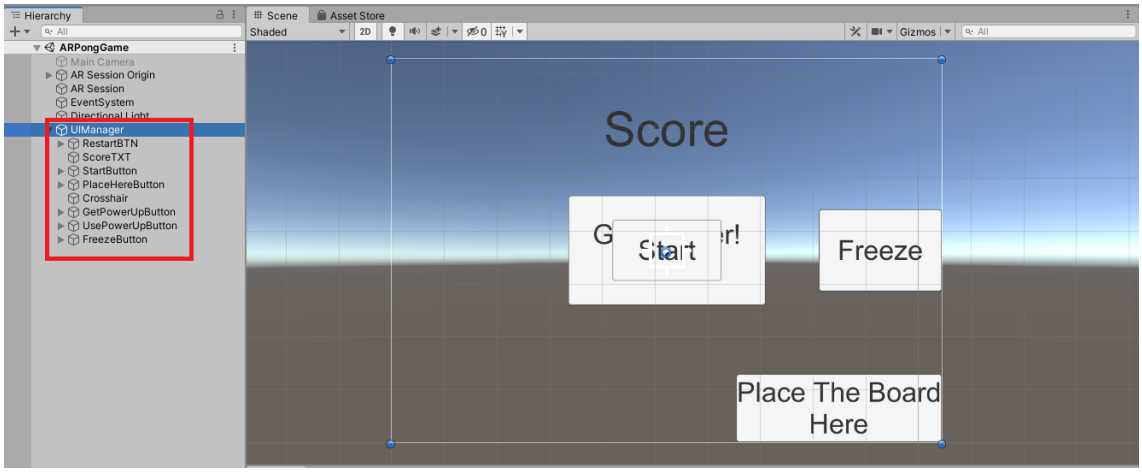

Figure 9. User Interface

As shown in Figure 9, on the **Hierarchy** view, the main Canvas is renamed to **UIManager**, including many UI objects as children. Most of the UI objects are buttons, else are a score text and a crosshair. As can be seen on the **Scene** view, that is the layout of the

UI. All UI elements are set its position and pivot based on the side of the screen, therefore, its position will adjust automatically in any screen sizes. The **UIManager** contains a script for controlling all the UI elements. Methods in the script are mainly provided Onclick behavior for all the buttons and several conditions to enable and disable buttons.

5.10 Gameplay

The Gameplay script is attached to the **AR Camera** game object, so that script can be executed correctly after the AR Camera is ready to run. Most of the tasks related to game operation are handled by this script, such as responding to event invocations, including board placed successfully, start the game, respawn a new ball, and so on, as well as invoke several events. The detailed script is written in Appendix 3.

Included in the gameplay mechanic are the goal and the scoring system. The goal script attached to each goal object is responsible for detecting any collision of the ball, then invoke an event to inform other components to react accordingly. Besides, the goal also can a goal into a normal wall when the player at that position is knocked out.

Last but not least, the score manager handles all score related activities, namely changing the score of all players and deciding when the game is over. The method to manage the score is shown in Listing 6:

```
void HandleGoalEvent(int goalOfPlayerGotHit)
    {
        scorePlayers[goalOfPlayerGotHit] -= 1;

        if (scorePlayers[goalOfPlayerGotHit] == 0)
        {
            Destroy(StarPlayers[goalOfPlayerGotHit].gameObject);

            // check which condition to invoke which event
            if (goalOfPlayerGotHit == 1 || AllBotKnockedOut())
            {
                needNewBall = false;
                unityEvents[EventName.GameOverEvent].Invoke(0);
            }
            unityEvents[EventName.KnockedOutEvent].Invoke(goalOfPlayerGotHit);
        }
        else
        {
            Destroy(StarPlayers[goalOfPlayerGotHit].trans-
form.GetChild(0).gameObject);
        }
```

```
        // check whether or not new ball is needed
        if (needNewBall)
        {
            unityEvents[EventName.RespawnBallEvent].Invoke(0);
        }
    }
```

Listing 6. Goal event handler

As illustrated in Listing 6, every time a goal got hit, the score of that player is decreased and one star on the board will be destroyed. If the player's score down to zero, that player will be knocked out. After that, a new ball needs to respawn.

5.11 Unity event

Unity event is a fairly complex system, indeed it improves the performance of the game significant. Instead of checking many conditions in the Update method in every script once every frame, that amount might increase rapidly when more features are added, using invoker and listener handle those tasks efficiently. The **EventManager** is initialized at the beginning of the game within the **AR Session Origin** as can be seen in Listing 7:

```
 public static void Initialize()
{
       // create empty lists for all the dictionary entries
       foreach (EventName name in Enum.GetValues(typeof(EventName)))
    {
            if (!invokers.ContainsKey(name))
        {
                invokers.Add(name, new List<IntEventInvoker>());
                listeners.Add(name, new List<UnityAction<int>>());
            }
        else
        {
                invokers[name].Clear();
                listeners[name].Clear();
            }
       }
 }
```

Listing 7. Initializing the Event manager

As shown in Listing 7, the **EventManager** is a static class, which contains two dictionary variables, namely invoker and listener. It loads and adds all invokers and listeners based on an enum class **EventName**, which stores all the names of events in the game. Furthermore, three methods to handle event process is depicted in Listing 8:

```
     public static void AddInvoker(EventName eventName, IntEventInvoker in-
voker)
    {
          // add listeners to new invoker and add new invoker to dictionary
          foreach (UnityAction<int> listener in listeners[eventName])
        {
              invoker.AddListener(eventName, listener);
          }
          invokers[eventName].Add(invoker);
     }

     public static void AddListener(EventName eventName, UnityAction<int> lis-
tener)
    {
          // add as listener to all invokers and add new listener to diction-
ary
          foreach (IntEventInvoker invoker in invokers[eventName])
        {
              invoker.AddListener(eventName, listener);
          }
          listeners[eventName].Add(listener);
     }

     public static void RemoveInvoker(EventName eventName, IntEventInvoker in-
voker)
    {
          // remove invoker from dictionary
          invokers[eventName].Remove(invoker);
     }
```

Listing 8. Processing event invoker and listener

In Listing 8, two method for adding new invoker and listener separately. The last method is to remove a particular invoker to prevent memory leakage. Most of the script in this project using this event system.

5.12 Power-up

Power-up is a useful added feature to provide more attractive and creative activities for the player, also encouraging the player to use more AR aspects. When the board is placed, a timer with a random duration is started for respawning a power-up at a random position above the board. Two of the three methods define the power-up can be seen in Listing 9:

```
    // always called before any Start functions and also just after a prefab
is instantiated
    void Awake()
    {
        // get random x, y and z for position of gameObject
        GameObject floor = GameObject.FindGameObjectWithTag("Floor");
```

```
        Vector3 meshColliderFloorSize = floor.GetComponent<MeshCol-
lider>().bounds.size;
        float randX = Random.Range(meshColliderFloorSize.x / 2, -meshCol-
liderFloorSize.x / 2);
        float randY = Random.Range(4, 8);
        float randZ = Random.Range(meshColliderFloorSize.z / 2, -meshCol-
liderFloorSize.z/ 2);

        randPowerUpPosition = new Vector3(randX, randY, randZ);
        Debug.Log(randPowerUpPosition);
    }

    // Start is called before the first frame update
    void Start()
    {
        Time.timeScale = 0.1f;

        // set new random postition for gameObject
        transform.position = randPowerUpPosition;
    }
```

Listing 9. Initialize the power-up

As illustrated in Listing 9, the **Awake()** method is typically called before any Start function, it randomizes a new position for the power-up and set that new position for the power-up in the **Start** method. At the same time, the time scale of the game is set to 0.1 when the power-up appears. The game enters the slow-motion mode, so the player can have enough time to pick up the power-up without losing attention to the ball.

Two scenarios have been designed for the power-up usage of real player and the computer enemies. If the real player obtains the power-up, one can activate the power-up anytime. After that, simply pointing at any enemy paddle and press a button to freeze that paddle in three seconds.

On the other hand, if the computer enemy got the power-up. A short random time after that, one big wall will appear above the main player's goal and block the sight of the paddle. Therefore, one needs to move around and turn the camera to see the paddle can gain back the control. The wall will disappear after 5 seconds.

Many operations associated with the power-up is carried out in the **Gameplay** and **UI-Manager** scripts. With this intention, all related methods in the UI manager are due to activity after pressing a button.

6 Conclusions

One journey is about to end. Throughout this report, various knowledge about Augmented reality has been explored and a project has been carried out to showcase what AR is capable of. That technology has huge potential to be popular soon.

Developing an AR mobile game is time-consuming and hard-working, which is similar to developing any software. The app performed pleasantly at the end due to most of the original ideas are fulfilled. Proper testing with an ideal number of testers has not carried out, there are only three people except the author has been playing the game with different versions. Based on their feedback, several improvements have been implemented to enhance performance and provide more only-AR-can-do experience.

Unfortunately, two expectations are not accomplished. First is the multiplayer mode, it requires more experience and time for server and cloud programming. Multiplayer in AR game and one in the normal mobile game are unalike at some extents. Second is the virtual object in the game did not blend ideally into the real world due to the shadow, lighting, and collision configurations.

Furthermore, challenges occurred in the development process. The author has to buy a new phone for testing because not all Android phone is ARCore supported. Besides, the testing phase is a demanding task, since all the tools for AR development are still in preview and not mature enough.

Moving beyond this report, many upgrades can be added to this project, such as complete the multiplayer mode, enhance the immersive experience from the AR perspective. One crucial element of a game, that has been forgotten the entire time of this project, is the audio. Future updates and the whole project can be found at the author's GitHub repository: https://github.com/NgocNguyen95/AR-Pong-ARFoundation. This thesis is believed to be an encouraging concept for not only the author but also other developers who are interested in AR technology.

References

1 Jonathan Linowes, Krystian Babilinski. Augmented Reality for Developers: Build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia [e-book]. Birmingham, UK. Packt. 2017.
URL: https://learning.oreilly.com/library/view/augmented-reality-for/9781787286436/

2 Joseph Hocking. Unity in Action: Multiplatform game development in C#. Second Edition [e-book]. New York, USA. Manning: 2018. p. 3-7.
URL: https://learning.oreilly.com/library/view/unity-in-action/9781617294969/

3 Jesse Glover. Unity 2018 Augmented Reality Projects [e-book]. Birmingham, UK. Packt: 2018. p. 13, 14.
URL: https://learning.oreilly.com/library/view/unity-2018-augmented/9781788838764/

4 Unity Technologies. About AR Foundation [Internet]. [cited 2020 May 22].
URL: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.0/manual/index.html

5 Dominic Cushnan, Hassan EL Habbak. Developing AR Games for iOS and Android [e-book]. Birmingham, UK. Packt: 2013.
URL: https://learning.oreilly.com/library/view/developing-ar-games/9781783280032/

6 Michelle Menard, Bryan Wagstaff. Game Development with Unity [e-book]. 2nd ed. Boston: CENGAGE Learning. 2014.
URL: http://ebookcentral.proquest.com/lib/metropolia-ebooks/detail.action?docID=3136735

7 Micheal Lanham. Learn ARCore - Fundamentals of Google ARCore: Learn to build augmented reality apps for Android, Unity, and the web with Google ARCore 1.0 [e-book]. Birmingham, UK. Packt. 2018.
URL: https://books.google.fi/books?id=05lUDwAAQBAJ&lpg=PP1&dq=learn%20arcore&lr&pg=PP3#v=onepage&q&f=false

8 Micheal Lanham. Augmented Reality Game Development [e-book]. Birmingham. UK. Packt. 2017.
URL: https://learning.oreilly.com/library/view/augmented-reality-game/9781787122888/index.html

9 Allan Fowler. Beginning iOS AR Game Development: Developing Augmented Reality Apps with Unity and C# [e-book]. Marietta, GA, USA. Apress: 2018.
URL: https://learning.oreilly.com/library/view/beginning-ios-ar/9781484236185/

Source code used for placing the board

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using UnityEngine.UI;
using System;

/// <summary>
/// Cast ray to demonstrate board game in real world
/// </summary>
[RequireComponent(typeof(ARRaycastManager))]
public class PlaceTheBoard : IntEventInvoker
{
    [SerializeField]
    [Tooltip("Instantiate this prefab on a plane at the ray")]
    GameObject board;
    Transform boardTransform;

    ARRaycastManager RaycastManager;
    ARPointCloudManager PointCloudManager;
    ARPlaneManager PlaneManager;
    ARSessionOrigin SessionOrigin;

    Vector2 centerScreenPos;
    static List<ARRaycastHit> listHits = new List<ARRaycastHit>();
    [SerializeField] Button PlaceHereButton;
    Pose hitPose;
    bool boardIsPlaced = false;

    Timer delayRespawnNewBoardTimer;

    /// <summary>
    /// The object instantiated as a result of a successful raycast intersection
with a plane.
    /// </summary>
    GameObject spawnedObject { get; set; }

    private void Awake()
    {
        RaycastManager = GetComponent<ARRaycastManager>();
        PointCloudManager = GetComponent<ARPointCloudManager>();
        PlaneManager = GetComponent<ARPlaneManager>();
        SessionOrigin = GetComponent<ARSessionOrigin>();
    }

    // Start is called before the first frame update
    void Start()
    {
        centerScreenPos = new Vector2(Screen.width / 2, Screen.height / 2);

        // add listener for restart game event
```

```
        EventManager.AddListener(EventName.RestartGameEvent, HandleRestart-
GameEvent);

        // add listener for game over event
        EventManager.AddListener(EventName.GameOverEvent, HandleGameOverEvent);

        // create timer
        delayRespawnNewBoardTimer = gameObject.AddComponent<Timer>();
        delayRespawnNewBoardTimer.Duration = 0.5f;
        delayRespawnNewBoardTimer.AddTimerFinishedEventListener(Han-
dleDelayRespawnNewBoardTimerFinishedEvent);

        PlaceHereButton.onClick.AddListener(PositionSelected);
    }

    // Update is called once per frame
    void Update()
    {
        if (!boardIsPlaced)
        {
            if (RaycastManager.Raycast(centerScreenPos, listHits, Trackable-
Type.PlaneWithinPolygon))
            {
                // Raycast hits are sorted by distance, so the first one
                // will be the closest hit.
                hitPose = listHits[0].pose;

                if (spawnedObject == null)
                {
                    spawnedObject = Instantiate(board);
                    SessionOrigin.MakeContentAppearAt(spawnedObject.transform,
hitPose.position, hitPose.rotation);
                }
                else
                {
                    SessionOrigin.MakeContentAppearAt(spawnedObject.transform,
hitPose.position);
                }
            }
        }
    }

    private void PositionSelected()
    {
        PlaceBoard();

        // diactivate existings trackable
        foreach (ARPlane plane in PlaneManager.trackables)
        {
            plane.gameObject.SetActive(false);
        }

        foreach (ARPointCloud pointCloud in PointCloudManager.trackables)
        {
            pointCloud.gameObject.SetActive(false);
        }
```

```
            // dissable plane and point cloud detections
            PointCloudManager.enabled = !PointCloudManager.enabled;
            PlaneManager.enabled = !PlaneManager.enabled;

            PlaceHereButton.gameObject.SetActive(false);
        }

        void PlaceBoard()
        {
            Debug.Log("Board placed!");
            boardIsPlaced = true;
            boardTransform = GameObject.FindGameObjectWithTag("PlayArea").gameOb-
ject.transform;
        }

        /// <summary>
        /// Handle restart game event
        /// </summary>
        /// <param name="unused">unused</param>
        void HandleRestartGameEvent (int unused)
        {
            Time.timeScale = 1;
            delayRespawnNewBoardTimer.Run();
        }

        void HandleDelayRespawnNewBoardTimerFinishedEvent()
        {
            GameObject newBoard = Instantiate(board);
            newBoard.transform.position = boardTransform.position;
            newBoard.transform.rotation = boardTransform.rotation;

            PlaceBoard();
        }

        /// <summary>
        /// Handle game over event
        /// </summary>
        /// <param name="unused">unused</param>
        void HandleGameOverEvent(int unused)
        {
            Destroy(GameObject.FindGameObjectWithTag("PlayArea").gameObject);
        }
}
```

Source code for controlling enemies movement

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

public class EnemyMovement : MonoBehaviour
{
    public GameObject enemy3;
    public GameObject enemy4;
    public GameObject enemy2;

    bool isFreeze2 = false;
    bool isFreeze3 = false;
    bool isFreeze4 = false;

    float speed = 10;
    float movAmount = 0.05f;

    bool[] knockedPlayers = { false, false, false, false, false };

    // getting the ball coordinates
    private Vector3 ballCoord;
    [SerializeField]GameObject ball;
    int i = 0;

    public bool move = true;

    Timer freezeEffectTimer;
    int playerFrozen;

    // Start is called before the first frame update
    void Start()
    {
        // add listener for ball respawned event
        EventManager.AddListener(EventName.BallRespawnedEvent, HandleBallRes-
pawnedEvent);

        // add listener for the knocked out event
        EventManager.AddListener(EventName.KnockedOutEvent, HandleKnockedOutE-
vent);

        // add listener for the player be freeze selected event
        EventManager.AddListener(EventName.PlayerBeFreezeSelectedEvent, Handle-
PlayerBeFreezeSelectedEvent);

        // add timer for the freeze effect
        freezeEffectTimer = gameObject.AddComponent<Timer>();
        freezeEffectTimer.Duration = 2;
        freezeEffectTimer.AddTimerFinishedEventListener(HandleFreezeEf-
fectTimerFinished);
    }

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

```
    {
        //Reading the ball's coordinates
        if (ball != null)
        {
            ballCoord = ball.transform.position;
            i = 0;

            //Tracking the ball and moving enemies
            if (move && ball.GetComponent<Ball>().ballmoves)
            {
                //Enemy 1
                if (!knockedPlayers[3] && !isFreeze3)
                {
                    if (enemy3.transform.position.x <= ballCoord.x && en-
emy3.transform.position.x <= 2.5F)
                    {
                        enemy3.transform.position = new Vector3(
                            enemy3.transform.position.x + movAmount,
                            enemy3.transform.position.y,
                            enemy3.transform.position.z);
                    }

                    if (enemy3.transform.position.x >= ballCoord.x && en-
emy3.transform.position.x >= -2.5F)
                    {
                        enemy3.transform.position = new Vector3(
                            enemy3.transform.position.x - movAmount,
                            enemy3.transform.position.y,
                            enemy3.transform.position.z);
                    }
                }


                //Enemy 2
                if (!knockedPlayers[4] && !isFreeze4)
                {
                    if (enemy4.transform.position.z <= ballCoord.z && en-
emy4.transform.position.z <= 2.5F)
                    {
                        enemy4.transform.position = new Vector3(
                            enemy4.transform.position.x,
                            enemy4.transform.position.y,
                            enemy4.transform.position.z + movAmount);
                    }

                    if (enemy4.transform.position.z >= ballCoord.z && en-
emy4.transform.position.z >= -2.5F)
                    {
                        enemy4.transform.position = new Vector3(
                            enemy4.transform.position.x,
                            enemy4.transform.position.y,
                            enemy4.transform.position.z - movAmount);
                    }
                }


                //Enemy 3
```

```
                if (!knockedPlayers[2] && !isFreeze2)
                {
                    if (enemy2.transform.position.z <= ballCoord.z && en-
emy2.transform.position.z <= 2.5F)
                    {
                        enemy2.transform.position = new Vector3(
                            enemy2.transform.position.x,
                            enemy2.transform.position.y,
                            enemy2.transform.position.z + movAmount);
                    }

                    if (enemy2.transform.position.z >= ballCoord.z && en-
emy2.transform.position.z >= -2.5F)
                    {
                        enemy2.transform.position = new Vector3(
                            enemy2.transform.position.x,
                            enemy2.transform.position.y,
                            enemy2.transform.position.z - movAmount);
                    }
                }
            }
        }
    }

    /// <summary>
    /// Handle ball respawned event
    /// </summary>
    /// <param name="unused">unused</param>
    void HandleBallRespawnedEvent (int unused)
    {
        ball = GameObject.FindGameObjectWithTag("Ball");
    }

    /// <summary>
    /// Handle the knocked out event
    /// </summary>
    /// <param name="playerKnockedOut">player who knocked out</param>
    void HandleKnockedOutEvent (int playerKnockedOut)
    {
        if (playerKnockedOut != 1)
        {
            knockedPlayers[playerKnockedOut] = true;
        }
    }

    /// <summary>
    /// Handle the player be freeze selected event
    /// </summary>
    /// <param name="playerGotFreeze">player who got freeze effect</param>
    void HandlePlayerBeFreezeSelectedEvent (int playerGotFreeze)
    {
        if (playerGotFreeze == 2)
        {
            isFreeze2 = true;
        }
        else if (playerGotFreeze == 3)
        {
```

```
                isFreeze3 = true;
            }
            else if (playerGotFreeze == 4)
            {
                isFreeze4 = true;
            }

            playerFrozen = playerGotFreeze;
            freezeEffectTimer.Run();
        }

        void HandleFreezeEffectTimerFinished ()
        {
            if (playerFrozen == 2)
            {
                isFreeze2 = false;
            }
            else if (playerFrozen == 3)
            {
                isFreeze3 = false;
            }
            else if (playerFrozen == 4)
            {
                isFreeze4 = false;
            }
        }
    }
```

Source code of the gameplay script

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

public class Gameplay : IntEventInvoker
{
    [SerializeField] GameObject BallPrefab;
    [SerializeField] GameObject PlayArea;

    [SerializeField] GameObject PowerUpPrefab;
    [SerializeField] GameObject BlockWallPrefab;
    GameObject powerUp;

    [SerializeField] GameObject[] Players = new GameObject[5];

    Timer ballRespawnDelayTimer;
    Timer powerUpRespawnTimer;
    Timer enemyTakePowerUpTimer;
    Timer enemyUsePowerUpTimer;

    bool gameOver = false;

    // Start is called before the first frame update
    void Start()
    {
        // add listener for board placed event
        EventManager.AddListener(EventName.BoardPlacedEvent, HandleBoard-
PlacedEvent);

        // add listener for game started event
        EventManager.AddListener(EventName.GameStartedEvent, Hand-
leGameStartedEvent);

        // add as listener for respawn ball event
        EventManager.AddListener(EventName.RespawnBallEvent, HandleRespawn-
BallEvent);

        // add listener for knocked out event
        EventManager.AddListener(EventName.KnockedOutEvent, HandleKnockedOutE-
vent);

        // add as invoker for power up respawned event
        unityEvents.Add(EventName.PowerUpRespawnedEvent, new PowerUpRes-
pawnedEvent());
        EventManager.AddInvoker(EventName.PowerUpRespawnedEvent, this);

        // add listener for power up taken event
        EventManager.AddListener(EventName.PowerUpTakenEvent, HandlePowerUpTak-
enEvent);
```

```
        // add this as invoker for power up taken event
        unityEvents.Add(EventName.PowerUpTakenEvent, new PowerUpTakenEvent());
        EventManager.AddInvoker(EventName.PowerUpTakenEvent, this);

        // create timer for ball respawn delay
        ballRespawnDelayTimer = gameObject.AddComponent<Timer>();
        ballRespawnDelayTimer.Duration = 0.5f;
        ballRespawnDelayTimer.AddTimerFinishedEventListener(HandleBallRespawnDe-
layTimerFinishedEvent);

        // create timer for power up respawn
        powerUpRespawnTimer = gameObject.AddComponent<Timer>();
        powerUpRespawnTimer.Duration = RandomPowerUpRespawnDuration();
        powerUpRespawnTimer.AddTimerFinishedEventListener(HandlePowerUpRespawn-
TimerFinished);

        // create timer for enemies take powerUp
        enemyTakePowerUpTimer = gameObject.AddComponent<Timer>();
        enemyTakePowerUpTimer.Duration = 0.3f;
        enemyTakePowerUpTimer.AddTimerFinishedEventListener(HandleEnemyTakePow-
erUpTimerFinished);

        // create timer for enemis use powerUp
        enemyUsePowerUpTimer = gameObject.AddComponent<Timer>();
        enemyUsePowerUpTimer.Duration = 3f;
        enemyUsePowerUpTimer.AddTimerFinishedEventListener(EnemyUsePowerUp);
    }

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

    }

    /// <summary>
    /// Handles the BoardPlacedEvent
    /// </summary>
    /// <param name="unused">unused</param>
    void HandleBoardPlacedEvent(int unused)
    {
        PlayArea = GameObject.FindGameObjectWithTag("PlayArea");

        // get all players and add to the players array
        for (int i = 1; i < Players.Length; i++)
        {
            GameObject player = GameObject.FindGameObjectWithTag("Player" + i);
            Players[i] = player;
        }
        gameOver = false;
    }

    /// <summary>
    /// Handle the game started event
    /// </summary>
    /// <param name="unused">unused</param>
    void HandleGameStartedEvent(int unused)
    {
```

```
            RespawnBall();
            powerUpRespawnTimer.Run();
        }

        /// <summary>
        /// Respawn a ball when call and invoke ball respawned event
        /// </summary>
        void RespawnBall()
        {
            Instantiate(BallPrefab, PlayArea.transform);
        }

        /// <summary>
        /// Handle repsawn ball event
        /// </summary>
        /// <param name="unused">unused</param>
        void HandleRespawnBallEvent (int unused)
        {
            ballRespawnDelayTimer.Run();
        }

        /// <summary>
        /// Handle knocked out event
        /// </summary>
        /// <param name="playerKnockedOut">player who knocked out</param>
        void HandleKnockedOutEvent (int playerKnockedOut)
        {
            Destroy(Players[playerKnockedOut].gameObject);
        }

        void HandleBallRespawnDelayTimerFinishedEvent()
        {
            RespawnBall();
        }

        /// <summary>
        /// Take a random duration for power up timer
        /// </summary>
        /// <returns>return a random float number</returns>
        float RandomPowerUpRespawnDuration()
        {
            return Random.Range(20, 40);
        }

        void HandlePowerUpRespawnTimerFinished()
        {
            powerUp = Instantiate(PowerUpPrefab, PlayArea.transform);
            unityEvents[EventName.PowerUpRespawnedEvent].Invoke(0);

            enemyTakePowerUpTimer.Run();
        }

        /// <summary>
        /// Handle power up taken event
        /// </summary>
        /// <param name="playerTookPowerUp">player number who took the power
up</param>
```

```
    void HandlePowerUpTakenEvent (int playerTookPowerUp)
    {
        Destroy(powerUp.gameObject);
        Time.timeScale = 1;
        powerUpRespawnTimer.Duration = RandomPowerUpRespawnDuration();
        powerUpRespawnTimer.Run();

        if (playerTookPowerUp == 1)
        {
            enemyTakePowerUpTimer.Stop();
        }
        else
        {
            enemyUsePowerUpTimer.Run();
        }
    }

    void HandleEnemyTakePowerUpTimerFinished ()
    {
        unityEvents[EventName.PowerUpTakenEvent].Invoke(0);
    }

    void EnemyUsePowerUp()
    {
        Instantiate(BlockWallPrefab, PlayArea.transform);
    }
}
```