DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

The Design and Development of a Multiplayer Augmented Reality Platform-Independent Framework for Superhuman Sports

Tarek Elsherif

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

The Design and Development of a Multiplayer Augmented Reality Platform-Independent Framework for Superhuman Sports

Das Design und die Entwicklung eines plattformunabhängigen Erweitere Realität Multiplayer-Framework für Superhuman Sports

Author:	Tarek Elsherif
Supervisor:	Prof. Klinker Gudrun Johanna
Advisor:	Eichhorn Christian
Submission Date:	15.06.2022

I confirm that this master's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15.06.2022 Tarek Elsherif

Acknowledgments

By the time I started working on this research, the effects of the Corona pandemic were still lingering and most of the research work was done from home. This was quiet a challenge for me as it highly affected my motivation. However, through my advisor's support and an extra push from my friends and family, this work was made possible.

So I would like to thank Christian Eichhorn, my advisor, for his support and guidance. His insights and feedback greatly assisted my work during the research.

I would also like to thank Prof. Gudrun Klinker for providing me with this opportunity. My appreciation extends to Sandro Weber and Maximilian Schmidt for their help with Ubi-Interact.

Finally, I would like to express my gratitude to my friends and family, especially to Pakinam Zeid, my wife.

Abstract

The field of superhuman sports is a new emerging discipline that has recently attracted both public and academic interest. It is considered to be a reform of sports as we know them, as it aims to utilize technology to provide an augmented experience to its participants. There is a wide range of technologies involved in this field, including robotics, augmented reality (AR), and real-time networking. This can render superhuman sports as complex systems and can create a gap between the developers and these technologies. This research aims to tackle this problem by presenting a framework that facilitates the process of developing superhuman sports games, specifically ones that utilize network-based AR and can work on several different platforms. It uses the ARSSP (Augmented Reality Superhuman Sports Platform) framework as a foundation and extends it with new features that tackle its limitations. The framework is then evaluated by assessing how well it can successfully create a superhuman sports game. The research arrives at promising results that would consider this framework as a forward step in the studies of superhuman sports that can help in their development and benefit future research.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Goal . . . 2
1.3 Structure of the thesis . . . 3

2 Literature Review **4**
2.1 Superhuman Sports . . . 4
2.2 Multiplayer AR Frameworks . . . 5
2.2.1 SARA . . . 5
2.2.2 CARS . . . 6
2.2.3 AR Flutter plugin . . . 7
2.3 Example Superhuman Sports Games . . . 7
2.3.1 VRabl . . . 8
2.3.2 League of Lasers . . . 9
2.3.3 STAR . . . 10
2.4 Example Multiplayer AR Games . . . 10
2.4.1 Holofight . . . 11
2.4.2 ARSOCCER . . . 11
2.4.3 Augmented Invaders . . . 12
2.4.4 Brick . . . 13
2.5 Software Design Patterns in Games Engineering . . . 14
2.5.1 Component Pattern . . . 14
2.5.2 Observer Pattern . . . 14
2.5.3 Singleton Pattern . . . 15
2.5.4 Command Pattern . . . 16
2.6 Technologies . . . 17
2.6.1 Game Engines . . . 17
2.6.2 Hardware Devices . . . 19
2.6.3 AR Frameworks and Solutions . . . 21
2.6.4 Networking Frameworks . . . 27

3 Implementation **29**
3.1 ARSSP Architecture . . . 29

3.2 ARSSP Frameworks . . . 31
3.2.1 AR Framework . . . 31
3.2.2 Networking Framework (BEvent) . . . 32
3.2.3 User Interface Framework . . . 34
3.2.4 Game Framework . . . 35
3.3 Extended Features . . . 36
3.3.1 Game Engine Upgrade . . . 36
3.3.2 Networking Frameworks Upgrade . . . 37
3.3.3 Magic Leap HMD Support . . . 37
3.3.4 World Space UI Interaction Support . . . 38
3.3.5 Multiplayer Network Lobby System . . . 44
3.3.6 External Hardware Integration . . . 50
3.3.7 OpenCV Support . . . 55
3.4 ARSSP Game Template . . . 58
3.4.1 Managers Spawner . . . 58
3.4.2 Global Managers Prefab . . . 59
3.4.3 Player Prefab . . . 62
3.4.4 Template Scene . . . 62

4 Evaluation **64**
4.1 Technical Aspects of Catching the Drone . . . 64
4.1.1 Networking . . . 64
4.1.2 Augmentation . . . 65
4.1.3 Game Logic . . . 65
4.2 ARSSP Assessment . . . 65
4.2.1 Networking . . . 66
4.2.2 Augmentation . . . 66
4.2.3 Game Logic . . . 67
4.3 Results . . . 68

5 Conclusion **69**
5.1 Summary . . . 69
5.2 Future Work . . . 69

List of Figures **70**

Bibliography **72**

1 Introduction

1.1 Motivation

Throughout the ancient times, sports have been a big part of humans' lives. Sports act as means of improving and maintaining the physical abilities and skills, while providing enjoyment to their participants and entertainment to their spectators. There are many different types of sports, ranging from those with a single player to those with teams or even hundreds of contestants. With the consistent advancements in technology, humans have seen technological evolution in every field in life, and sports is no different. In fact, technology helped introduce new different types of sports, and one particular revolutionary type is the Superhuman Sports.

As defined by Kunze K. et al. in their research about superhuman sports, "The field of superhuman sports combines competition and physical elements from traditional sports with technology to overcome the somatic and spatial limitations of our human bodies" [1]. In other words, superhuman sports focus on using technology to provide the participates with physical and cognitive improvements that render them as augmented or super humans. This augmentation can be targeted at different aspects of the sport, including the human body, the sport field, or the players' vision. In superhuman sports, players can do things that they can not normally do without technology like, for example, extra strength, flying, high jumping, or augmented vision.

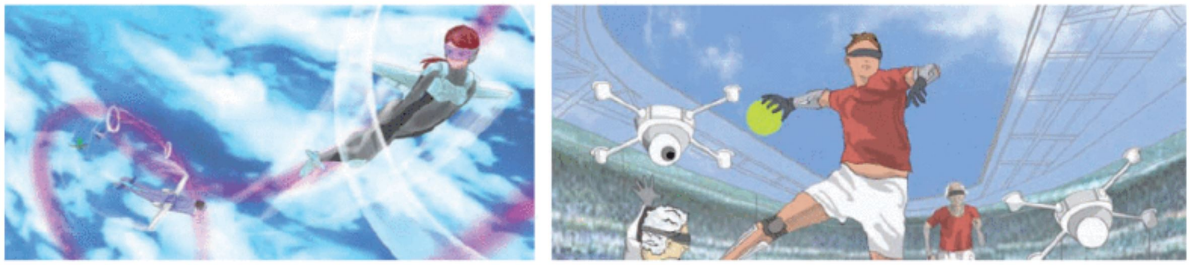


Figure 1.1: Concept art of novel superhuman sports game ideas.

Although this new sports field is currently still in development, there are several superhuman sports that are currently active and well-recognized. Sports like Bubble Jumper (or Bubble Sumo) [2] and HADO [3] are examples of such superhuman sports. Furthermore, this field open up a lot of different possibilities, including augmenting old sports or creating new sports with new concepts. Some inspirations of new sport ideas can also be drawn

from fictional fantasy sports like, for example, Quidditch from the *Harry Potter* series. In fact, a research by Eichhorn C. et al. introduces the concept of the superhuman sports game Catching the Drone [4], which uses a drone-ball in a competitive team-based game that is highly inspired by Quiddich.

Despite the fact that there are some implemented superhuman sport games, the field is still novel and is far from being mainstream. There are several technical challenges in realizing and implementing some of the superhuman sports game concepts, but with the consistent improvement in fields of robotics, real-time networking, and augmented reality (AR), a lot of game concepts can come to reality. However, there is still a gap between developers and tools that would help them create such games, because in some cases superhuman sports games can require very complex systems. For that purpose, this research tries to minimize that gap by proposing a framework that can facilitate the process of creating superhuman sports games. This framework would focus specifically on multiplayer AR sport games that can be played cross-platform. A game like Catching the Drone [4] can be a good example for such a superhuman sports game.

1.2 Goal

This research aims to present a framework that would fill the gap between developers and AR Multiplayer Superhuman Sports games. This framework would utilize the recent advancements in technologies like augmented reality and networking to provide an out-of-the-box tool to help developers create AR multiplayer superhuman sports games without worrying about the low-level tools working underneath. There are key characteristics that define this framework, and they are the following:

- **Familiar Environment.** This framework takes on a form of a plugin or a template that would be used in an accessible development environment that most developers are already familiar with, like Unity or Unreal game engines.
- **Platform-Independent.** The framework is cross-platform and can work on several different devices with different operating system. It should help allow the developers to deploy the application to several different platforms (e.g. Android, iOS, and LuminOS) without requiring significant changes for porting.
- **Plug-and-develop.** The framework template provided is a functional out-of-the-box solution without making the developer worry about the dependencies of the the framework.
- **Open-source and Modular.** The architecture of the framework is modular, or, in other words, is divided into loosely-coupled layers with each has its own responsibility and can be modified without altering other layers. Furthermore, it is open-source as it allows for the developers to modify the framework to their specific needs if required.

With these key characteristics defined, this research does not implemented this framework from scratch. Instead it utilizes a framework presented by Ben Jazia M. [5] as a foundation which is already functional and provides these key characteristics. However, this research tries to evaluate it, tackle its limitations, and extend it with features that would make it a better functional tool to create superhuman sport games while maintaining the key characteristics of the framework. In order to achieve this, several objectives have been drawn which act as an outline for this research. These objectives are as follows:

1. Review related work and literature on key topics that can build a solid foundation for this research. This includes (1) other AR network-based frameworks developed, (2) software engineering design patterns used in games, (3) state-of-the-art AR superhuman sports games, (4) other relevant multiplayer AR games, and (5) relevant technologies with their recent advancements.
2. Review and present the design and software architecture of the ARSSP (AR Superhuman Sports Platform) framework presented by Ben Jazia [5].
3. Modify and extend the ARSSP framework with new features and functionalities based on its limitations and the games reviewed in the literature.
4. Evaluate the newly modified framework by assessing how well it technically realizes a superhuman sports game like Catching the Drone [4], and then present those results.

1.3 Structure of the thesis

In the following chapter, a literature review on related work and relevant topics will be discussed. Afterwards, the implementation will be presented, starting with an overview of the ARSSP framework, then the extended features and modifications implemented, and ending with presenting a final result template of the framework. The following chapter will then evaluate the implementation and present results, and then the thesis will be concluded in the final chapter.

2 Literature Review

In order to lay a solid foundation for which this research and its implementation will be based on, several state-of-the-art frameworks, AR applications, and technologies have been reviewed and evaluated. This chapter aims first to provide a definition to the Superhuman Sports in the scope of this research. It will then present some AR frameworks focusing on their software architecture and design overview and patterns - most of these frameworks are not essentially aimed for Superhuman sports nevertheless they provide relevant solutions in the scope of Multiplayer AR applications. Afterwards, some games and Superhuman sports applications will be presented to provide an insight of the current state of the AR multiplayer scene. Then the topic of software design patterns used in games will be discussed. Finally, this chapter will arrive at an overview of the technologies that are essential or useful for the implementation of this research.

2.1 Superhuman Sports

The Superhuman Sports is considered as re-inventions of sports as we know them. Its existence is new and it can be traced back to the Superhuman Sports Society [6] (or the Superhuman Sports Academy), which started in Japan in 2015. the Superhuman Sports Society is composed of a group of researchers and professionals in several fields, including technology engineering, media, game design, and sports. Their aim is to create sports that are combined with technology and to consistently develop these sports in order to make everyone - including young, elder, healthy, disabled, amateur, professional - enjoy them. People participating in these sports are considered as "device-integrated" humans. The extended abilities provided by the Superhuman Sports can include extended strength, flying, teleportation, super dynamic vision, or object manipulation. These features and abilities can open up for endless possibilities for designs and rules in superhuman sport games.

Figure 2.1: Superhuman Sports Society logo.

The technologies used by superhuman sports can include external mechanical devices, special googles, robots, or drones. Another technology involved in these sports is Augmented Reality (AR), where augmentation of virtual objects in real life is usually done through AR Head-Mounted-Displays or smartphones. One example of such sports is HADO [3]. HADO combines AR technology, motion sensor, Head-mounted display, smartphone, and physical movement to create a Superhuman Sport where teams can compete to earn scores by shooting virtual energy balls against the other team (Figure 2.2). It was established in 2017 and became one of the most well-established superhuman sports where it has international tournaments, featuring teams from all around the world. HADO gained a lot of popularity and interest into the field of Superhuman Sports and is currently acting as inspiration for further ideas in AR multiplayer sport games.

Figure 2.2: Third-person perspective play capture of HADO.

2.2 Multiplayer AR Frameworks

In this section, an overview of several AR frameworks will be presented. The focus will be more on the structure of the system architecture of these frameworks. Although these Multiplayer AR frameworks are more targeted towards collaborative AR applications, their design decisions can still be very beneficial to this research.

2.2.1 SARA

In one of the recent researches, SARA, or Shared Augmented Reality experiences and Applications, was introduced by Vaquero-Melchor et al. [7]. SARA is a framework built with a microservice-based architecture that is used to facilitate the development and deployment of collaborative cross-platform Augmented Reality experiences. Its architecture consists of a set of services that will enable the communication among the participants and the management

of the collaboration itself. Figure 2.3 shows an overview of the SARA architecture with its communication and collaboration management micro-services and clients. Each client represents a combination of a user and a device, and the participants that are going to collaborate in the application can either use one of the clients at a time or combine several of them.

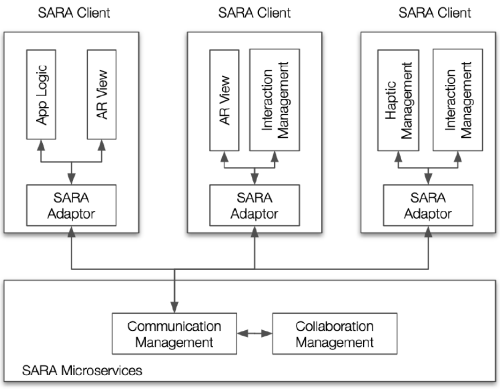


Figure 2.3: High-level overview of the SARA architecture.

The architecture of SARA is designed based on the micro-service paradigm [8]. This design ensures that each service is loosely coupled and small, and thus assuring the maintainability, scalability, and reusability of the system. This also means that the architecture can still be functional even if a given service is down or not working. Figure 2.3 shows the full architecture of SARA and the implementation of different services. The architecture can be divided into three parts that are integral to the system: (1) The Communcation Service, which handles the communication among other services, (2) the SARA Client, which, as mentioned before, representes a combination of a user and a device, and (3) the Collaboration Services, which contains several services that are used for collaboration between users.

2.2.2 CARS

Another framework that is aimed for collaborative Augmented Reality is CARS (Colaborative Augemented Reality for Socialization) [9]. In this research, CARS is presented as a framework that improves the Quality of Experience for AR in terms of end-to-end latency and reuse of networking and computation resources. The core functionality of CARS is that it helps users exchange Augmented Reality data while using cloud-based AR [10], which means that the image/object recognition is done on a cloud-based engine, thus potentially saving time and resources for each user.

An example was drawn in the paper, where two users are each standing in front of a different painting and using cloud-based recognition they get AR results for their respective painting (i.e. image recognition and annotation content). Now when the two users switch their paintings, instead of running cloud-based recognition again which may waste mobile

data usage and computation resources, they can simply exchange the results of previous runs through Device-to-Device communications, which is made possible through CARS. Using cloud-based AR also is an integral part for this efficiency, as the previous work of Zhang et al. [11] demonstrates that doing the recognition directly in the pipeline takes noticeably longer time.

The design of CARS consisted of two main building blocks: the APP layer and Base layer. The APP layer is responsible for retrieving the camera frames for object recognition and tracking of recognized objects, and the layer can be further divided into three components: Recognition Proxy, Visual Tracker, and Annotation Renderer. On the other hand, the Base layer is responsible for handling the object recognition, downloading annotations, and synchronizing with the APP layer, and this layer can also be divided into five components: Motion Detector, Scene Matcher, Annotation Synchronizer, Peer Manager, and Cloud Manager. The decoupling of the APP and Base layers makes it possible to scale and modify each layer independently from the other. Furthermore, it makes it easy for integration other AR applications into CARS.

2.2.3 AR Flutter plugin

Another collaborative framework that uses Cloud-based AR was introduced by Carius et al. [4]. This framework is aimed to make it easier to develop multi-user AR applications for businesses. It's key characteristics is that it works on a familiar development environment, provides cross-platform functionality for iOS and Android, is open-source and accessible to everyone, and supports collaboration for its users. Unlike most frameworks which choose an engine-based approach for their framework, like Unity, this framework works on Flutter, which is a popular open-source cross-platform SDK [12] developed by Google, and is commonly used in starting businesses.

The AR Flutter framework can be seen as a plugin which can be used to build AR applications on top of it. As seen in Figure 2.4, the system architecture of the plugin itself consists of two main building blocks: a platform-specific implementation block and a cross-platform API. The platform-specific block consists of all the functionality that are specific to either Android or iOS and can not abstracted to the Flutter level. On the other hand, the cross-platform API is common on all platforms and it exposes interfaces that can be used by the AR application using the plugin.

2.3 Example Superhuman Sports Games

In this section, some superhuman sports examples will be presented. The aim is to provide an insight of the state of some of the recent Superhuman Sports games and present briefly their technical implementation as well as some of the challenges they may have faced.

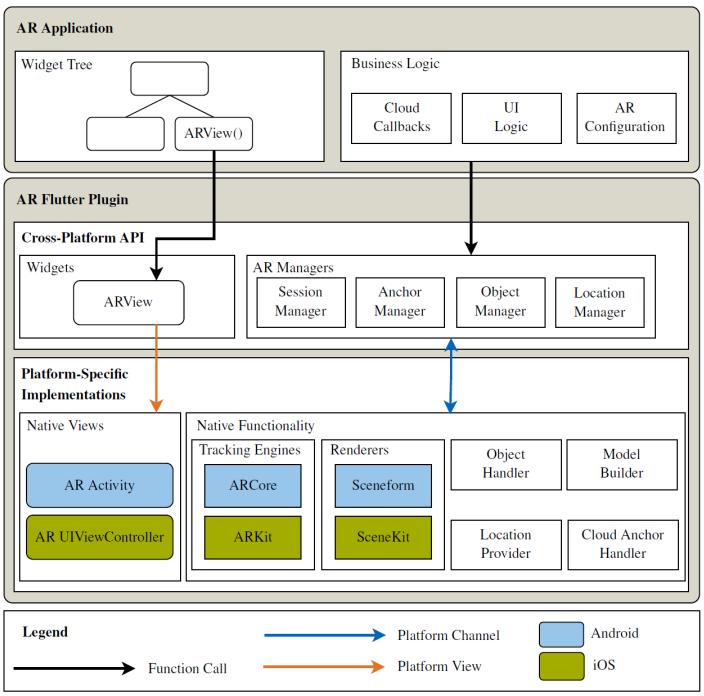


Figure 2.4: System architecture of AR Flutter Plugin.

2.3.1 VRabl

One of the researches introduced an AR superhuman sports game, VRabl [13]. VRabl is a multiplayer AR sports game that stimulates the players activity using gamified aspects. It plays with very similar rules to the popular physical sport dodgeball, while making it more engaging and attractive for younger generations. The game relies on players using Microsoft's Hololens AR HMDs which provides the user with the tracking and augmentation.

VRabl was developed on Unity engine and uses Photon Networking. It also uses the HoloToolKit, which is a toolkit that is intended to accelerate development of holographic applications targeting Windows Hololens [14]. Two different networking servers are used to enable exchange of data several Hololens devices used in the game. The first server is the Microsoft Sharing Service, which is part of the Sharing library within the HoloToolkit, and it is responsible for sharing game objects between users. The second server is the Photon Unity Networking framework, which is a cloud server that is responsible for communication between devices. This server also handles sharing gamedata as Photon provides better support for real-time multiplayer games.

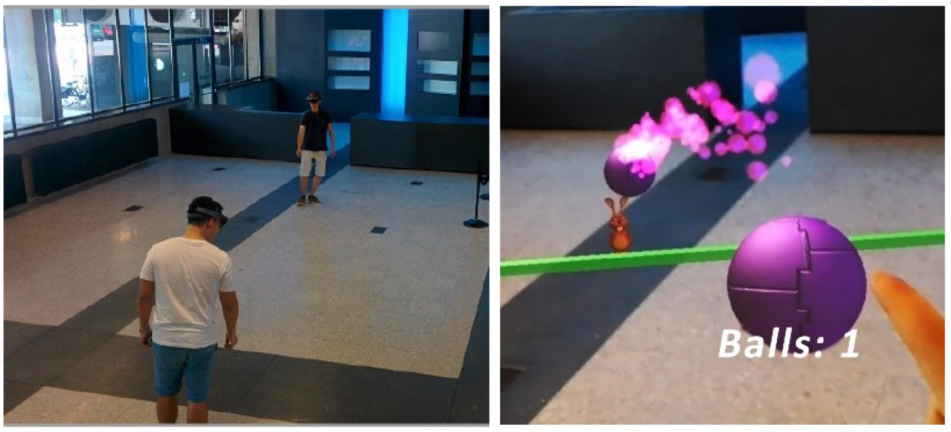


Figure 2.5: Players getting ready to play VRabl (left) and a screenshot of the gameplay (right).

2.3.2 League of Lasers

Another example of a Superhuman Sports Game that requires physical activity is League of Lasers [15]. League is Lasers uses mixed-reality and physical motion to provide an immersive superhuman sports experience. The idea of the game is inspired by the famous classic game, Pong. Each player plays in an augmented play area and controls a virtual mirror that reflects a laser pulse upon collision. The mirror is attached to the player's head so that it follows the exact rotation and position of the head. The game would involve two teams, where each team is responsible to protect their target from the laser pulse. Figure 2.6 provides a concept of the gameplay.

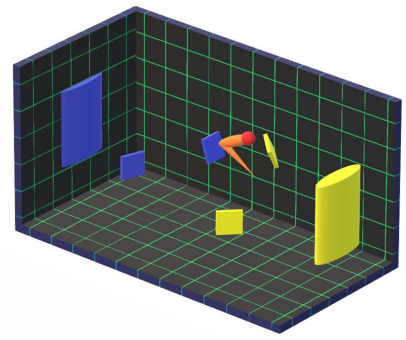


Figure 2.6: Overview of League of Lasers game concept.

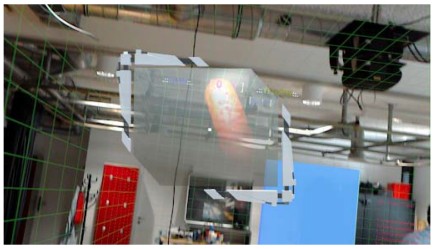


Figure 2.7: Screenshot of League of Lasers' mirror.

Similar to VRabl, League of Laser uses Microsoft Hololens AR HMDs. It also uses a dedicated client-server architecture, where the server is run on a desktop and the clients are on the Hololense devices. This avoids overloading the devices with tasks and makes it possible to use the server for a spectator view of the game, which can be used to be projected

to the audience. In addition to the game server, there is also a web-server that is used for sharing files between clients (e.g. anchor data files).

2.3.3 STAR

STAR, or Superhuman Training in Augmented Reality, is another superhuman game that relies on physical activity [16]. It is an adventure shooter game where the player has to navigate through a narrow path above lava while gathering energy and avoiding or shooting enemies. The game also provides a cooperative multiplayer mode where players work together to destroy a virtual energy core. The goal of STAR is to survive long enough to be able to destroy the energy core.

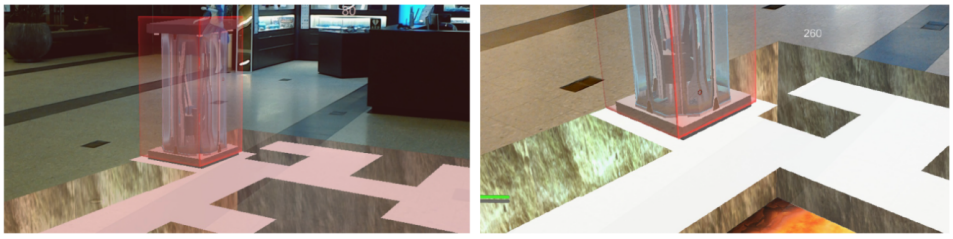


Figure 2.8: Screenshots of STAR taken from the Hololens view.

STAR was implemented on the Microsoft Hololens AR HMD. Windows Sharing Service was used for communication between devices in the multiplayer mode, and a spatial anchor was used to create a fixed point in the real world in which all objects are anchored to it. Although this is sufficient and functional, the paper mentions that there is a challenge imposed by synchronizing anchors. The problem is that anchors do not align between devices, and this is caused by the fact that the spatial anchors depend on feature points that are detected by the Hololens in the real environment. These feature points are exported and shared between devices. If the exporting device didn't scan enough feature points or both Hololens devices did not start in the same position, the exported feature points will become insufficient. Thus, in order to fix this, STAR includes a mandatory scan at the start of the game.

2.4 Example Multiplayer AR Games

In this section, some examples of multiplayer AR games will be presented. Although these games are not superhuman sports related, they still solve similar technical problems and use similar technologies. The concept of each game will be provided as well as a brief discussion of their technical implementation and the challenges they might have faced during execution.

2.4.1 Holofight

Holofight is introduced as a multiplayer Augmented Reality fighting game [17]. It provides a new immersive gameplay for a rather traditional game genre that can be experienced by colocated and remote users through the Hololens AR HMDs. The game is developed using Unity engine, and the game logic is built above several third party APIs, including Mirosoft's MRTK (Mixed Reality Toolkit) [18] for tracking and Hololens support and Photon PUN2 for Networking. Figure 2.10 gives a brief overview of the application structure.

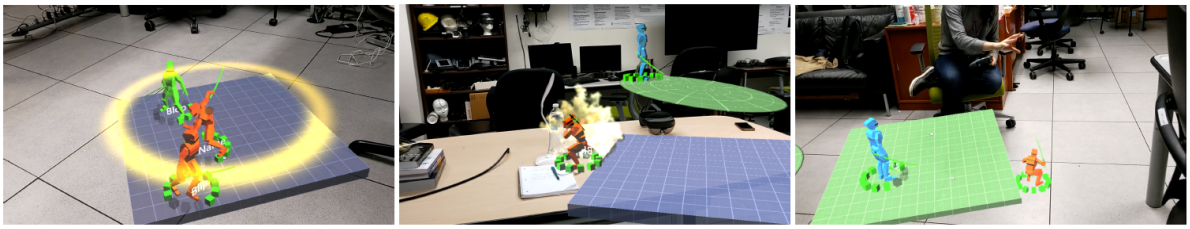


Figure 2.9: Screenshots of Holofight demo.

There are two ways players can play Holofight together, either in the same space or remotely in different rooms. The application uses MRTK to track hand gestures, eye and head gaze to align multiple players' perspectives during calibration, and Azure Spatial Anchors are used to automatically align the players' perspectives when that are in the same space. However, if they are playing remotely, alignment can be manually done through marker tracking of a playcard.

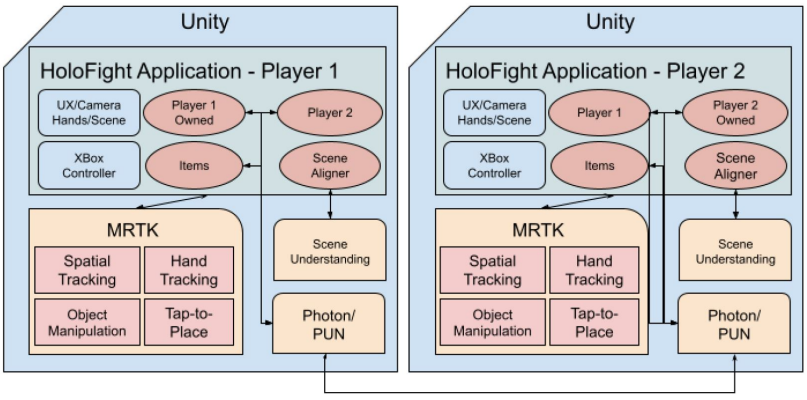


Figure 2.10: Overview of Holofight application structure.

2.4.2 ARSOCCER

Similar to Holofight, ARSOCCER [19] takes a traditional popular videogame genre and implements it in Augmented Reality. It is introduced as a mobile AR game that is inspired by

the popular soccer simulator videogame, FIFA [20]. The game implements similar mechanics to FIFA with the innovative addition of making the game viewable in real space through a mobile's camera using AR. This provides the player with the ability to have better control of the camera, making it easier to move around the field and follow the virtual ball or players. In addition to being able to play with bots using artificial intelligence, the game has a multiplayer feature, where players with two different devices can compete in the game.

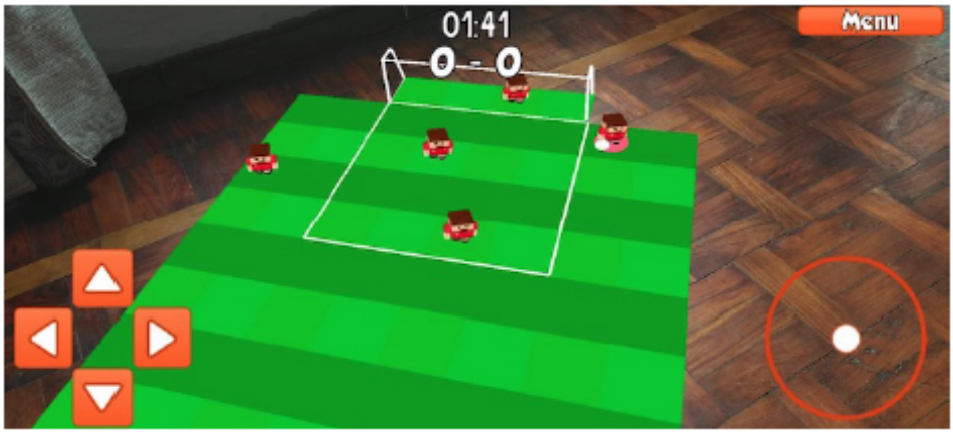


Figure 2.11: Screenshot for ARSoccer demo.

The game was implemented using Unity engine, AR Foundation, and Colyseus [21]. Unity was used as the game engine that runs the client, and AR Foundation plugin was used to provide augmentation while enabling it to work on cross-platforms, Android and iOS. Colyseus, which is a cloud-based multiplayer framework for NodeJS, was used to implement the servers of the game. The servers enables the communication between the players, computes the overall state of each game, and provides the ability to create lobbies. There is also an AI server, which was built with NodeJS, and computes the actions of bots during an ongoing game.

2.4.3 Augmented Invaders

Augmented Invaders is a multiplayer AR game that is implemented on mobile and can also be played while using the mobile as an MR headset [22]. The game is designed to be played outdoors where players can play together and their goal is to fight against virtual alien drones and destroy them. The game uses geographic location to calculate the relative position between the players and the mobile's rear camera for augmentation of the environment.

The game was built using Unity engine, for networking and communication between players, UNet [23] was used, which is a built-in networking service for Unity. For creating an augmented scene, Vuforia toolkit [24] was used, which is plugin that handles augmentation using the mobile's camera. The positions of the players are obtained by from devices' GPS

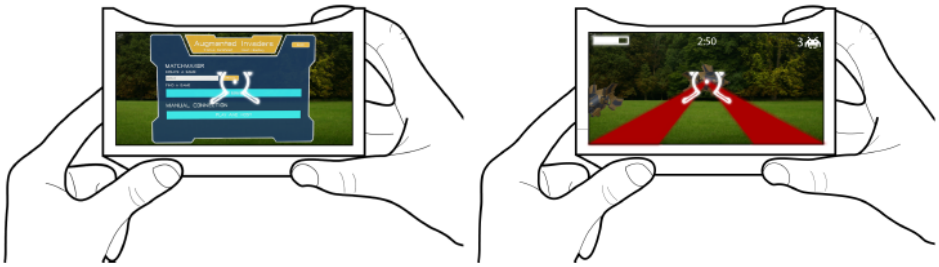


Figure 2.12: Illustration of Augmented Invaders' concept.

and converted to the Cartesian Unity world coordinate system. In order to synchronize the positions of the virtual elements in the scene between players, the player who created the game session determines the origin of the virtual world and shares it with other players.

2.4.4 Brick

Brick is a multiplayer AR synchronous game [25]. The goal of the game is that players collaborate together to fill in a pattern of empty slots with virtual "blocks" that are scattered around the place, as seen in Figure 2.13. In Bricks gameplay, there are four different colors for the bricks and each player is assigned two colors to interact with. Players have to concurrently transport the bricks using tap-and-hold interaction in AR and place them in their correct spots in the wall in order to fill the whole wall and win the game.

Figure 2.13: Screenshots of Brick on two devices.

Unlike the previously mentioned games that were developed using a game engine like Unity, Brick was developed natively on Android. In order to support augmentation, Bricks uses Google's ARCore technology [26]. For networking between devices, ARCore Cloud Anchors were used as it provides features that support synchronous multi-user networking.

One challenge was that each device would have a different coordinate space, which means that each player would see the same objects in different positions. This was solved by providing a custom network transform which would convert the position and rotation data relative to the ARCore Anchor, making the anchor behave as a world pivot, thus synchronizing the transforms of objects between players.

2.5 Software Design Patterns in Games Engineering

The Superhuman Sports framework is a complex framework that is composed of several APIs and Plugins that the system needs to use and manage them. With technology and APIs consistently changing and advancing everyday, the scalability and maintainability of such a complex system should be ensured in order to welcome future changes and improvements. When designing such a system, maintaining clean well-structured code-base is recommended by following some software design patterns in its development. Design patterns can provide abstraction and decoupling of the systems and improve the general readability and maintainability of the codebase. In his book, *Game Programming Patterns*, Nystorm R. talked about good software architecture designs and said "The measure of a design is how easily it accommodates changes" [27]. In this section an overview of some software design patterns that can be useful for the implementation of this research will be presented.

2.5.1 Component Pattern

The purpose of the Component Pattern [28] is to make a single entity span multiple domains. The code of each domain is placed in its own "component" class in order to keep domains isolated. The entity is then presented as a container of components. This pattern is the core concept behind the entity-component system which is used by game engines, like Unity. In Unity, for example, a game scene is structured as a set of entities, or *GameObjects*, and each entity would contain a set of components, normally sub-classes of the *MonoBehavior* class, that define the behavior for this entity. Structuring entities' behaviors in components make them decoupled and reusable among several entities. It also helps avoid building huge classes with several responsibilities and duplicating code. It is in the best practices to make each component follow the single responsibility paradigm, by making it responsible for a single behavior for the entity. Figure 2.14 shows a simple diagram for an entity-component system example, where there is a player and enemy entities, and they both share some of the same components.

2.5.2 Observer Pattern

The observer pattern is one of the most widely used and widely known of the original Gang of Four patterns [29]. The observer pattern [30] is the communication backbone of countless programs and frameworks, and its also commonly used with the Model-View-Controller architecture pattern. The Observer pattern consists of subject, or publisher, and observers.

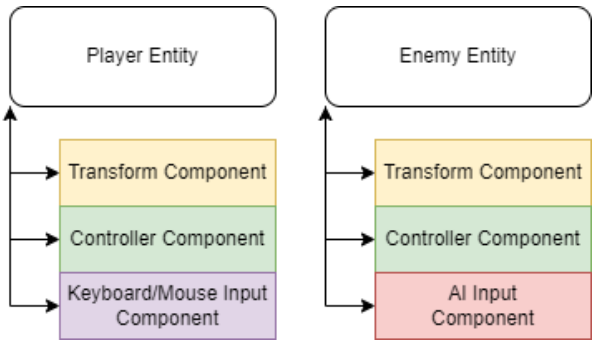


Figure 2.14: Example of an entity component system structure.

The subject maintains a list of observers that its will notify whenever a change happens in it state, which usually is caused by a method call. One example is when there is an achievement system in the game. Instead of having a class that consistently checks each frame if the player fulfilled the requirements of an achievement, it would be implemented as an observer class that listens, or subscribes, to a subject class that will notify the observer when the player fulfills the achievement's requirements. Figure 2.15 shows a simple example diagram for how a subject-observers relationship looks like.

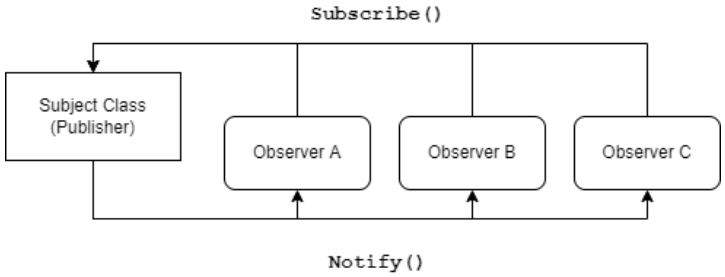


Figure 2.15: Example of subject-observer relationship in Observer pattern.

2.5.3 Singleton Pattern

Another popular pattern from the original Gang of Four patterns is the singleton pattern [31]. However, there is always a discussion about how this pattern should be used cautiously or even totally avoided. Nystorn R. said "despite noble intentions, the Singleton pattern described by the Gang of Four usually does more harm than good" [27]. Before presenting its potential issues and how to avoid them, the intent of the singleton pattern will be discussed first.

The aim of the singleton pattern is to restrict a class to one instance only and provide a global pointer to it to be accessed by any other class in the system. In some cases this is essential when a class can not perform correctly if it has more that one instance of it. A File

System could be an example for such a class, where it does file operations that should be done concurrently and must be able to coordinate with each other. Furthermore, the pattern provides a global static access which means any other class that needs to use the File System can have a direct call to its singleton instance. However, the complications of this pattern arises from this global access, as it is argued that this is not good for several reasons. These reasons are that (1) global static calls can be hard to debug in big code-bases as it hides the dependencies of the code, (2) it encourages coupling for developers who are not familiar with the code-base, and (3) it is not concurrency-friendly as when dealing with multi-threaded approaches, it can easily cause deadlocks, race conditions, and other thread-synchronization bugs.

There are several ways to avoid or reduce the potential problems of this pattern while maintaining its powerful benefits. One way is to use dependency injection techniques where the instance of the singleton object is injected in other classes that depend on it without providing a global access to that instance [32]. Another way is to have a global static class that can be responsible for creating the instances for classes that intend to be singletons and providing global functions for retrieving those instances. While this approach does not eliminate global classes, it certainly reduces their number. There are also other approaches that include using other patterns like the Service Locator pattern or the Subclass Sandbox pattern.

2.5.4 Command Pattern

The Command Pattern [33] is a behavioral design pattern that is commonly used in game systems. The Gang of Four book describes the Command Pattern as a pattern that "encapsulate a request as an object, thereby letting users parameterize clients with different requests, queue or log requests, and support undoable operations" [29]. To explain it in a different way, it is responsible for turning an action into a standalone object that contains all the information about that action, and by doing this transformation you can pass actions as method arguments, delay or queue an actions's execution, and support undoable operations. One of the common uses of this pattern is design input systems in games.

The implementation of this pattern usually involve the creation of a Command class that holds a virtual execution function. Then any action that is intended to be used in the application would be designed as a subclass of this Command class. The class would then hold member variables that include the parameters and dependencies required for this command, and the execution function would hold operations of the command which are usually done by a Receiver class that does the actual logic of the action. That way the command can be invoked by a Sender class which handles any form of input like a button click, keyboard input, controller input, or any other interaction. This pattern can also help with queuing or delaying the commands to have more control of when to execute them. Figure 2.16 shows the structure of this design pattern.

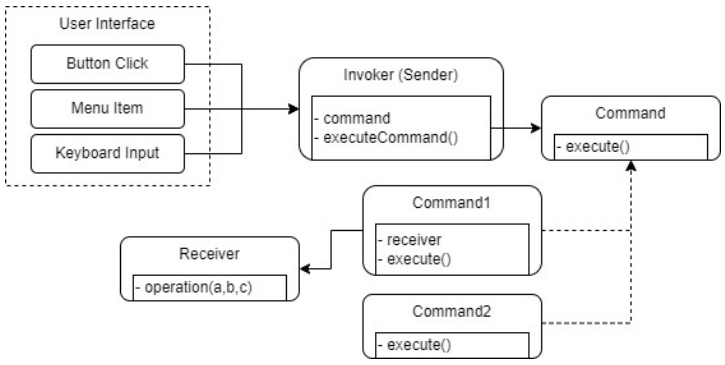


Figure 2.16: Design structure of Command Pattern.

2.6 Technologies

The technological advancements throughout several different fields of Game Engineering made the development of a complex systems, such as multiplayer AR games, possible. In this section, several state-of-the-art technologies that are essential for building these games will be discussed.

2.6.1 Game Engines

A game engine [34] can act as a foundation block for any video game, simulation, VR/AR experiences, or interactive applications in general. It provides all the needed components for the development of these applications, thus helping developers avoid creating all these components from scratch. Some of these components include: (1) 2D or 3D renderer, (2) physics simulator and collision detection system, (3) input and output management, (4) audio player and mixer, and (5) user interface. Furthermore, depending on the development needs more layers can be built above those component with the Game Logic right on the top, as seen in Figure 2.17.

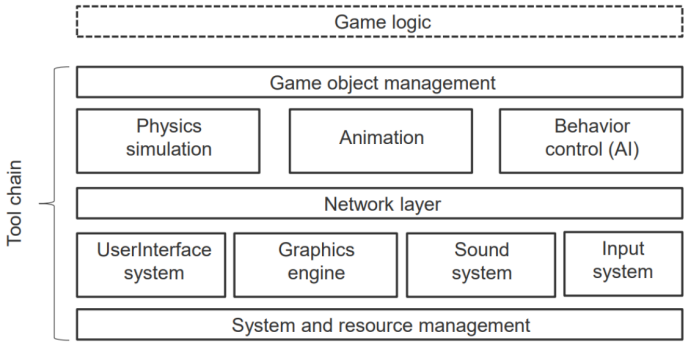


Figure 2.17: Overview of a game engine component structure.

There is a low-level approach of developing a game instead of a game engine. This approach depends on using an low or intermediate level programming language like C++ and graphics libraries like OpenGL, Vulkan, or DirectX. However, there are several game engines that are accessible and free for developers to use. These engines prove to be powerful and full of built-in features that make the development of games significantly easier. This sub-section will continue by providing some examples of these engines.

Unity

Unity is a cross-platform game engine developed by Unity Technologies [35]. It was first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X-exclusive game engine. As of 2018, the engine had been extended to support more than 25 platforms. As of 2021, Unity powers millions of games, including 71% of the top 1,000 mobile games and half of all PC and console games. More than 2.5 billion people play games with Unity, and around 4-5 billion devices have Unity-powered software on them [36].

Figure 2.18: Unity logo.

In addition to being one of the most common engines for developing games, Unity is also commonly used for developing AR applications. It provides AR foundation [37], which is multi-platform augmented reality framework (more about AR Foundation in subsection 2.6.3). There are also several networking frameworks that support Unity. Furthermore, Unity relies on object-oriented high-level programmings and its architecture is based on the entity-component system, which is an architectural pattern that defines every object found in the game scene as an entity object. All these features make Unity a suitable game engine to use for developing a multiplayer AR framework for superhuman sports.

Unreal Engine

Another popular cross-plateform game engine is Unreal Engine [38], which is developed by Epic Games. Unreal Engine was first showcased in 1998 in a first-person game called Unreal. It has since been widely used in the game industry for developing 3D games, and it was also adopted by other industries, such as film and television. As of 2015, Unreal Engine can be downloaded for free. It currently supports a wide range of platforms, including desktop, mobile, console, and VR/AR devices. This makes it another great tool that is accessible for developers and can be used to create high-quality games or simulations across several platforms.

Figure 2.19: Unreal Engine logo.

In addition to being used for platform-independent games, Unreal Engine can also be used for developing Augmented Reality applications. Similar to AR Foundation in Unity, the Unreal Engine AR framework [39] provides the rich tools needed in Unreal Engine to develop AR applications that work both for Android and iOS handheld devices. Furthermore, Unreal Engine also support several networking frameworks for multiplayer applications. This makes Unreal Engine a good candidate to be used for developing Multiplayer AR games for Superhuman Sports.

2.6.2 Hardware Devices

In order to use Augmented Reality applications, hardware devices with a complex composition of hardware modules are necessary. The devices should have the minimal necessary hardware components to run Augmented Reality and Networking solutions. When building a multiplayer AR game, there are several potential hardware devices to be considered as part of this system. Since the goal is to make the application platform-independent, several devices should be considered and evaluated in terms of the advantages and disadvantages they lay out. In this section, a review of the hardware devices that are potential components of AR Super Human Sports games will be presented.

Smartphones

Historically, smartphones are the first terminal devices to be used for commercial AR applications, with the Apple iPhone 3GS being one of the first. Later on, it was followed by Android tablets like the Samsung Galaxy Tab and Apple's iOS-based iPad. "We're evolving to a more visual and immersive web," says Jennifer Liu, director of product management at Google, "with the smartphone camera powering a new visual era of helpfulness" [40].

As of today, most smartphones are powerful enough to run Augmented Reality applications, with tablets having even more computing power and generally larger displays. Smartphone are wide-spread, always present with the end-user, and typical hosts for new AR applications. However, they are limited in their spatial interactions since in most cases users have to use a 2D touch screen as their input. They also lack the immersiveness needed in some Augmented Reality applications, like the superhuman sports for example, since the display portal is

Figure 2.20: AR application runninig on iPhone 3Gs.

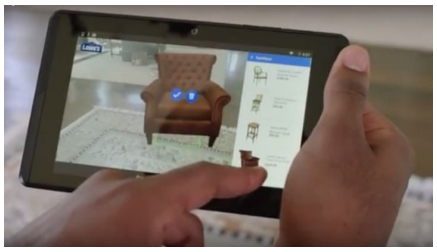


Figure 2.21: AR applicaion on Android tablet.

just the device's screen. However, there have been recently some setups lets users use their smartphones as HMDs like Google Cardboard [41], thus providing more immersiveness.

HMDs

The first three-dimensional Head Mounted Display was presented by Sutherland, Ivan E. in the fall joint computer conference in 1968 [42]. His paper is seen as a fundamental blueprint in the field of VR and three-dimensional HMDs. Ever since, there have been consistent research and development to produce HMDs for commercial use in VR and AR. With recent advancements, several OST (Optical-See-Through) AR HMDs have been introduced, including Microsoft's Hololens in 2015 and Magic Leap in 2016. These OST HMDs use advanced optical devices that displays augmented objects on see-through glasses to be aligned on real objects with respect to the user's view. So far, these HMDs provide the best immersiveness for AR applications with user input from hand-detection or spatially tracked controllers. Although these devices are not cheap and not commonly used in comparison to smartphones, they are consistently being adjusted to adapt to commercial use.

Figure 2.22: Microsoft Hololens Gen 1 AR HMD.

Figure 2.23: Magic Leap AR HMD.

Microcontrollers

One of the other components for an AR superhuman sports game could be a device with a microcontroller. It could be a drone or any other device that needs to communicates data and give feedback to the AR/VR devices in the system. For example, Sasaki, Tomoya, et al. proposes a Virtual Super-Leaping system that provides users with immersive virtual experience with the feeling of extreme jumping in the sky [43]. In this system, the player uses a VR headset and a haptic device with microcontroller that drives propeller units and gives feedback to the VR device, thus providing immersion (Figure 2.24). Another example, is as proposed by Eichhorn, C. et al. in the Catching the Drone Superhuman Sports game [44]. In the game, a drone-ball with microcontroller is used as a smart throwable ball that sends feedback to the AR devices of the players (Figure 2.25).

Figure 2.24: Virtual Super-Leaping with visual and haptic feedback.

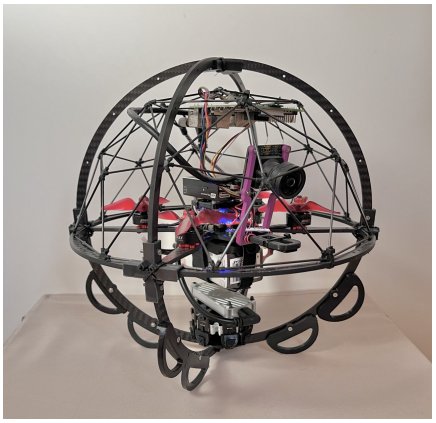


Figure 2.25: Droneball in Catching The Drone Super Human Sports game.

There are several microcontrollers that can be used for that purpose. Raspberry Pi [45] and Arduino [46] are among the most popular boards among hobbyist, electronics builders, and professionals. They both provide programmability for quick prototyping. The most notable difference between the two boards is that Raspberry Pi is a microprocessor mini computer which provide all the features of a computer with a processor, memory storage, graphics driver, and connectors on the board. On the other hand, the arduino board is a microcontroller that contains CPU, RAM, and ROM, as it focuses more on circuits prototyping.

2.6.3 AR Frameworks and Solutions

When building an Augmented Reality application, there are several problems that need to be solved. These problems can include plane tracking, object recognition, object position and orientation tracking, movement estimation in real-world, and much more problems. Through out the past years, these problems have been solved using complex algorithms and tools,

Figure 2.26: Raspberry Pi 4 board.

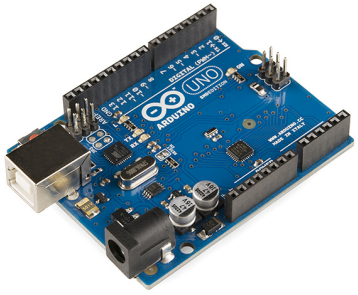


Figure 2.27: Arduino Uno board.

which are still constantly being improved by time. Several of these algorithms and tools have been encapsulated in AR frameworks that come in the form of development kits, APIs and plugins and aims to make the development cycle of an AR application much easier. In this section, some examples of such AR frameworks will be presented.

ARCore

ARCore is a software development kit developed by Google and used for building augmented reality applications on Android mobile devices [47]. It is widely used as it is available across several platforms, including Unity, Unreal Engine, Android, and iOS. Dave Burke, the Android engineer VP at Google, stated that ARCore targets 100 million Android devices after the preview, which represents 5% of the 2 million active Android users [48].

Figure 2.28: ARCore logo.

ARCore exploits the smartphone's camera and uses several APIs to provide three key features for augmenting virtual elements into the real world. The features include the following:

- **Motion Tracking.** It allows the detection of the device's position relevant to the real world.
- **Environmental understanding** It allows the device to detect different types of surfaces in the environment, such as vertical, horizontal, and angled surfaces.
- **Light estimation.** It provides the detection and estimation of the environment's current lighting conditions.

These powerful features render ARCore as one of the strongest augmented reality software development kits in the market. It can also be used totally free of charge, which makes it convenient for any person or business developing an AR application.

ARKit

Another popular software development kit that is used for building augmented reality applications is ARKit [49]. ARKit was developed by Apple and released in 2017 as an API tool for developers. It is designed to allow the developers create high-quality augmented reality apps by taking advantage of all of the sensors and cameras built into an Apple device (e.g. iPhone and iPad).

Figure 2.29: ARKit logo.

ARKit was designed to use SLAM (Simultaneous Localization and Mapping) [50] to scan and accurately map the surrounding world environment to be augmented. Moreover, it fully exploits the Apple device hardware to provide several powerful features. These features include the following:

- **TrueDepth Camera.** Using the TrueDepth Camera feature, the device can be used to accurately detect and track a face, along with its expressions, topology, and position.
- **Visual Inertial Odometry.** It is used to correctly read the surrounding environment. This reading along with data captured through the devices camera allows the device to accurately sense its movement in the room without any further calibration.
- **Scene understanding.** It allows for the detection of surfaces in the enviroment.
- **Light estimation.** It detects and estimates the current lighting conditions of the environment. This lighting data can be used to integrate virtual objects with more realism.

Several of these features are similar to its counterpart, ARCore. However, a study by Nowacki and Woda shows that ARKit has a subtle performance superiority in comparison to ARCore [51]. ARKit's performance surpasses ARCore's when it comes to the percentage of incorrectly determined planes (1,468% against 5,518%) and the time to detect the first plane (4,809% against 5,604%). Nevertheless, they explain this subtle difference is because of the Apple devices being more optimized than that of Android. Having strong performance, providing powerful features, and being free of charge make ARkit a competitive tool to be used for developing AR applications.

MRTK

MRTK, or Mixed Reality Tool Kit, is a cross-platform tool kit developed by Microsoft [18]. It is used to accelerate the development of Augment Reality and Virtual Reality applications across several platforms. It was first introduced as the HoloToolKit and was used for development on the Microsoft Hololens, but now is retargeted a more general cross-platform solution. It is also integrated in some game engines, including Unity and Unreal Engine. The features that the tool kit provides include the following:

- An input system and building blocks the provide several functions for spatial interactions and UI.
- In-editor simulation that helps the developer view their changes immediately, thus providing rapid prototyping.
- Flexibility in the framework components in which developers can specify the components they want to use or swap out ones they do not need.
- The support of a wide range of platforms and devices, as seen in Figure 2.30.

Platform	Supported Devices
OpenXR (Unity 2020.3.8+)	Microsoft HoloLens 2 Windows Mixed Reality headsets
Windows Mixed Reality	Microsoft HoloLens Microsoft HoloLens 2 Windows Mixed Reality headsets
Oculus (Unity 2019.3 or newer)	Oculus Quest
OpenVR	Windows Mixed Reality headsets HTC Vive Oculus Rift
Ultraleap Hand Tracking	Ultraleap Leap Motion controller
Mobile	iOS and Android

Figure 2.30: Platforms and devices supported by MRTK framework [18].

Unity's AR Foundation

Unity developed the AR Foundation framework which provides a modular augmented reality solution that works on multiple platforms [37]. AR Foundation does not provide an AR solution itself, but instead it is built above several native AR plugins, and it provides a common API for all the supported platforms of these plugins. The plugins that are supported by AR foundation are ones that are already supported by Unity and, as of 2021, they include ARCore for Android support, ARKit for iOS, MLTK (Magic Leap XR Plugin) for Magic Leap,

and MRTK (Windows XR Plugin) for Microsoft's Hololens.

AR foundation supports multiple powerful features for augmentation. These features include plane detection, point cloud generation, light estimation, raycasting, and much more. Figure 2.31 shows AR Foundation's current feature support for each plugin. Given its modularity and its multi-platform support, AR Foundation makes a great AR framework to be used for the development of AR applications that are designed to be cross-platform.

	ARCore	ARKit	Magic Leap	HoloLens
Device tracking	✓	✓	✓	✓
Plane tracking	✓	✓	✓	
Point clouds	✓	✓		
Anchors	✓	✓	✓	✓
Light estimation	✓	✓		
Environment probes	✓	✓		
Face tracking	✓	✓		
2D Image tracking	✓	✓	✓	
3D Object tracking		✓		
Meshing		✓	✓	✓
2D & 3D body tracking		✓		
Collaborative participants		✓		
Human segmentation		✓		
Raycast	✓	✓	✓	
Pass-through video	✓	✓		
Session management	✓	✓	✓	✓
Occlusion	✓	✓		

Figure 2.31: AR Foundation features support for each plugin [37].

Vuforia

Vuforia [24] is a software development kit used for developing Augmented Reality applications on mobile devices. It is one of most popular commercially used AR frameworks. It uses efficient computer vision algorithms to track 2D planner images as well as 3D objects in real-time. Developers can then use the tracked points in real images as a reference to position virtual 3D objects. Vuforia's most noteable features can be listed as follows:

- Tracking of a different 2D and 3D targets, including markerless images, 3D models, and a form of fiducial markers, also known as VuMark.
- Device localization in space with 6 degrees of freedom.

- Occlusion models which enables the masking of physical objects so that virtual objects would not be rendered in front of them.
- The ability to create and reconfigure target sets at runtime.

Additionally, Vuforia provides API that supports several different environments, including C++, Java, Objective-C and C#, and it also provides an extension in Unity. This renders it as an SDK that supports the development natively on iOS and Android as well as using a game engine, like Unity, and making it accessible and easy to use.

Figure 2.32: Vuforia logo.

OpenCV

OpenCV, or Open Computer Vision Library, is a library of programming functions that utilizes the CPU and GPU for real-time computer vision [52]. The library is free-to-use and supports cross-platform functionalities. Most of the current Augmented Reality Frameworks are built with OpenCV in their foundation layer. That being said, its possible to implement Augmented Reality solutions from scratch using OpenCV. Although this adds more complexity in the implementation, it provides several benefits in terms of tailoring and optimizing the computer vision algorithms to specific use cases that are relevant to the application.

Figure 2.33: OpenCV logo.

Unity Barracuda

Barracuda is an open-source package that can be used in Unity Engine to allow developers to run Neural Networks or Machine Learning algorithms in their Unity application. It is compatible with all Unity-supported platforms, including desktop, mobile, and consoles. Barracuda interfaces smoothly with Unity's rendering pipeline to allow quick and easy graphics and machine learning compatibility. Further more, the integration of Machine

Learning can open up several implementations in computer vision, like object detection or face recognition, and thus can be extended to provide augmented reality solutions.

2.6.4 Networking Frameworks

With the growing presence of online multiplayer games and real-time multi-user experiences, some networking frameworks were developed to make building such applications easier. These frameworks act as an API that provides functions for communication between devices. In this section, some of these frameworks will be presented and briefly discussed in the scope of this research.

Ubi-Interact

Ubi-Interact is a networking framework that is developed in the Technical University Munich and introduced by Weber S. et al. [53]. It is designed for interactive applications and provides real-time communication between devices that are distributed all over a network. Developers can use Ubi-Interact for building real-time systems that require multi-user communication with fast response times. Ubi-Interact provides several features that make it a prominent tool to be used when developing multiplayer interactive applications. These features include the following:

- It facilitates development of distributed network applications by providing the ability to combine and connect different systems and environments and enabling data exchange between them.
- It facilitates incorporating new devices by making the existing system adaptable to new hardware.
- It decouples system behaviour/interactions from specific devices used and makes I/O devices interchangeable if they use the same data.
- It provides code reusability between applications and different enviroments, and thus minimizing the cost of re-implementing working code when switching to different devices.
- It gives a good performance in the context of HCI, and thus making interactive tasks viable over the system.

Moreover, Ubi-Interact is built with a Publish/Subscribe architecture, where there is a publisher of topics that the other devices in the network can subscribe to. It uses Google protocol buffers to perform the data transport between devices, and in addition to its node javascript integration, it provides a Unity plugin for running clients on Unity applications, which makes integrating it to a Unity AR application feasible.

Mirror Networking

Mirror Networking is a high-level networking API that is built for Unity [54]. Mirror is intended to be an alternative to the now deprecated UNet framework, which was Unity's built-in networking solution. Mirror is built with the same concept as UNet but uses a client/server architecture instead. It provides several features, including the ability to switch between different transport layers (e.g. TCP, Sockets, etc.) and the ability to multicast in order to discover LAN games.

Figure 2.34: Mirror Networking logo.

Mirror is a free open-source solution with good documentation, and it is used commercially in several published games. However, it lacks a match making feature, which is common in other paid networking solution. Nevertheless, it provides a simple API which can be integrated with Unity and used for creating prototypes or proof-of-concept applications.

3 Implementation

The AR Superhuman Sports Platform (ARSSP) was presented by Ben Jazia [5] in his master thesis in 2020 at the Technical University in Munich. The ARSSP offers a solution to tackle the technical gap of developing AR Superhuman Sports games. It does so by providing a modular cross-platform framework that aims to abstract all the complex lower-level structures in order facilitate the development of AR Superhuman Sport games. The ARSSP will be used as a foundation for the implementation of this research, and it will be extended with modifications that further improve its usage.

In this chapter, the initial state of the ARSSP framework architecture will first be discussed. Then the extended features that aims to tackle the limitation of this framework will be introduced. These features include improvements in some existing functionalities in ARSSP as well as some completely new features in the platform. The last section will then present how developers can use this platform in order to implement their own AR Superhuman Sport games.

3.1 ARSSP Architecture

Software architecture works as a design for a system, abstracting its functional complexity and coordinating communication among its components. Its one of the most important pillars of software engineering and having a good modular architecture design is essential for building robust software applications. As seen in the literature review, frameworks like SARA [7], CARS [9], and the AR Flutter Plugin [4] all prove that modularity is important when implementing a complex framework. The architecture of the ARSSP framework is built on Unity game engine and uses the Unity API which uses an entity component system that utilizes the component pattern [33], as discussed in subsection 2.5.1. The framework then introduces several other layers that are built over the game engine and uses Unity's functions and the entity-component system. These layers consist of the following:

- **ARSSP Fixed Modules Layer:** The fixed modules layer consists of components that tackle basic difficulties and are found in almost all Superhuman Sports scenarios. It's easy to conceive of them as the foundations of any game. Examples of such components include the UI system and the gameloop, which are things that are associated with any game, not just superhuman sports.
- **ARSSP Interchangeable Modules Layer:** In contrast to the fixed modules layer, the interchangeable modules layer consists of components that are exchangeable depend-

ing on the configuration needed for the Superhuman Sports game being developed. This includes Networking and AR components that encapsulate over several different solutions.

- **ARSSP Interface Layer:** The interface layer has the highest level of abstraction in the architecture. It is composed of classes that interface and expose functions for the ARSSP developer that uses the other two layers and the game engine layer. Examples of these class are the singleton Managers used through out the application, like a Player Manager or AR Manager.

Furthermore, these layers can be visualized in a software architecture diagram in Figure 3.1.

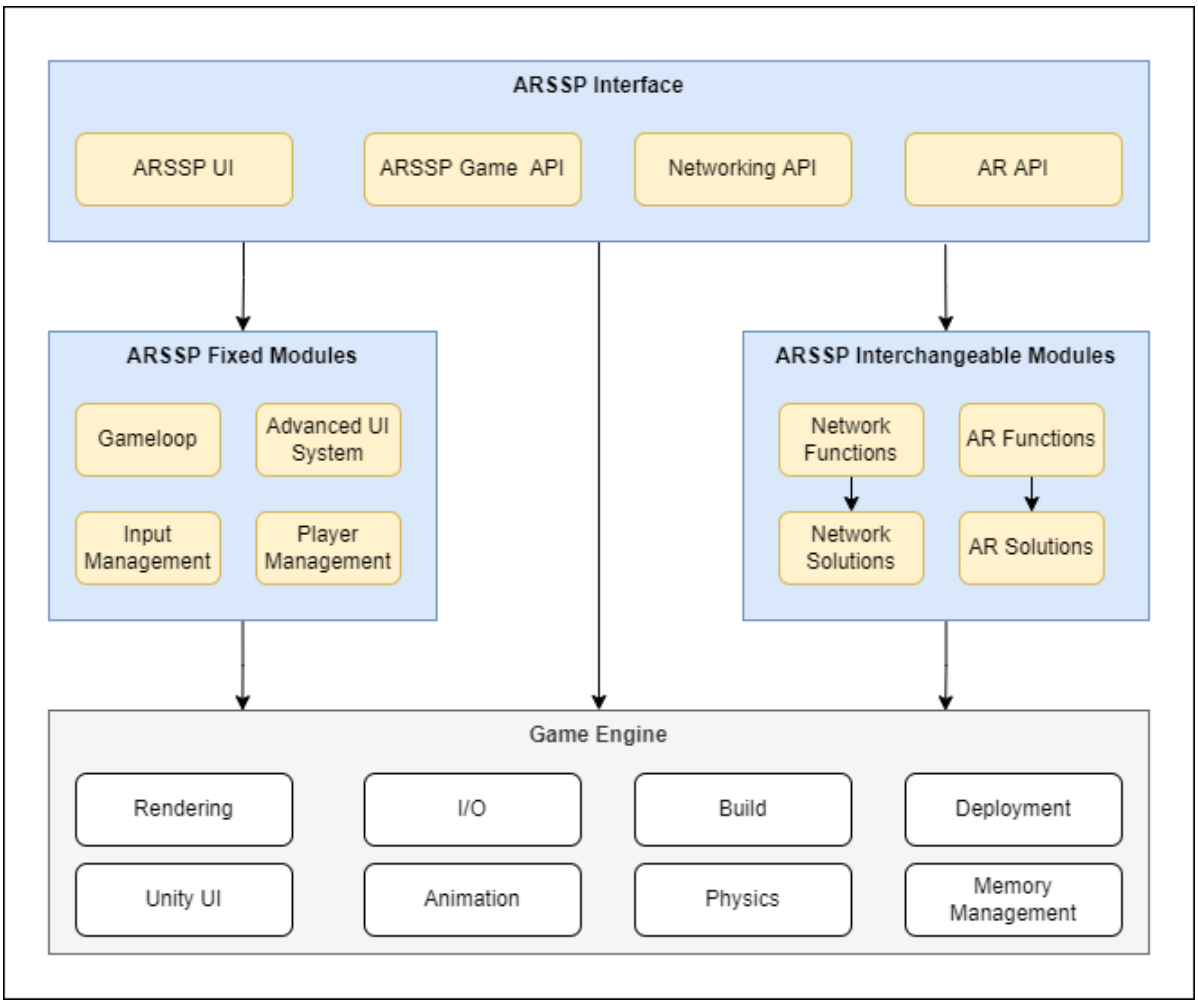


Figure 3.1: ARSSP Software Architecture.

3.2 ARSSP Frameworks

In this section, the different types of frameworks used in ARSSP will be discussed. These modular frameworks are used together to implement key functionalities in the ARSSP framework as a whole. All the classes in the frameworks are built over the `BBehavior` class, which is an extension of Unity's `MonoBehavior`.

3.2.1 AR Framework

The AR Framework is the essence of the ARSSP. It's in charge of offering basic and complex augmented reality capabilities by using existing solutions and frameworks. The AR Framework adheres to the platform's modularity concept by providing a layer of abstraction over the platform's useful core services. The AR Framework utilizes both middle-level (e.g. AR Foundation [37]) and low-level (e.g. ARCore [47] and OpenCV [52]) existing solutions for AR and CV problems, and it exposes an API that can be used by ARSSP developers to implement AR features without directly interacting with the middle-level and low-level parts. Figure 3.2 visualizes this approach.

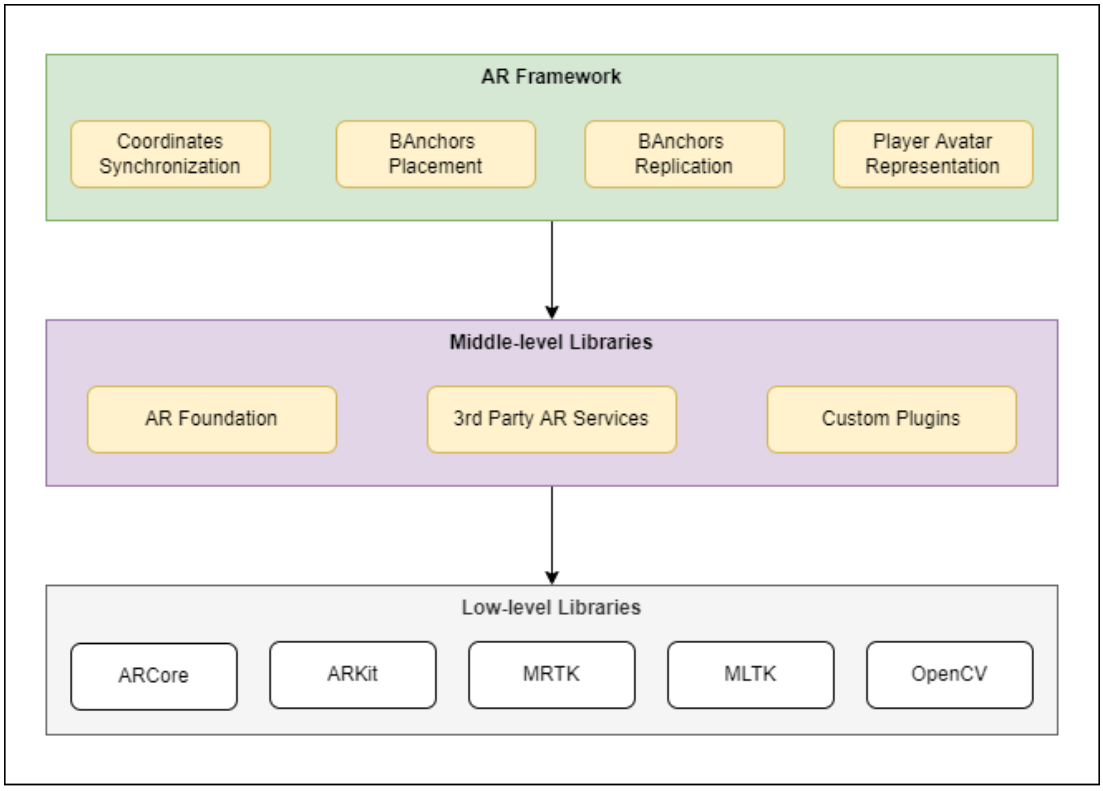


Figure 3.2: Diagram that visualizes the AR Framework abstraction of middle-level and low-level AR solutions.

The ability to augment virtual things in the real environment is a key aspect of AR, and there are several options for doing so. There are several existing algorithms that provide AR features like detecting physical markers, detecting planes, or estimating the device's pose. These features, as well as other complex ones, were developed and perfected by SDKs like ARCore [47] and ARKit [49]. This enables for far more realistic and immersive virtual content augmentation in the real world. These features are enveloped in the ARSSP framework in a system called BAnchor System.

A BAnchor primarily stores information about the position and orientation of a virtual object with respect to the coordinate system. It acts as a base class for any virtual object that exists in the augmented reality space. Furthermore, the BAnchor System provides several features that utilizes the BAnchors, including tracking a World root BAnchor which acts as the root of the coordinate system and replicating the position and orientation of the BAnchors throughout the network so that they would be synchronized between all clients. This helps make the ARSSP a multiplayer experience. It also utilizies networking features provided by the Networking Framework, which will be discussed in the following subsection.

3.2.2 Networking Framework (BEvent)

The Networking Framework, or the BEvent Framework, is considered to be the platform's foundational layer for communication. It offers a general communication system that links all of the modules and their components together. It allows for a networking framework to link several instances, regardless of their platform or operating system. In other words, it provides a generic means of communication which refers to how the system's architecture provides a degree of abstraction between the BEvent Framework and the backend networking solution (e.g. Ubi-Interact [53] and Mirror [54]), so it doesn't rely on a single data transmission protocol. This design architecture allows the implementation of more than one networking solution and abstracting it with one interface that makes the implementation modular and easy to modify.

The BEvent framework is composed of a set of classes, each with a responsibility. They aim to make the development process more structured and the implementation easier. These classes include the following:

- **BEvent:** The BEvent is meant to act as an alternative for the events in Unity's event system while providing a performance advantage, since it uses a native C# implementation [55]. It is the main implementation of events used both offline and online in the ARSSP, and it utilizes the observer pattern [30], as defined in subsection 2.5.2, which allows for callback functions to be binded to the BEvent.
- **BEventHandle:** The BEventHandle is a container that hold information intended to be transmitted with the BEvent. There is an `AbstractBEventHandle` class which is an abstract class that allows for different types of concrete implementations. The most commonly used implementation is to hold data types like `int`, `float`, or `string`.

- **BEventCollection:** The BEventCollection is a dedicated class that stores a list of all BEvents that are relevant to a specific application. In the BEventCollection, all the BEvents names as well as their associated BEventHandles are defined.
- **BEventDispatcher:** The BEventDispatcher is the main class responsible for firing the events and replicating them through out the network. There are several types of concrete dispatcher implementation depending on the networking solution and they are all subclasses to an `AbstractBEventDispatcher` class. For example, there is an `UbiiBEventDispatcher` which dispatches events using Ubi-Interact networking solution [53] and `MirrorBEventDispatcher` which uses Mirror [54].
- **BEventManager:** The BEventManager is a singletone class the manages the BEvent communication by keeping track of which type of BEventDispatcher is used and exposes networking related functions, like hosting/joining a game, invoking a BEvent, or switching between BEventDispatchers.

Furthermore, the diagram in Figure 3.3 demonstrates how these classes work with each other to communicate data in the BEvent framework.

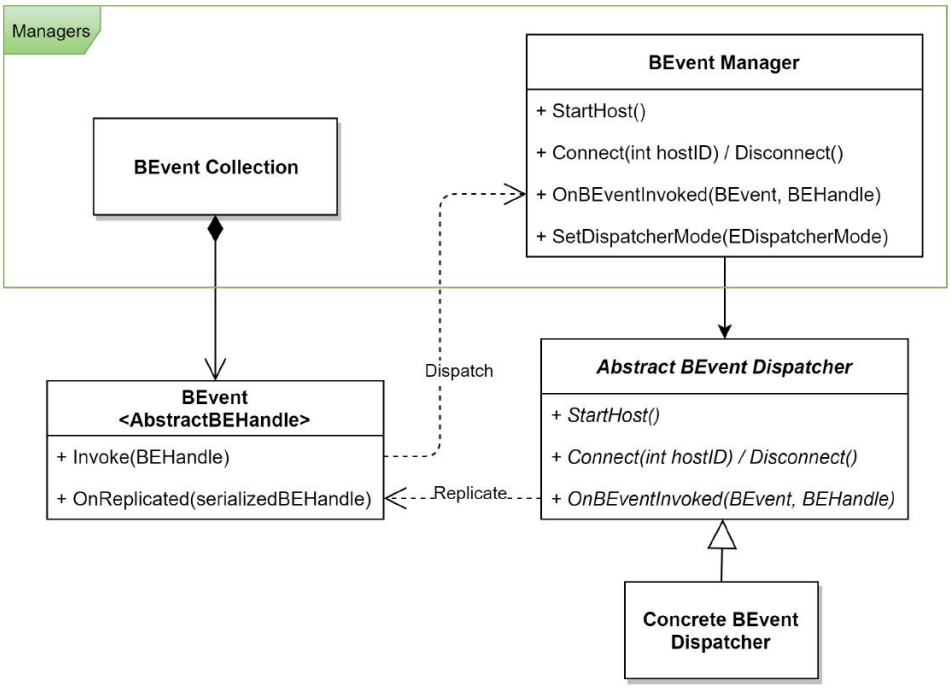


Figure 3.3: A UML Class diagram that represents how the BEvent Framework communicates data.

3.2.3 User Interface Framework

The User Interface (UI) Framework is built above Unity's UI System. Its aim to provide a more hierarchical approach with a unified structure for UI elements. The ARSSP's UI Framework also provides additional features above that of Unity's UI system. First, it adds a more substantial model for representing menus in games. It adds easy menu navigation and support for different types of inputs for selection. It also exposes more events for interaction with UI buttons. Furthermore, the UI Framework is built up of several elements of the type `BUIElement` class. These elements consist of the following:

- **BUIElement:** The BUIElement is the base abstract class that is the parent class for every UI element in the framework. It allows of recursively containing children and exposes some generic UI functionalities like hiding and showing the UI element.
- **BCanvas:** The BCanvas is a BUIElement that acts as a wrapper for the Unity's Canvas. It is the ground structure on the UIElements and it is at the top of the heirarchy. Every other BUIElement is expected to be contained inside a BCanvas.
- **BFrame:** The BFrame is a BUIElement that is designed to contain a collection of UI Elements that are based on a certain contextual structure. The BCanvas can have several BFrames as children and it manages them by having the focus on one BFrame at a time. This means that the BFrame that is in focus is the one that the user can interact with.
- **BMenu:** The BMenu a BUIElement that is at the core of the contextual design approach. Every BFrame needs to have at least one BMenu as a child. Only one BMenu in the BFrame can be shown at once and any other BMenu in the same BFrame will be hidden. This allows for easier navigation between different menus in the game.
- **BImage:** The BImage is a BUIElement that is an alternative for Unity's Image and Raw Image components. It provides a solution that encapsulates both the Image and Raw Image.
- **BText:** The BText is a BUIElement that acts as an alternative for Unity's Text UI elements. It encapsulates three different text rendering solutions, including Unity's Text component, TextMeshPro, and 3D TextMesh.
- **BButton:** The BButton is one of the most important BUIElements. It acts as a rigid and scalable alternative to Unity's UI Button. It provides navigation features to enable the user to easily navigate focus between buttons if they are using a controller input. It also exposes different types of events in the Unity Editor's inspector for interaction with the button.

Furthermore, Figure 3.4 demonstrates an example for a User Interface hierarchy using BUIElements.

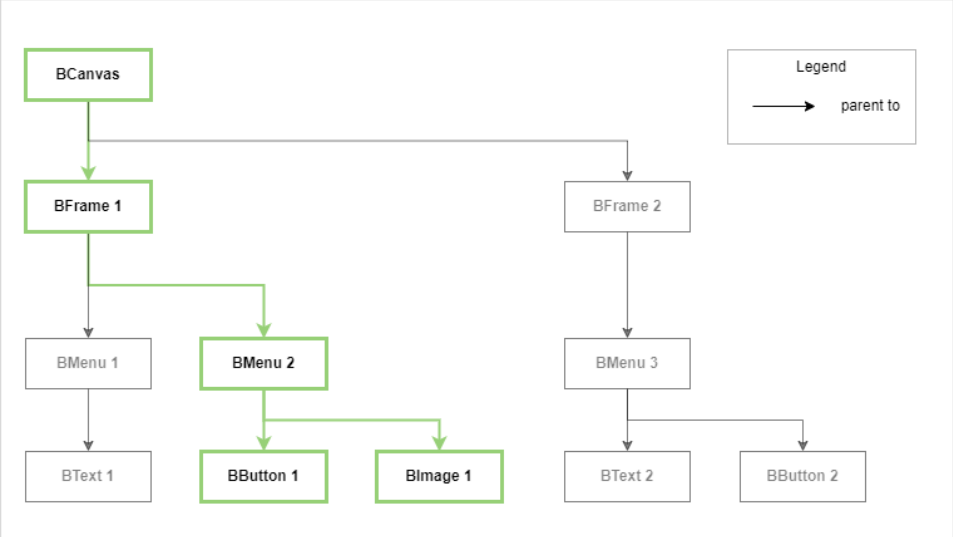


Figure 3.4: An example of a BUIElements hierarchy, with the green elements currently on focus.

3.2.4 Game Framework

The game framework is a basic structure that links all of the game logic's components. It's an important aspect of the gameloop's administration and construction. It focuses on components of the game that entail the user interacting with the system in a way that can change its state. The Game Framework's responsibility can be split into three major parts: Controllers Management, Players Management, and Game Modes & Player Stats. They are all guided by a singleton manager class `BGameManager`, and each can be defined as follows:

- **Controllers Management:** Provides a `BControllerManager` class that is responsible for managing and propagating any type of input to the rest of the system. It handles different input types like gamepads, mouses, AI controllers, and even replicated input through the network.
- **Players Management:** Provides a `BPlayerManager` class which is responsible adding players to the game session, assigning them with PlayerIDs, and associating them to existing ControllerIDs. It also responsible for tracking some extra parameters like the TeamID of the player and if they are ready or not.
- **Game Modes & Player Stats:** A Game Mode is responsible for dictating the rules of the game that is going to affect the game loop. It provides an abstract base class `AbstractGameMode` that can be used as a base to create concrete game modes. On the other hand, the `PlayerStats` is a structure that stores information about the player that is relevant to the state of the game.

Figure 3.5 and Figure 3.6 are diagrams that present the structure of the mentioned classes in the Game Framework of the ARSSP.

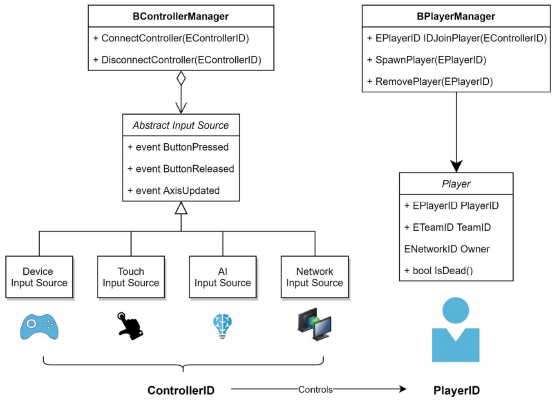


Figure 3.5: A UML diagram that demostrates the structure of BControllerManager and BPlayerManager.

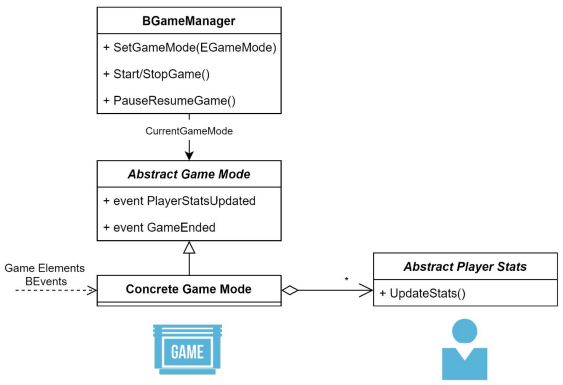


Figure 3.6: A UML diagram that demonstrates the structure of the game mode and player stats.

3.3 Extended Features

In this section, the extended features and modifications applied to the ARSSP in the scope of this research will be introduced. The goal is to add more functionalities and use cases to the ARSSP in order to make is a more useful tool for developing AR Superhuman Sports games. These features act as stepping stones to achieve this goal and to further close the gap between Superhuman Sports game design and the actual technical implementation of the game. They will also help realize a proof-of-concept game like Catching The Drone [44].

3.3.1 Game Engine Upgrade

One of the first modifications made to the ARSSP framework was an upgrade to the environment it was working on. This is essential as to make the framework more inline with the consistent advances in technologies of games engineering. This upgrade involved updating ARSSP's game engine layer, which was Unity 2019.4, to Unity 2020.3 with a Long-term Support (LTS). According to Unity Technology's website [56], Unity 2020 brings several improvements to the engine, which enhances the iteration and import times and improves the optimization tools in the engine. The most significant new Editor features in Unity 2020 include the following:

- Improvements to Package Manager which provide better user interface and better support for custom package workflows.
- Ability to see task progress and monitor sub-tasks easily.
- Ability to enter Prefab Mode and modify the prefabs with full context in a scene.
- Improvements to the profiling tool, which include GPU data availability, better visualization, and standalone profiler mode.

- Verified support for the new Unity Input System.
- Device Simulator to test how the application will run on several devices without leaving the Unity Editor.
- Improvements to the FBX importer with new axis conversing setting and custom properties for SketchUp objects.
- Focused Inspector where users can choose and compare game objects easily.

Furthermore, the upgrade was made to the game engine while ensuring that it does not break any of the initial functionalities and features of the framework. Thus the framework remains perfectly functional as intended after the upgrade.

3.3.2 Networking Frameworks Upgrade

Not only there was an upgrade to the game engine layer, but also the the networking plugins used in the ARSSP framework. The ARSSP was built with a networking layer that abstracts two different networking plugins, Ubi-Interact and Mirror. As mentioned in the Literature Review chapter, Ubi-Interact is a networking framework used for developing real-time distributed and reactive applications [53]. The framework is in constant improvement and since the development of the ARSSP, there have been several updates that improves Ubi-Interact framework's performance and usability. In order to accommodate these improvements in the ARSSP, the Ubi-Interact plugin was upgraded to its most recent update.

Furthermore, Mirror Networking plugin [54], which is a high level Networking library for Unity, also had several updates since the development of the ARSSP. The Mirror version initially used in the ARSSP was 10.4.7, which was supported for Unity 2018.3.6 and later versions. Since the Unity Engine was updated to 2020.3, it was only convenient to upgrade the Mirror plugin also to support the newer engine version. The newer version also included improvements in the performance and usability as well as several bug fixes.

3.3.3 Magic Leap HMD Support

When it comes to displaying an AR Superhuman Sports game, HMD devices do it best as they provide the immersion and flexibility required for such physical games. Moreover, several Superhuman Sports games that were reviewed, including VRabl [13], League of Lasers [15], and STAR [16], use HMD devices for immersion. In the initial state of the ARSSP framework, mobile phones - with Android and iOS operating system - were the only platforms supported for augmented reality. Even the proof-of-concept game, DropZone [5], previously designed for the ARSSP framework is implemented and tested only on Android smartphones. While maintaining Android and iOS mobile devices support is still intended, extending the ARSSP framework with Head-Mounted Display devices support is an essential step towards making the platform useful for Augmented Reality Superhuman Sports games.

Figure 3.7: Magic Leap logo.

To achieve this, the support for the Magic Leap Optical-See-Through HMD was added to the ARSSP framework. This process required only minor modifications in the framework, since Unity has a module that already supports building on Magic Leap's operating system - Lumin - and since AR Foundation is already compatible with the Magic Leap XR Plugin [37]. The process only required installing the Lumin SDK and adding the Magic Leap XR Plugin to the ARSSP framework. By doing so the framework can be used to implement and build Augmented Reality applications on the Magic Leap HMD device. Furthermore, as seen in Figure 2.31, AR Foundation supports several augmented reality features on the Magic Leap, including the following:

- **Device tracking:** device's position and orientation can be tracked in physical space.
- **Plane detection:** horizontal and vertical surfaces can be detected.
- **Anchors:** an arbitrary virtual position and orientation that can be tracked by the device.
- **Meshing:** triangle meshes can be generated to correspond the physical space.
- **2D image tracking:** 2D images can be detected and tracked by the device.
- **Raycast:** physical surroundings can be queried for detected planes and feature points.
- **Session management:** platform-level configuration can be manipulated automatically when AR Features are enable or disabled.

For the scope of this research, the 2D image tracking was implemented as an alternative for marker tracking. This was done using the example provided by the Magic Leap Unity documentation for image tracking [57]. An example can be seen in Figure 3.8.

3.3.4 World Space UI Interaction Support

Adding Magic Leap HMD Support only was not enough to provide an immersive experience in the ARSSP game. As mentioned, the ARSSP framework was first designed and tested on smartphones only, and that means that the User Interface elements in the framework are all designed to be fixed to an overlay canvas over the game screen with the touch screen as an input. However when using the same approach with HMD devices, it breaks the immersion of the game. The first reason is that having an overlay canvas on the game screen in the HMD will hinder the player's view of the virtual and real spaces. The second reason is that there will be

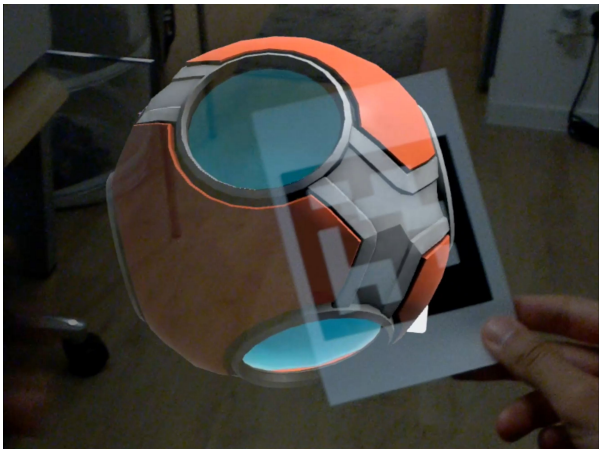


Figure 3.8: Magic Leap Image tracking detecting an ArUco marker and placing a 3D ball on it.

no way for the player to physically interact with the UI buttons, making it seem more artificial.

There are two ways to solve this problem. The first approach is to keep the UI in the screen space on an overlay canvas, but try to position the UI elements in boundary areas that will not hinder the player's view. In that case, the interaction will be using the Magic Leap's controller buttons to navigate between the UI elements and select them. The second approach would be to convert the screen-space UI into a world-space UI that is positioned in the virtual environment of the game scene. In other words the UI would be rendered in the game as virtual images in the 3D space. In that case, the user can interact with the virtual UI elements using the motion tracking of the Magic Leap controller by detecting a collision between the controller the UI element or casting a ray from the controller to the UI element. This approach can be seen used in the demos of reviewed AR Superhuman Sports games like VRabl [13] and League of Lasers [15].

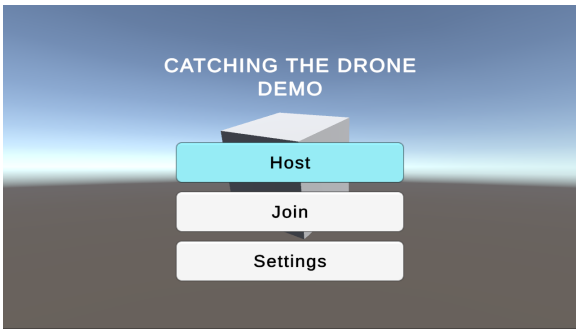


Figure 3.9: Example of Screen Space UI canvas on screen.

Figure 3.10: Example of World Space UI canvas in VR.

The latter approach was chosen to solve this problem, as the UI elements in that case would feel more integrated in the real world. Their visuals will seem realistic enough to make the player feel more present in the augmented environment and not break the immersion, but not too realistic as it will seem artificial enough to be differentiated from the real world. Furthermore, the interactions with UI elements will also feel more natural as this approach utilizes the controller's motion sensors to track the positioning of player's hand controller and thus detecting the player's natural interactions of, for example, pressing a UI button or pointing towards it.

In order to implement this, the approach has been first tested in the ARSSP framework environment on an example template scene in Unity. The first step was to convert the render mode UI canvas in the scene from 'Screen Space - Overlay' to 'World Space'. This change can be done in the Canvas component on the Canvas game object in the scene. Now that the UI canvas is in world space, it is recommended to adjust the transform of the canvas in order to be suitably positioned and scaled with respect to the camera. For further optimization, the camera can be referenced in the 'Event Camera' field in the Canvas component, but this can also be done dynamically in a script. The next step is to chose whether the UI canvas will be fixed in place or dynamically positioned to follow the view of the player.

Since a camera-following behavior would be more user-friendly as the player will not need to look around extensively to find the interactable UI elements, this behavior was implemented. A `CameraFollower` custom script was added and attached as a component to the Canvas game object. The `CameraFollower` class has to be a subclass of the Unity's `MonoBehavior` class (or `BBehavior` also in the ARSSP framework) so that it could be added as a component to a game object. The camera-following behavior is then implemented in the `LateUpdate()` function of the script which should look like this:

```
void LateUpdate()
{
    Camera mainCamera = Camera.main;
    transform.position = Vector3.Lerp(transform.position,
                          mainCamera.transform.position
                          + mainCamera.transform.forward
                          * offsetDistance,
                          Time.deltaTime * followSpeed);
    transform.LookAt(mainCamera.transform);
}
```

In this calculation, the position of the canvas is linearly interpolated to follow the main camera's position. It involves an `offsetDistance` variable, which represents the desired distance between the camera and the canvas, and the `followSpeed`, which represents how fast does the canvas follow the camera in the scene.

Since one of ARSSP's goals is to be a cross-platform framework, the amount work needed to make this UI modification should be minimized as much as possible. Afterall, the aim is to have one scene that can be built on several platforms without any effort for changes and modifications in the application. For that purpose, a script was created to automate the process of switching the UI canvas behavior depending on the current selected platform. In other words, the script would detect if the current platform is Lumin OS - Magic Leap's operating system - and do all the required changes to the UI canvas automatically. To achieve this, a `HMDWorldSpaceUI` script was created and attached as a component to the canvas game object. The script makes use of the pre-compiled if-condition statement `#if PLATFORM_LUMIN` in order to detect if the current Unity's building settings is set to Lumin OS.

In addition to adding the `LateUpdate()` above, a `Start()` function was also added. It will be responsible for setting the canvas render mode to World Space then resizing the canvas by scaling . By doing so, when the application runs on a Magic Leap build, the UI canvas will be automatically switched from the Screen Space render mode to the World Space render mode and scaled to the desire size. Furthermore, at each frame, the canvas will smoothly move to follow the camera's position in order to be always in view.

A World Space UI that is compatible the HMD view has been acheived. However, it is still only visual and UI elements are still not interactable. In most cases, some intertable UI elements are required, such as buttons. For that purpose, an interaction system was designed and implemented in order to give the player the ability to interact with UI elements that are on a World Space UI canvas while using the Magic Leap HMD. Since the Magic Leap provides motion controllers with its HMD device (Figure 3.11), it was used as the main source of input for the interaction system. Through that system, the player can use the Magic Leap controller to point towards the UI button and do a press button interaction using the "Bumper" button in the Magic Leap (labeled 2 in Figure 3.11).

Figure 3.11: Magic Leap motion controller.

In order to implement this interaction, the position and orientation of the controller need to be first tracked in Unity. This feature is already provided by the Magic Leap XR Plugin for Unity and it is encapsulated in the ARSSP framework in a prefab named `ML_Controller` that has to be present in the active scene of the game. After successfully tracking the transform of the controller, a Raycast needs to be calculated from the controller's position towards the forward direction of the controller's orientation, and then detect if the raycast collided with any interactable UI element. The raycast behavior can be implemented using Unity's builtin `Physics.Raycast` [58] call in the `Update()` function of a custom script called `ControllerBeam`.

The `RaycastHit` [59] object will hold all the information of the game object that collided with the raycast. However, the `Physics.Raycast` will only detect collision with objects that has a physics collider. For that reason, colliders need to be dynamically added to interactable UI elements that are present in the World Space UI canvas. Since the `BButton` is used as an extension to the Unity's UI Button, it can be used to be dynamically detected and obtain a collider. To implement this behavior, the `HMDWorldSpaceUI` script mentioned earlier can be used. Upon starting, the script should detect all the BButtons in its game object's children and add colliders to them. This can be implemented by adding the following code to the `Start()` function:

```
BButton[] bButtons = GetComponentsInChildren<BButton>();
foreach (BButton bB in bButtons)
{
    RectTransform rect = bB.GetComponent<RectTransform>();
    if (rect != null)
    {
        BoxCollider col = rect.gameObject.AddComponent<BoxCollider>();
        col.size = new Vector3(rect.sizeDelta.x, rect.sizeDelta.y + 10, 1);
    }
}
```

In this code snippet, the BButtons in the children are first fetched in an array. The script then iterates over the array in order to add a `BoxCollider` on the button and resize it to fit the BButton's dimensions. By doing so, the `Physics.Raycast` mentioned earlier will be able to detect when the controller's raycast collides with a BButton's collider in the World Space canvas. In that case, the `Physics.Raycast` in the `ControllerBeam` script can be updated to look like this:

```
void Update()
{
    RaycastHit hit;
    if (Physics.Raycast(transform.position, transform.forward, out hit))
    {
        selectedBButton = hit.transform.GetComponent<BButton>();
```

```
        if (selectedBButton != null)
        {
            selectedBButton.OnHoveredEnter();
        }
    }
    else
    {
        if (selectedBButton != null)
        {
            selectedBButton.OnHoveredExit();
            selectedBButton = null;
        }
    }
}
```

Here, the `selectedBButton` is a member variable that references the last button the raycast collided with. Furthermore, the script checks that the variable does not have a `null` value then calls the `OnHoveredEnter()` function on the BButton to visually display that it is currently selected. In case no collision is detected, the `selectedBButton` is reset by calling the `OnHoveredExit()` and then setting it back to `null`.

The next step is to detect the "Bumper" button press on the Magic Leap controller and simulate the pressing of the selected BButton. The Magic Leap XR Library provides an event delegate `OnButtonDown` that fires when a button is pressed on the controller. To implement the required behavior, an observer has to observe the `OnButtonDown`. This can be done creating a callback function and adding it as a listener to the event. This can be added as a listener to the `OnButtonDown` event of the controller. The callback function looks like this:

```
void OnButtonPressed(byte controller_id, MLInput.Controller.Button button)
{
    if (button == MLInput.Controller.Button.Bumper)
    {
        if (selectedBButton != null)
        {
            selectedBButton.OnPressed();
            selectedBButton.OnReleased(true);
        }
    }
}
```

In the `Start()`, a check is first done to ensure that the Magic Leap is running using the `MLInput.IsStarted` boolean. Then the controller is fetched and the function `OnButtonPressed` - as seen above - is added as a listener to the `OnButtonDown` event of the controller. That way, when the `OnButtonDown` event is fired, the `OnButtonPressed` function is called which

simulates the pressing and releasing of the selected BButton.

By achieving this, the interaction system is fully functional now and works as intended. one more additional thing that could be added is a visualization for the raycast so that the player would get the proper feedback for the direction their controller is pointing towards. This can be done by using Unity's `LineRenderer`. First the `LineRenderer` component needs to be added to the controller game object or the game object that hosts the `ControllerBeam` script. Afterwards the values of the `LineRenderer` needs to be set every frame in the `Update()` function of the `ControllerBeam`. This operation is done by fetching the `LineRenderer` component from the game object first, then setting the position of the first and second points of the line. In that case the first point would be in the position of the controller and the second point will be shifted towards the forward direction with a desired distance specified by the float `lineDistance`. This can be demonstrated in the lines bellow:

```
LineRenderer beamLine = GetComponent<LineRenderer>();
beamnLine.SetPosition(0, transform.position);
beamnLine.SetPosition(1, transform.position + transform.forward * lineDistance);
```

Furthermore, since this will run on each frame, the code can be optimized by caching the `LineRenderer` reference instead of calling `GetComponent<LineRenderer>()` each frame. An example of the HMD UI interaction system can be seen in Figure 3.12.

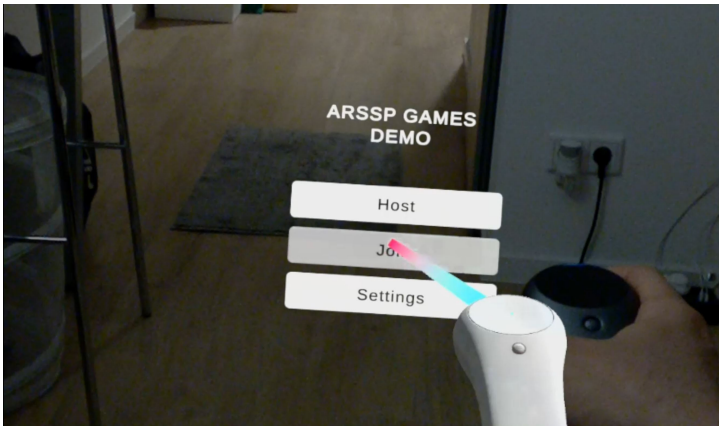


Figure 3.12: Magic Leap view of controller raycasting on a world space UI that is visualized by a line renderer.

3.3.5 Multiplayer Network Lobby System

The ARSSP framework provides a robust abstraction for the Networking layer that is used for making the AR Superhuman Sport games a multiplayer experience. As mentioned before,

ARSSP Networking layer encapsulates two different Networking solutions: Ubi-Interact and Mirror. This provides the ARSSP developer with the flexibility to switch between the two networking solutions while using the same interface. This interface provides several essential functions for a multiplayer game including hosting a game, joining a game, or broadcasting an event. However, what is missing is to further encapsulate these functions with the ARSSP's UI to provide a game lobby system that can be easily used by the player.

Game lobbies are menu screens that the players can view when joining a game session before the start of the game. In the game lobby, players can see other players who joined the same session as well as display, adjust the game settings, and - in some cases - communicate with other players. In multiplayer games, there are several approaches to enable the player to network and start a game lobby with other players. One approach is matchmaking, which allows the player to join a game session automatically with other players of similar game settings and - in most cases - similar level of stats. Another simpler approach is allowing the player to manually host a game or join another game through a screen that displays a list of all the game sessions currently hosted. This approach can also be seen in the demo of several multiplayer AR games - including Augmented Invaders [22] and Brick [25]. Furthermore, this approach is the method that is going to be used for implementing the Network Lobby System for the ARSSP.

The goal of implementing a Network Lobby System is to provide the players with a set of UI menus that make it easy to start a networked multiplayer session. Another goal is to implement all the basic multiplayer networking functions (like hosting, joining, etc.) by wrapping over the ARRSP Network Layer and providing a foundation that other ARSSP developers can use and extend with other functions relevant to AR Superhuman Sports game they are developing. Through this Network Lobby System, the player should be able to do the following:

- Host a game session.
- View available game sessions.
- Join a game session.
- Disconnect or leave a game session.
- Assign the player's team in the lobby.
- Assign if the player is ready or not in the lobby.
- Start the game if the player is the host and all the players are ready.
- Adjust networking settings (like switching between Ubi-Interact and Mirror APIs).

The system provides a set of UI BMenus - ARSSP's extended menu class that interfaces those functionalities for the user of the ARSSP game. In the following subsections, the UI and functions of each menu will be discussed.

Main Menu

The Main Menu is the first menu that is presented to the player - it is simple and straight to the point. It presents the user with a list of three UI buttons: Host Game, Join Game, and Settings buttons.

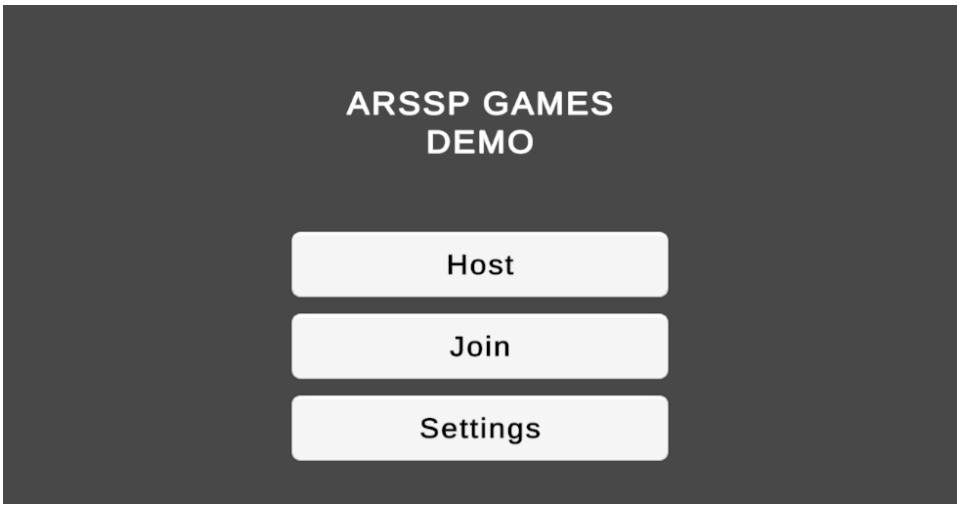


Figure 3.13: Main Menu is the ARSSP Demo Template.

- Host button: hosts a game session and navigates to the Lobby menu.
- Join button: navigates to the Host List Menu where the player can view all hosted sessions.
- Settings button: navigates to the Settings menu.

Since all the buttons' purpose is to navigate to another menu, this behavior is implemented by adding `MenuNavigationButton` component to the BButton game object and assigning the relevant BMenu in the component's "To B Menu" serialized field in Unity Editor's inspector. Each BButton in this BMenu contains this component with a reference to the BMenu it navigates to. Furthermore, the Host button has an additional functionality of actually hosting the game. This is done through the `StartHost()` function which is called in `BEventManager`'s singleton. This function is wrapped inside a script then added to a component that is in the Main Menu's BMenu game object and referenced as a callback function to the Button Release event of the Host BButton.

Host List Menu

In the Host List Menu, the player is presented with a list of all the hosted game sessions in which they can choose the game session they want to join. The BMenu contains a Scroll Rect,

Unity's scrollable UI panel, which is populated with a list of UI panels grouped in a vertical layout. Each UI panel represents a hosted game session. It also contains a text that shows the ID of the host and a Join button that allows the player to join this game session (Figure 3.14). Since this UI panel is reused, it has been packed in a prefab with a `HostViewUI` component that can be referenced by the script that will populate the list. Furthermore, the menu also has a Refresh button and back button. The Refresh button refreshes the host list and the Back button navigates back to the Main Menu.

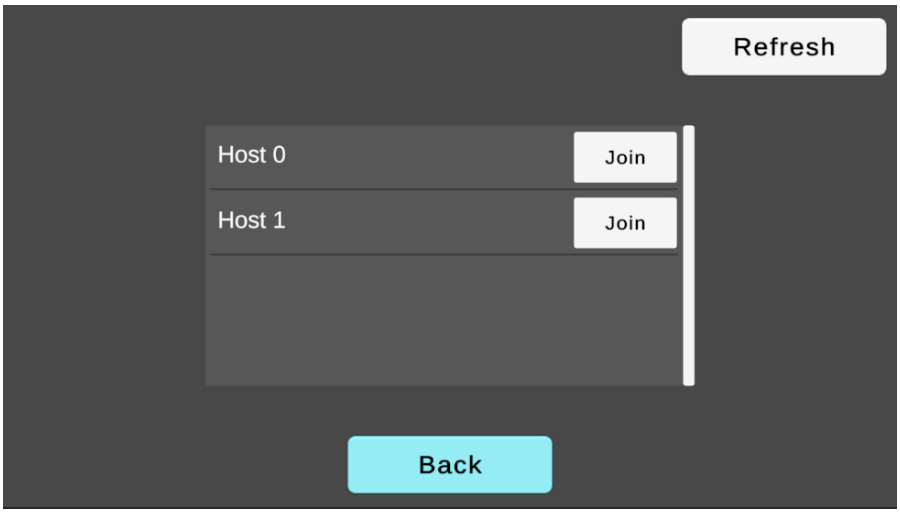


Figure 3.14: Host List Menu in the ARSSP Demo Template.

In order to populate the host list, a custom script was created and added as a component to the Host List BMenu game object. The script has a reference to the UI panel that represents the hosted game view and a reference to the Scroll Rect content transform parenting the host UI panels. It also has to dynamically add a callback function to each join button, allowing the player to join the relevant game session. The function should look like this in code:

```
public void UpdateHostList()
{
    // fetch all available hosts
    string[] hosts = BEventManager.Instance.GetAvailableHosts();

    // populates the scroll view with host view panels
    for(int i = 0; i < hosts.Length; i++)
    {
        HostViewUI hostView = Instantiate(hostViewPrefab, contentParent);
        int hostIndex = i;
        // create callback function for the Join button click
        System.Action joinHost = () => {
```

```
            uiElement.ParentBFrame.UpdateCurrentBMenu(lobbyMenu);
            BeventManager.Instance.ConnectToHost(hostIndex);
        };
        hostView.SetHostView("HOST " + i, joinHost);
    }
}
```

In the `UpdateHostList()` function, the content of the Scroll Rect needs to be cleared first. Then a list of available hosts is fetched from the `BEventManager` singleton instance using the `GetAvailableHosts()` function. Afterwards, the Scroll Rect is populated with the updated list of available hosts by iterating over the list and instantiating a `HostViewUI` prefab for each host. Furthermore, for each host, the ID text is set and a callback for the Join button click event is added.

Lobby Menu

The Lobby Menu is the menu responsible for presenting the player of the current state of the game session before the game starts. Through this menu, the player can view other players in the session, choose a team to be assigned to, toggle if ready, or start a hosted game. For that purpose, the menu contains a Scroll Rect, similar to the Host Menu's, populated with UI panels that represent each player (Figure 3.15). Each UI panel is encapsulated in a prefab and contains the player's ID, the team the player is assigned to, and whether the player is ready or not. The player is also presented with several buttons in the menu:

- Ready button: Toggles if the player is ready or not
- Switch Team button: Assigns player to a team.
- Refresh button: Repopulates the list of players in the session.
- Start button: Starts the game (only visible to the host of the game).
- Back button: Disconnects and leaves the session.

The functionality of each button is added to a custom script which is added as a component to the Lobby Menu game object. The functions are then each assigned as a callback to the release event of its respective button. The behavior of populating the Scroll Rect is also added to that script and called when the menu is shown or when the Refresh button is clicked. This behavior is implemented in a function called `UpdateLobbyPlayerList()`.

In the `UpdateLobbyPlayerList()` function, the content of the list in the Scroll Rect is first cleared, then a loop starts to iterate over the active players in the `PlayerManager` singleton instance. For each player a UI panel that represents the player is created and assigned with player name, team, and ready-state. Furthermore, the toggle Ready state functionality is implemented through a function called `ToggleReady()`.

Figure 3.15: Lobby Menu in the ARSSP Demo Template.

In the `ToggleReady()` function, the current player ID is fetched and then used to check if the player is ready or not using the `PartyStatusMap` in the `PlayerManager`. The ready-state of the player is then toggled and broadcasted as a BEvent. Upon doing so, the UI updates accordingly on all the users' screens throughout the network. Another functionality that can be presented is the assigning of team, which can be done through this function:

Similar to the previous function, this function first fetches the current player ID, then is used to get the team ID of the player. Afterwards, the team ID is incremented through an `enum` of team ID, `ETeamID`. It is then broadcasted over the network to be updated for the all the players in the session.

Settings

Since the ARSSP Network layer abstracts two different Networking frameworks - Ubi-Interact and Mirror - there has to be means to specify the Networking framework used in the application. It can be predefined before building the application or it can dynamically set during the run-time. While the ARSSP developer can already specify the networking solution they want to use before building the application, the Settings menu provides means for the player to switch between the solutions dynamically during run-time (Figure 3.16). This can be helpful for testing out the different networking solutions, or it can act as an example for how exactly the functionality is implemented.

The Settings Menu contains the following UI elements:

- A text element to show the current Network Dispatcher used.
- A toggle button to switch between the dispatchers.
- Text fields for specifying the IP and port used for the Ubi-Interact server connection.

- A back button to go back to the Main Menu.

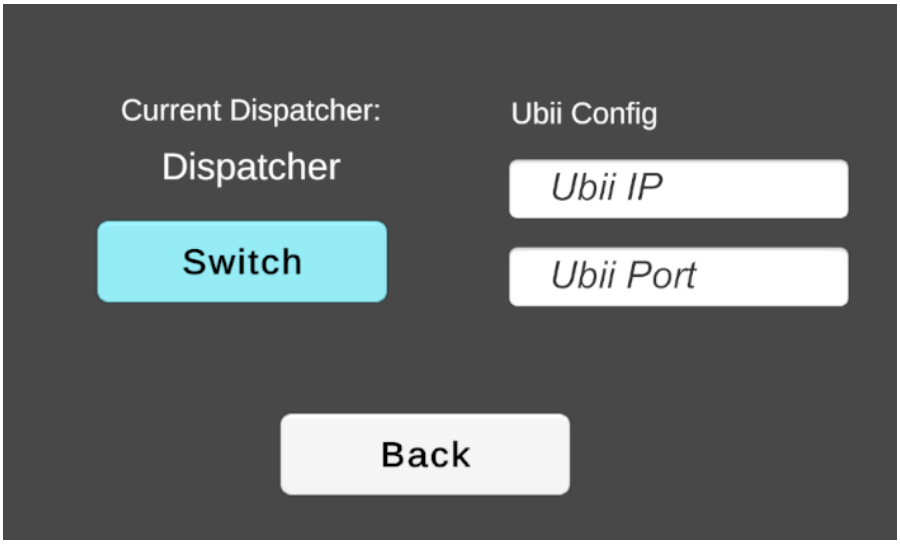


Figure 3.16: Settings Menu in the ARSSP Demo Template.

Given that, for example, the application is starting with the Mirror Networking dispatcher, the user can switch to Ubi-Interact dispatcher by first specifying the IP and port of the Ubi-Interact server in the text fields and then pressing the Switch button. The functionality of switching between dispatcher can be done by a function that looks like this:

In this function, the current event dispatcher is first checked by getting the type of the `BEventDispatcher` in the `BEventManager` singleton instance. The type of the `BEventDispatcher` is stored as an `enum`, so a switch operation is performed on its value. In case the result of the dispatcher type is `MIRROR` - representing the Mirror solution event dispatcher type - the dispatcher will be switched to the `UBI_INTERACT` type by calling `SetBEventDispatcher()` on the `BEventManager` singleton instance. On the other hand, if the current dispatcher type is `UBI_INTERACT` - representing the Ubi-Interact solution event disptacher - it will be switched to Mirror.

3.3.6 External Hardware Integration

The use of external hardware, like micro-controllers, is essential in the implementation of some of the AR Superhuman Sport games. For example, when realizing a game like Catching the Drone [44], a drone-ball that is controlled by a Raspberry Pi board is integrated in the game system. The game can also use micro-controllers that are embedded in the smart goals and send their signal through out the network of the game session. Another example is also with the Virtual Super-Leaping research [43] which uses external hardware to simulate the experience of extreme jumping in the sky. For that purpose, the integration of external micro-controller hardware was evaluated and implemented in the ARSSP system. Since Raspberry Pi [45], as seen in section 2.6.2, is a powerful compact computer that provides

powerful features, a relatively miniature size, and ease of use, and since it was also used in the drone-ball implementation of Catching the Drone, it proves to be a good example of external hardware to be integrated into the networking system of the ARSSP. Thus it was chosen for the implementation of this section.

The goal is to integrate a Raspberry Pi with the networking system of the ARSSP framework in order to allow it to communicate data with the other clients in the system. The challenge is to find an approach to run networking functionalities that are similar to that on the other clients running Unity. One direct approach was to find a way to run the modified headless (with no graphics) version of the Unity application on the Raspberry Pi so it would simulate another client in the system. This can be done by two approaches:

1. Build the Unity application on WebGL [60] and run it from a WebGL supported browser on the Raspberry Pi with a compatible operating system.
2. Install an Android-based operating system on the Raspberry Pi so that it would be able to run an Android build from the Unity application.

However, further research and testing proved these solutions to be insufficient as they are overly performance-intensive for a hardware like that of the Raspberry Pi, so the application will be tremendously slow and it will not perform as intended. This means that another approach should be explored.

The challenge now is to find a more tailored approach for the Raspberry Pi that would work efficiently with its hardware and operating system. The Raspberry Pi model used for this implementation is Raspberry Pi Zero W (Figure 3.17) running Raspbian, a Linux-based Raspberry Pi tailored operating system [61]. Since the Raspberry Pi's operating system is Linux-based, it supports several programming languages like, C/C++, Python 2/3, and Scratch. This operating system opens the window for several new approaches. One approach is to use an Ubi-Interact API to create a client application that will run on the Raspberry Pi. There are several advantages that make Ubi-Interact the most suitable choice, and they include the following:

- The Ubi-Interact solution provides a Python API which makes it compatible with the Raspbian operating system.
- Ubi-Interact solution is tailored to work for low-latency real-time applications like the ARSSP games.
- It is already running on the Unity client side so communication with the Raspberry Pi side can be easily enabled.

The Ubi-Interact Python framework, which was developed and documented by Weber S. and Schmidt M. [62], was used to get Ubi-Interact working on the Raspberry Pi device. First, the development environment on the Raspberry Pi needs to be set up. For that purpose, the

Figure 3.17: Raspberry Pi Zero W board.

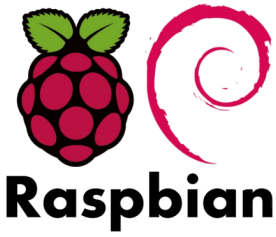


Figure 3.18: Raspbian OS logo.

Raspbian operating system was first installed on the Raspberry Pi through a micro SD card. The process is documented in the Raspberry Pi Documentation website [45]. Afterwards, the installation of all the requirements of the framework was ensured. The framework has the following software requirements for Linux operating system:

- Python version 3.7 or greater installed.
- Using Virtual Environment (`venv`) is recommended with Python package management system (`pip`) installed.

After ensuring that the mentioned requirements are met, the Ubi-Interact Python package can be installed using `pip`. This can be done through the following command in the terminal:

```
$ python -m pip install ubii-node-python[cli]
```

There is another approach to install Ubi-Interact Python package without using `pip`. It can be done through the following command:

```
$ git clone git@github.com:SandroWeber/ubii-node-python.git
$ cd ubii-node-python
$ <create and activate virtual environment>
$ pip install -e .[cli]
```

After ensuring that the package is installed, the following command line can be used to make sure that it is working correctly:

```
$ ubii-client --help
```

Now that everything is set, a Python example script can now be written in order to implement a simple Ubi-Interact client application that communicates with other clients in the network. For this application, the following features need to be implemented:

- Connecting to an Ubi-Interact server using a given URL (with IP and port).
- Subscribing to a Topic in the network.

- Publishing a message to a Topic in the network.

The first step done when writing the script was to make sure all the dependencies were added. For this, several import calls were done at the beginning of the script. They look like this:

```
import asyncio
import argparse
from ubii.framework.logging import parse_args
from ubii.node import connect_client, Subscriptions, Publish, DefaultProtocol
from codestare.async_utils.nursery import TaskNursery
```

After importing, the function `start_client()` is added. This function will be responsible for starting the connecting first to a server through a URL and then performing the subscribe and publish operations. First, the URL is parsed from the command call in the terminal when running this application. It should look like this:

```
def start_client():
    parser = argparse.ArgumentParser()
    parser.add_argument('--url', type=str, action='store', default=None, help='')
    args = parse_args(parser=parser)
```

The `parse_args()` function gets the default arguments as well as a URL from the command. After parsing the and arguments and URL input, an asynchronous function `run()` is created inside the `start_client()` function. This asynchronous function starts the server connection, subscribes to topics, and keeps publishing messages in a loop. The function looks like this:

```
loop = asyncio.get_event_loop_policy().get_event_loop()
async def run():
    client: UbiiClient[DefaultProtocol]
    async with connect_client(url=args.url) as client:
        constants = client.protocol.context.constants
        assert constants
        info_topic_pattern = constants.DEFAULT_TOPICS.INFO_TOPICS.REGEX_ALL_INFOS
        info, = await client[Subscriptions].subscribe_regex(info_topic_pattern)
        print(f"Subscribed to topic {info_topic_pattern!r}")
        info.register_callback(print)
        value = None
        while client.state != client.State.UNAVAILABLE:
            value = 'foo' if value == 'bar' else 'bar'
            await asyncio.sleep(1)
            await client[Publish].publish({'topic': '/info/custom_topic',
                                           'string': value})
```

```
    loop.stop()
```

This asynchronous function contains two parts, a part for subscribing to a topic and a part to publish messages to a topic. In the first part, the `info_topic_pattern` variable is used to hold the name of the topic, which in that case is the globally stored default topic name for infos. The client is then subscribed to the topic using `subscribe_regex` function which is called on `client[Subscriptions]`. A `print` callback is also registered to this subscription so that whenever a message is received from the subscribed topic, it gets printed out. The other part of the function consists of a loop that consistently publishes messages to a given topic. That is done using the `publish` function which is called on `client[Publish]`. This function takes a topic name and the message that is going to be sent. In that case, the client is going to consistently publish the string `'foo'` and `'bar'` to the `'/info/custom\_topic'` topic.

Afterward, the asynchronous function needs to start and run forever. This can be done using this code inside the `start_client()` function:

```
nursery = TaskNursery(name="__main__", loop=loop)
nursery.create_task(run())
loop.run_forever()
assert not loop.is_running()
loop.close()
```

Finally, the `start_client()` function needs to be called in a main function like this:

```
if __name__ == "__main__":
    start_client()
```

Now that the example script is finished, it can be run from the command line with the URL of the Ubi-Interact server as a parameter. It is recommended to do this after running a python virtual environment. The command for running the python script should look like this:

```
$ python3 example.py --url https://192.168.0.6:8101
```

In this command the URL is specified after `--url`. In this example, the IP is `192.168.0.6` and the port is `8101`. This URL should be adjusted according to the IP and port of the server that is the running the Ubi-Interact master node.

Now that the example is functional, it can be further adjusted to to be integrated in the ARSSP game. Note that this example only servers as a guide on how to use Ubi-Interact API functionalities on the Raspberry Pi. In order to integrate the Raspberry Pi with the ARSSP framework, the topics used for publishing and subscribing needs to be adjusted accordingly. Furthermore, this script can be used to fire BEvents in the ARSSP network. This can be done

be first setting the publish topic to a relevant topic, like RAW_BEVENT, which should look like this:

```
bevent_topic = 'RAW_BEVENT'
```

Furthermore, in order to fire a BEvent, the BEvent data can be written in a JSON format and sent as a string with the publish message. It should look like this:

```
jsonBEvent = '{"$id": "1",
           "InvokingBEventName": "TEST_Vector3Test",
           "DebugEvent": true,
           "Arg1": {"$id": "2", "x": 5.045, "y": -3.24533, "z": 704.7499} }'
```

This JSON string encapsulates a TEST_Vector3Test BEvent where its name is assigned under the "InvokingBEventName" property. The arguments of this BEvent is assigned under the "Arg1" property, which in this case contains a Vector3 struct with values for the x, y, and z properties. This is a general JSON format to be used when firing a BEevent through the network from the script. Another example is for a TEST_FloatTest BEVent that looks like this:

```
jsonBEvent = '{"$id": "1",
           "InvokingBEventName": "TEST_FloatTest",
           "DebugEvent": true,
           "Arg1": 3.14 }'
```

Furthermore, publishing the BEvent message to its relevant topic should look like this in the script:

```
client[Publish].publish({'topic': bevent_topic, 'string': jsonBEvent})
```

By doing so, the Raspberry Pi can communicate with other Unity Ubi-Interact clients in the ARSSP game by firing BEvents and replicating them throughout the network.

3.3.7 OpenCV Support

One of the features in a Superhuman Sports game like Catching the Drone [44] is that the players should be able to detect the drone-ball through their devices' cameras. This happens by detecting a color pattern in the LEDs of the drone-ball (Figure 3.19). The detection of the pattern is done through OpenCV [52]. Thus, it is one of the goals to extend the ARSSP framework with OpenCV to allow such feature to be implemented within the system. The goal, however, is not to implement this exact OpenCV application but rather on extending the framework to allow for any OpenCV implementation to be easily integrated in the ARSSP framework.

Figure 3.19: Drone-ball with LEDs.

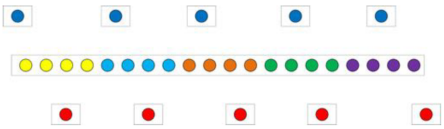


Figure 3.20: LED color pattern to be detected.

In order to extend the ARSSP framework with OpenCV support, an OpenCV plugin was added in the Unity project of the ARSSP framework. The plugin used in the "OpenCV plus Unity" plugin [63] can be obtained from Unity's Asset Store website or from the Package Manager in the Unity Editor. This package uses `OpenCVSharp` - an open source C# port of OpenCV - and provides an out-of-the-box integration of OpenCV within the Unity Engine. It also provides several example features including a face detector, ArUco marker detector, and alphabet characters detector. For this section, the integration of the ArUco marker detector application will be used as an example.

An ArUco marker is a synthetic square image with a broad black border and an inner binary matrix that defines its identifier/ID (Figure 3.21). The image's black border makes it easier to be detected by computer vision algorithms, while the binary coding makes it possible to identify it and apply error detection or correction procedures to it. The internal matrix's size is determined by the marker size. A 4x4 marker, for example, is made up of 16 bits. An ArUco markers is usually used for marker-detection algorithms to identify it and determine its orientation so that 3D objects can be augmented and overlayed on it. For this application, the goal is to be able to detect ArUco markers' poses in real-time and augment 3D objects on them within the ARSSP framework.

After importing the "OpenCV plus Unity" plugin, the ArUco marker detector can be implemented in a `MonoBehaviour` script using the `CvAruco` class and the `OpenCVSharp` library. The following is a code example that shows the functions that can be used in order to detect the ArUco markers:

```
// Convert image to grasyscale
Mat grayMat = new Mat ();
Cv2.CvtColor (mat, grayMat, ColorConversionCodes.BGR2GRAY);
```

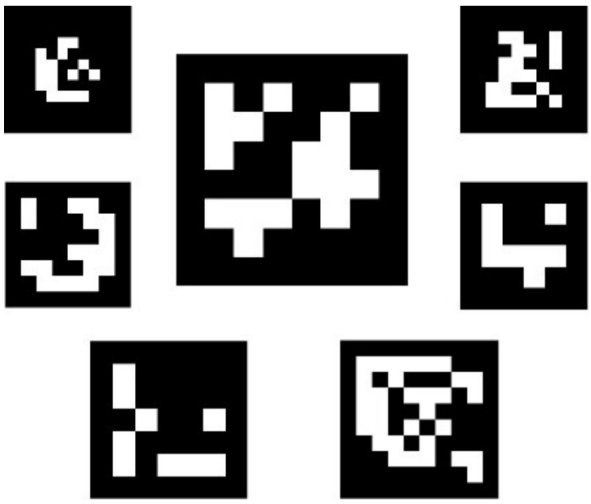
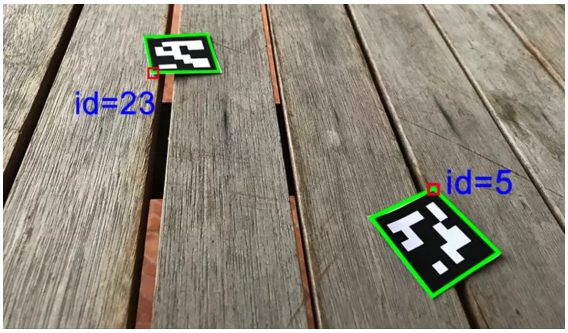


Figure 3.21: ArUco markers example.

```
// Detect and draw markers
CvAruco.DetectMarkers (grayMat, dictionary, out corners, out ids,
                       detectorParameters, out rejectedImgPoints);
CvAruco.DrawDetectedMarkers (mat, corners, ids);
```

This example `CvAruco` and `OpenCVSharp` functions were used to detect ArUco markers and display the results. Figure 3.22 shows an example image and the result after running the ArUco marker detection function on it.

(a) The input image.



(b) The result image.

Figure 3.22: The input and result iamges of the ArUco marker detector.

Furthermore, this functionality needs to be extended in order to get the detected marker's pose and augment a virtual 3D object on it with respect to the player's view. This will require access to the device's camera feed as well as extending the feature to overlay the 3D object on the detected marker in real-time. This functionality can be done through the `ARMarkerTracker`

script which was provided by the AR Marker Detector package available for free at the Unity Asset Store [64]. This script uses images from the device's camera input, detects the ArUco markers using the same detection techniques as mentioned before, and extends this feature by overlaying the resulted image with a 3D model positioned with respect to the pose of the detected markers. A screenshot of an example running on an Android mobile device can be seen in Figure 3.23

Figure 3.23: Screenshot of AR Marker Tracker example running on android mobile device.

3.4 ARSSP Game Template

After extending the ARSSP with the features mentioned in the previous section, these features were all integrated and used in creating a new ARSSP game template that can be used to develop Superhuman Sport games. The goal of this template is to make it easier for ARSSP developers to get a head-start in using the framework to develop their Superhuman Sport games. This template comes in a form of a Unity package that contains a template scene as well as some prefabs. In this section, the structure of the template will be presented as well as a basic guide for its use.

3.4.1 Managers Spawner

The Managers Spawner is a game object in the ARSSP template scene containing a `ManagersSpawner` script. This script is a global script that can be used to configure the main settings of the ARSSP game. These settings include: which AR solution to be used (e.g. AR Foundation or OpenCV), which Networking solution to be used (e.g. Ubi-Interact or Mirror), or which game mode to be used. This script provides a user-friendly interface in the inspector for modifying this configuration with the help of Odin Inspector Unity plugin [65]. The settings that can be configured through the inspector of the script are the following:

- Starting scene of the application.

- Target frame rate of the application.
- Networking events dispatcher type to be used for networking in the game.
- The AR framework to be used for augmentation in the game.
- Input settings configuration.
- Mode of the game.
- Player settings configuration.
- Debug settings configuration.

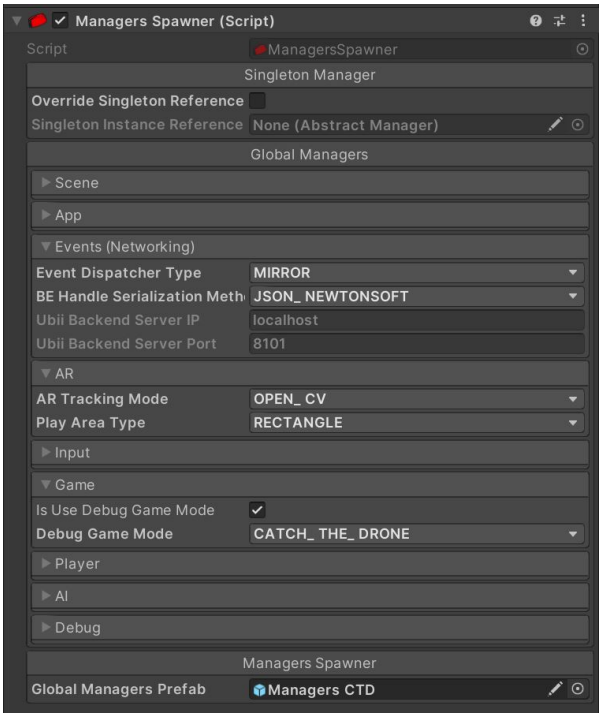


Figure 3.24: Inspector view of the Managers Spawner script in the Unity Editor.

3.4.2 Global Managers Prefab

Upon running the application, the `ManagersSpawner` will instantiate in the scene a Global Manager Prefab, which is referenced in the ManagersSpawner's inspector. The configuration set in the `ManagersSpawner` is then transferred to the `GlobalManagers` script which is in the parent game object of the prefab. This script references the children of this prefab which contains a collection of game objects that have manager scripts running globally throughout the game. These manager scripts follow the singleton pattern [31] - as seen in subsection 2.5.3. The managers in the children game objects are the following:

- **ARManager:** Contains the `ARManager` script, responsible for abstracting over AR solutions and exposing of functions to be used for the AR features. This game object also parents several other game objects that contain augmentation scripts - including AR Foundation and OpenCV scripts.
- **AppStateManager:** Contains the `AppStateManager` script which is responsible for handling and tracking the global state all over the application run-period.
- **AppSceneManager:** Contains the `AppSceneManager` script which is responsible for handling game scenes and navigating between them.
- **DebugManager:** Contains the `DebugManager` script which is responsible for displaying debug messages on canvas or in the form of notifications.
- **EventManager:** Contains the `BEventManager` script which controls and abstracts the event dispatchers used for networking. The game object also contains the `BaseBEventsCollection` script holding all available events and their parameters. Developers can add their custom BEvents Collection scripts there by creating a script that is a subclass of `AbstractBEventsCollection` and adding it as a component to this game object.
- **GameManager:** Contains the `GameManager` script which is responsible for tracking the game status and exposing game-loop-related functions like starting, pausing, or ending the game.
- **InputManager:** Contains the `InputManager` script which is responsible for tracking input sources and binding input buttons to game functions. It contains other scripts, too, that work with the `InputManager` to handle different types of input sources.
- **PlayerManager:** Contains the `PlayerManager` script which tracks all the the players in the game session and spawns the Player game objects from a referenced Player prefab (more on this topic in the next sub-section). The game object also contains `PlayerNetworkManager` script which is responsible for broadcast and received-player-related network events.
- **SoundManager:** Contains the `SoundManager` script which is responsible for managing the sound effects in the game.
- **UIManager:** Contains the `UIManager` script which is responsible for keeping track of the UI display state.
- **CustomManager:** This is an extra game object that can contain any custom manager or script that is only relevant to the game being developed.

The ARSSP developers can use this Global Managers prefab as a template and create their own variant of the prefab and have the scripts and configuration adjusted to the needs of their game. A diagram of the structure can be seen in Figure 3.25.

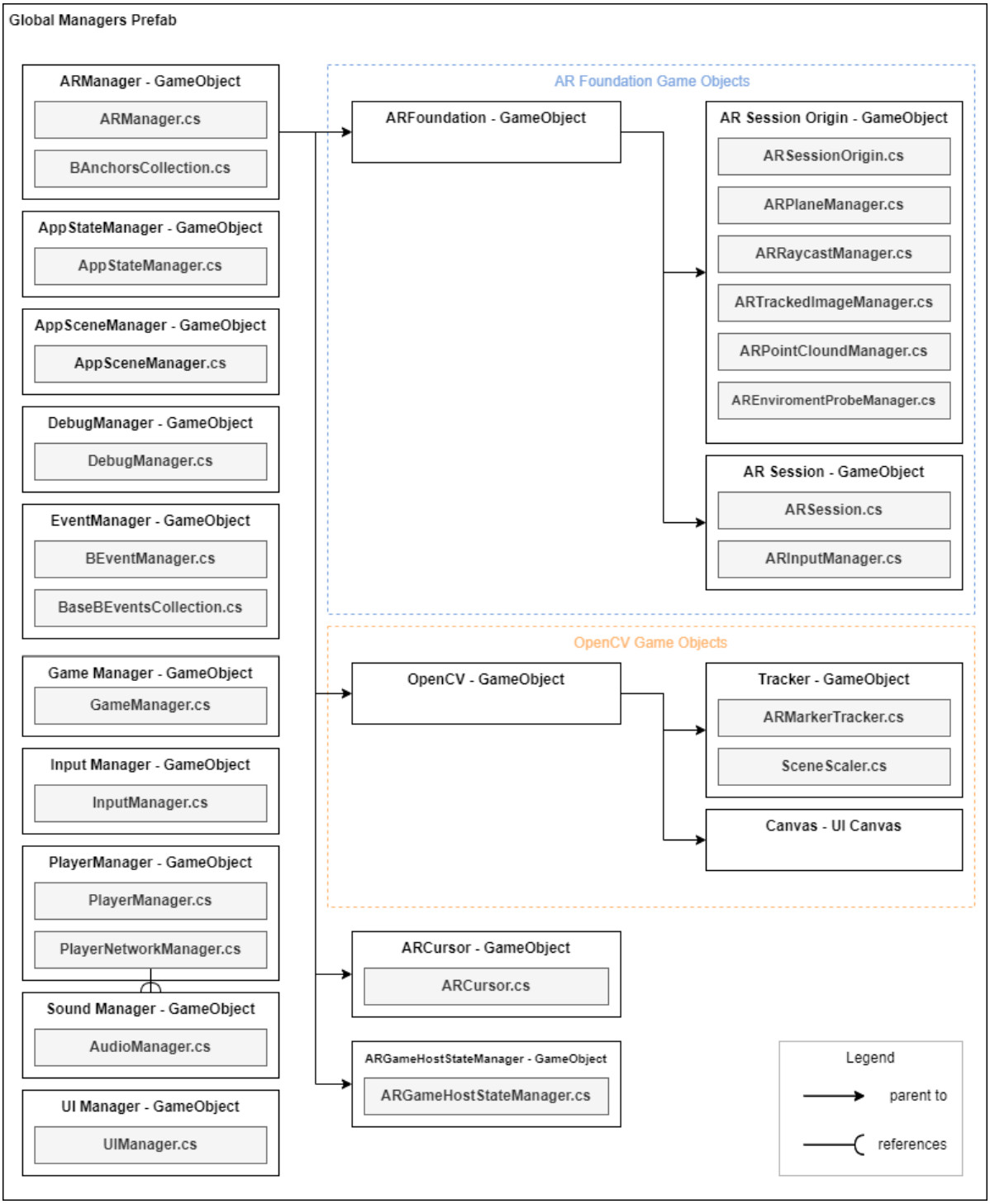


Figure 3.25: Diagram for the structure of the Global Managers prefab.

3.4.3 Player Prefab

The Player Prefab is a prefab that contains the game object representing a player in the game. This game object contains several scripts that are essential for simulating the player in a networked game. Moreover, it contains a custom Player script that is a subclass of `AbstractPlayer` class which holds main information about the player, like the `PlayerID`, `NetworkID`, `Position`, and `Rotation`. It also contains a `PlayerAvatarBAnchor` script, responsible for anchoring the player in the augmented reality space, and a `BAnchorReplication` script which replicates this position all over the network so that it would be synchronized for all the players.

3.4.4 Template Scene

The Template Scene is the main game scene that will run for the ARSSP game. This scene can be used as a guide for building the main scene of the ARSSP game. It includes several game objects essential for running the ARSSP game. It holds, furthermore, the Managers Spawner game object which contains the `ManagerSpawner` script mentioned in subsection 3.4.1. It also has a basic UI canvas containing the main menu as well as the networking lobby menus. Moreover, it has some other game objects that are relevant only to the Magic Leap build of the game. The structure of the scene can be seen in Figure 3.26.

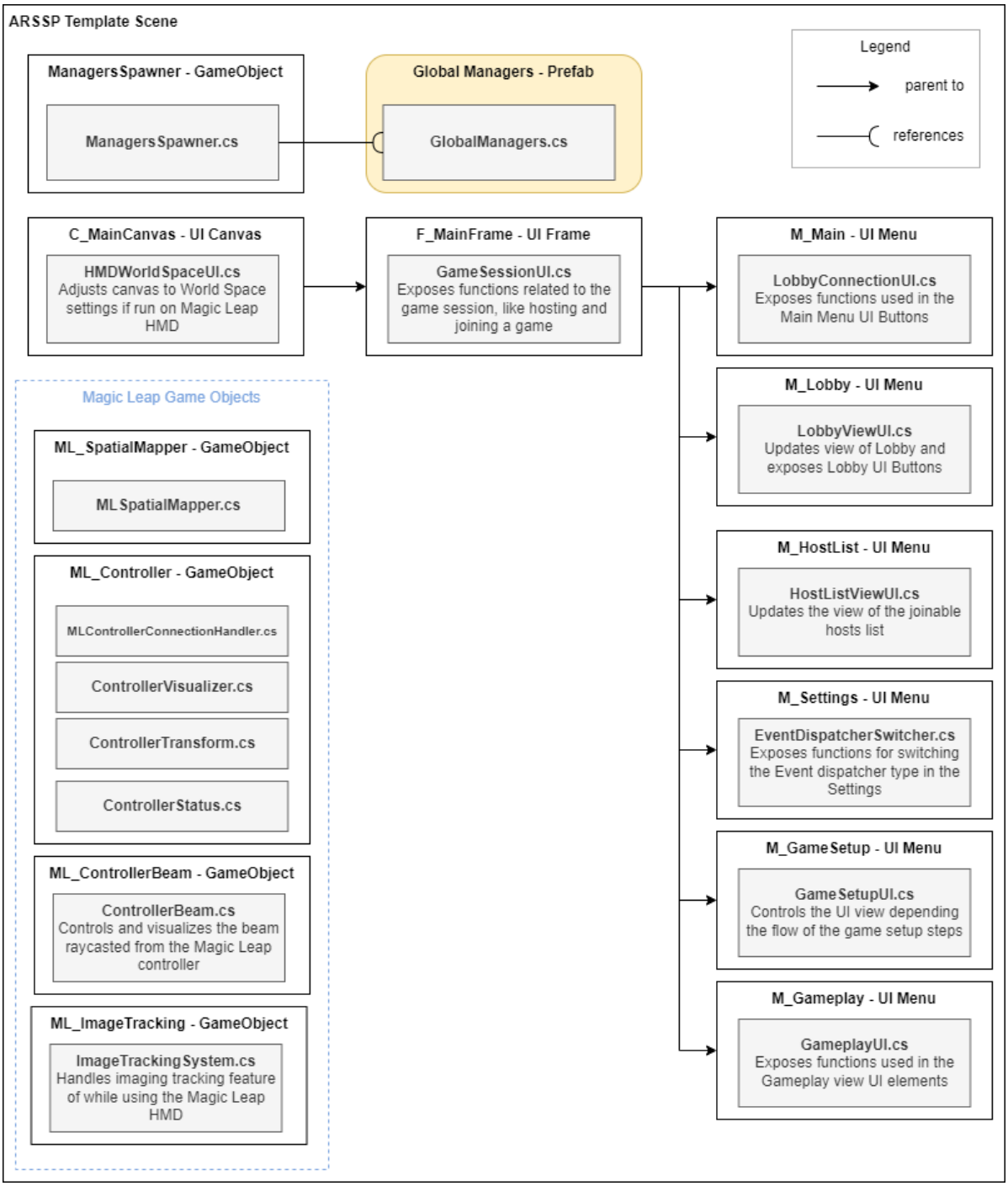


Figure 3.26: Diagram for the structure of the ARSSP template scene.

4 Evaluation

In this chapter, an evaluation of the modified ARSSP framework will be presented. The assessment of this evaluation comes in the form of how well does the ARSSP framework realize the AR Superhuman Sports game, Catching the Drone [44]. The steps of this assessment are as follows:

1. Defining a list of key technical aspects that are presented Catching the Drone game.
2. Providing a proof-of-concept game using the ARSSP framework that tries to realize Catching the Drone game's features.
3. Assessing what are the features that were successfully covered and what are the limitations currently present in the framework.

4.1 Technical Aspects of Catching the Drone

In this section, the key technical aspects of the Catching the Drone game will be discussed. These aspects can be split into three parts or layers: Networking, Augmentation, and Game Logic. Furthermore, some features were drawn from each part in the form of criteria that the game requires. In the following sub-sections, each part will be discussed and their requirements will be presented. Each criteria will be labeled with a tag (e.g. A.1) so that it can be referenced later in the evaluation.

4.1.1 Networking

Since Superhuman Sports games - like Catching the Drone - are multiplayer games, the networking layer is an important aspect to the game. The requirements for networking in Catching the Drone are listed as follows:

- **(N.1) Networking layer should abstract more than one networking solution.** Since networking can be unstable depending on the solution used, it is preferable to have more than one networking solution and the ability to easily switch between them. This also provides a possibility of integrating new types of devices into the system.
- **(N.2) Networking between cross-platform devices should be enabled.** Different players can be using different types of devices but they can still connect together and play the game.

- **(N.3) Integration of micro-controllers should be enabled in the network.** This is essential for integrating external hardware devices - like the drone ball - into the game itself.

4.1.2 Augmentation

Augmented Reality is a core part of Superhuman Sports game, especially for Catching the Drone. The key requirements for augmentation in Catching the Drone are as follows:

- **(A.1) Marker detection on the drone-ball should be enabled.** Catching the Drone uses a drone-ball with LED lights that need to be tracked and augmented by the players' devices.
- **(A.2) Augmentation on the tracked drone-ball should be enabled.** The appearance of the drone-ball can be altered through augmentation on the players' devices.
- **(A.3) Augmentation layer should abstract more than one AR solution.** This should allow swapping between different AR solutions that use different augmentation algorithms and might provide specific solutions for certain scenarios.

4.1.3 Game Logic

The Game Logic part defines the game features and the interactions that the player is able to do in the game. For Catching the Drone, these features can be presented as the following requirements:

- **(G.1) Game should provide team-assigning to players.** Catching the Drone is a competitive game between two teams. The game should allow the players to specify their team during the game.
- **(G.2) Game should have a globalized scoring system.** Catching the Drone has a scoring system for the teams to allow for winning cases in the game. This scoring should be replicated to all players' devices throughout the network.
- **(G.3) Player's abilities are enhanced by using communication with drone-ball.** One major aspect of Catching the Drone is that the players are provided with enhanced capabilities through feedback from the drone-ball and special abilities on the drone-ball.

4.2 ARSSP Assessment

After defining the key criteria presented by Catching the Drone game, the ARSSP framework is evaluated to see if it can successfully fulfill these criteria. This is done through an ARSSP proof-of-concept game that tries to implement those technical aspects of Catching the Drone. In this section, each requirement is discussed in terms of how the ARSSP game covers it and what are the present current limitations.

4.2.1 Networking

(N.1) Networking layer should abstract more than one networking solution.

The ARSSP's networking layer abstracts over two networking solutions, Ubi-Interact and Mirror. Both networks have to work and support hosting a game, joining a game, and broadcasting of BEvent messages. These features were tested on two different devices, concurrently, for both the Ubi-Interact and Mirror solutions. All features proved to work as intended with limitation to this requirement.

(N.2) Networking between cross-platform devices should be enabled.

The ARSSP currently supports several platforms including Android, iOS, and Magic Leap Lumin. Two Unity builds were made for the ARSSP game: one for Android and another for Magic Leap. They were both tested to see if they can connect together by hosting from one device and joining from the other. There are no limitations presented for that feature using both the Ubi-Interaction and Mirror networking solutions.

(N.3) Integration of micro-controllers should be enabled in the network.

As mentioned in subsection 3.3.6, the integration of the Raspberry Pi into the ARSSP network was implemented using Ubi-Interact python API solution. However, there is no implemented solution that supports networking through Mirror on the Raspberry Pi. With only this limitation noted, the communication between the Raspberry Pi and the ARSSP Unity client was tested through the Ubi-Interact network and is working as intended.

4.2.2 Augmentation

(A.1) Marker detection on the drone-ball should be enabled.

In Catching the Drone, the drone-ball contains LEDs that have a lighting pattern that should be detected by a device using OpenCV algorithms. This algorithm was not used during the implementation of the ARSSP game. However, an alternate solution was used: the ArUco marker detector mentioned in subsection 3.3.7. These markers can also be placed on the drone-ball for implementing a prototype. Furthermore, ARSSP's support for OpenCV also adds the possibility of integrating new OpenCV features in future work. This can include the LED tracking algorithm or any other OpenCV algorithm that is specific to a certain game feature. However, there is limitation on the Magic Leap device as OpenCV is not supported. Alternatively, an implementation of image-tracking provided by the Magic Leap XR API was used.

(A.2) Augmentation on the tracked drone-ball should be enabled.

Not only was the tracking of the markers implemented but also the augmentation of virtual 3D object on the tracked image. This should be used in order to augment the drone-ball and

alter its appearance by adding visual effects to it. In the ARSSP, this is tested by augmenting a virtual 3D sphere on tracked markers to simulate the drone-ball visuals in the augmented reality space.

(A.3) Augmentation layer should abstract more than one AR solution.

The ARSSP's augmentation layer currently abstracts over two AR solutions, AR Foundation and OpenCV. Both implement marker-tracking features. Furthermore, the AR Foundation framework provides even more built-in features - like plane detection and device tracking. Both solutions were tested on the ARSSP game and work as intended.

4.2.3 Game Logic

(G.1) Game should provide team assigning to players.

Since Catching the Drone is a team-based multiplayer game, a team-assigning mechanic should be added. During the implementation of the ARSSP lobby system - as mentioned in subsection 3.3.5 - a team-assigning mechanic was added to the game. Each player would assign themselves to the team they want to join through the lobby menu. Each player would be presented with a "Switch Team" button that toggles between available teams, with also a 'NONE' team state assigned for spectators. At the moment, the system allows for four different teams to add a spectator state. This can be further adjusted depending on the requirements of the ARSSP game in development.

(G.2) Game should have a globalized scoring system.

Since Catching the Drone is a competitive multiplayer game, there has to be a way to decide when a team wins the game. For this reason, a scoring system was implemented to keep track of the score of each team. This is done through Game Modes. The ARSSP provides several game modes with different game logic. ARSSP developers can create customized game modes for their Superhuman Sport games. For this prototype, a Catching the Done game mode was created to track the scoring of teams. This scoring is replicated throughout the network, so it is globalized to all players in the network.

(G.3) Player's abilities are enhanced by using communication with drone-ball.

Testing this feature fully was not possible due to the lack of an actual drone-ball for the prototype. It was not possible to test features like getting feedback from the drone-ball during the game or activating superhuman abilities on the drone. However, since the integration of external hardware - the Raspberry Pi - was implemented and functional, this opens up the possibility of easily integrating the communication with the drone-ball when it is available. This will, furthermore, require internal implementation on the drone-ball hardware itself.

4.3 Results

After evaluating the ARSSP prototype game to see how well it does in realizing a game like Catching the Drone, the results are generally positive with some limitations due to the lacking of required hardware. The limitations include lack of access to the actual drone-ball hardware during testing and to the OpenCV algorithms used for LED light-tracking. However, the results show that the ARSSP framework passes successfully in several criteria, which means that it is getting closer to making a superhuman sports game like Catching the Drone a reality. Nevertheless, there remains several areas to be improved in the framework. One of which is finding a solution to enable the use of OpenCV algorithms on the Magic Leap device. Another is finding a way to enable networking solutions other than Ubi-Interact - like Mirror - to work on external hardware like Raspberry Pi. Furthermore, further development and testing would, of course, be required when the physical drone hardware is available and integrated in the system.

5 Conclusion

5.1 Summary

This thesis aims to present a framework that would facilitate the development of multiplayer AR superhuman sports games for developers. The solution uses the ARSSP as a foundation and the research aims to work on its limitations and improve it. A set of steps were undertaken to achieve this aim. First, a literature review was done on several relevant topics: the previous AR network-based frameworks created, the software patterns used in games engineering, the state-of-the-art multiplayer AR games and superhuman sports, and the technologies relevant to this research. This review helped build a foundation for the implementation part.

The next step was implementation; the process involved was, first, reviewing and presenting the initial design state of the ARSSP framework. Afterwards, several new features were implemented to tackle the framework's limitations, extending its capabilities and improving certain parts of it. Finally, an evaluation was made of the framework based on how well it fulfills certain criteria to be able to implement a superhuman sports game like Catching the Drone.

The results of the evaluation show that the newly modified ARSSP framework brings us a step closer into fully implementing superhuman sports games that utilize networking and augmented reality. The framework successfully fulfills several criteria in the evaluation. However, the limitations presented were mostly the result of lacking of proper hardware to create a fully-functional demo for Catching the Drone game. Nevertheless, there is promising progress as several aspects of the game have been tackled.

5.2 Future Work

There is good room for future work after research's efforts. The ARSSP framework can be further improved in several ways. One way of improvement would be finding a solution to make OpenCV algorithms work on the Magic Leap device. Another challenge would be figuring out how to make networking options other than Ubi-Interact - such as Mirror - work on external hardware such as Raspberry Pi. Furthermore, since the demo application was missing the physical drone hardware, further development and testing is required when the hardware is available and integrated in the framework.

List of Figures

1.1 Concept art of novel superhuman sports game ideas. 1

2.1 Superhuman Sports Society logo. 4
2.2 Third-person perspective play capture of HADO. 5
2.3 High-level overview of the SARA architecture. 6
2.4 System architecture of AR Flutter Plugin. 8
2.5 Players getting ready to play VRabl (left) and a screenshot of the gameplay (right). 9
2.6 Overview of League of Lasers game concept. 9
2.7 Screenshot of League of Lasers' mirror. 9
2.8 Screenshots of STAR taken from the Hololens view. 10
2.9 Screenshots of Holofight demo. 11
2.10 Overview of Holofight application structure. 11
2.11 Screenshot for ARSoccer demo. 12
2.12 Illustration of Augmented Invaders' concept. 13
2.13 Screenshots of Brick on two devices. 13
2.14 Example of an entity component system structure. 15
2.15 Example of subject-observer relationship in Observer pattern. 15
2.16 Design structure of Command Pattern. 17
2.17 Overview of a game engine component structure. 17
2.18 Unity logo. 18
2.19 Unreal Engine logo. 19
2.20 AR application runninig on iPhone 3Gs. 20
2.21 AR applicaion on Android tablet. 20
2.22 Microsoft Hololens Gen 1 AR HMD. 20
2.23 Magic Leap AR HMD. 20
2.24 Virtual Super-Leaping with visual and haptic feedback. 21
2.25 Droneball in Catching The Drone Super Human Sports game. 21
2.26 Raspberry Pi 4 board. 22
2.27 Arduino Uno board. 22
2.28 ARCore logo. 22
2.29 ARKit logo. 23
2.30 Platforms and devices supported by MRTK framework [18]. 24
2.31 AR Foundation features support for each plugin [37]. 25
2.32 Vuforia logo. 26
2.33 OpenCV logo. 26

2.34 Mirror Networking logo. 28

3.1 ARSSP Software Architecture. 30
3.2 Diagram that visualizes the AR Framework abstraction of middle-level and low-level AR solutions. 31
3.3 A UML Class diagram that represents how the BEvent Framework communicates data. 33
3.4 An example of a BUIElements hierarchy, with the green elements currently on focus. 35
3.5 A UML diagram that demostrates the structure of BControllerManager and BPlayerManager. 36
3.6 A UML diagram that demonstrates the structure of the game mode and player stats. 36
3.7 Magic Leap logo. 38
3.8 Magic Leap Image tracking detecting an ArUco marker and placing a 3D ball on it. 39
3.9 Example of Screen Space UI canvas on screen. 39
3.10 Example of World Space UI canvas in VR. 39
3.11 Magic Leap motion controller. 41
3.12 Magic Leap view of controller raycasting on a world space UI that is visualized by a line renderer. 44
3.13 Main Menu is the ARSSP Demo Template. 46
3.14 Host List Menu in the ARSSP Demo Template. 47
3.15 Lobby Menu in the ARSSP Demo Template. 49
3.16 Settings Menu in the ARSSP Demo Template. 50
3.17 Raspberry Pi Zero W board. 52
3.18 Raspbian OS logo. 52
3.19 Drone-ball with LEDs. 56
3.20 LED color pattern to be detected. 56
3.21 ArUco markers example. 57
3.22 The input and result iamges of the ArUco marker detector. 57
3.23 Screenshot of AR Marker Tracker example running on android mobile device. 58
3.24 Inspector view of the Managers Spawner script in the Unity Editor. 59
3.25 Diagram for the structure of the Global Managers prefab. 61
3.26 Diagram for the structure of the ARSSP template scene. 63

Bibliography

[1] K. Kunze, K. Minamizawa, S. Lukosch, M. Inami, and J. Rekimoto. "Superhuman sports: Applying human augmentation to physical exercise". In: *IEEE Pervasive Computing* 16.2 (2017), pp. 14–17.

[2] R. Ando, A. Ando, K. Kunze, and K. Minamizawa. "Bubble jumper: Enhancing the traditional japanese sport sumo with physical augmentation". In: *Proceedings of the First Superhuman Sports Design Challenge: First International Symposium on Amplifying Capabilities and Competing in Mixed Realities*. 2018, pp. 1–6.

[3] H. Araki, H. Fukuda, T. Motoki, T. Takeuchi, N. Ohta, R. Adachi, H. Masuda, Y. Kado, Y. Mita, D. Mizukami, et al. ""HADO" as Techno Sports was born by the fusion of IT technology and sports". In: *ReVo* (2017), pp. 36–40.

[4] L. Carius, C. Eichhorn, D. A. Plecher, D. A. Plecher, and G. Klinker. "Cloud-Based Cross-Platform Collaborative AR in Flutter". In: TUM. 2020.

[5] M. Ben Jazia. "Software Design of a Dynamic AR Superhuman Sports Platform-Independent Architecture". In: (2020).

[6] *Superhuman Sports Academy*. `http://superhuman-sports.org/academy/eindex.html`. Accessed: 2022-04-10.

[7] D. Vaquero-Melchor, A. M. Bernardos, and L. Bergesio. "SARA: A Microservice-Based Architecture for Cross-Platform Collaborative Augmented Reality". In: *Applied Sciences* 10.6 (2020), p. 2074.

[8] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel. "The pains and gains of microservices: A systematic grey literature review". In: *Journal of Systems and Software* 146 (2018), pp. 215–232.

[9] J. Ren, Y. He, G. Huang, G. Yu, Y. Cai, and Z. Zhang. "An edge-computing based architecture for mobile augmented reality". In: *IEEE Network* 33.4 (2019), pp. 162–169.

[10] R. Shea, A. Sun, S. Fu, and J. Liu. "Towards fully offloaded cloud-based AR: Design, implementation and experience". In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. 2017, pp. 321–330.

[11] W. Zhang, B. Han, and P. Hui. "On the networking challenges of mobile augmented reality". In: *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. 2017, pp. 24–29.

[12] S. Boukhary and E. Colmenares. "A clean approach to flutter development through the flutter clean architecture package". In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2019, pp. 1115–1120.

[13] T. Buckers, B. Gong, E. Eisemann, and S. Lukosch. "VRabl: stimulating physical activities through a multiplayer augmented reality sports game". In: *Proceedings of the First Superhuman Sports Design Challenge: First International Symposium on Amplifying Capabilities and Competing in Mixed Realities*. 2018, pp. 1–5.

[14] S. Ong. "Introduction to the HoloToolkit". In: *Beginning Windows Mixed Reality Programming*. Springer, 2017, pp. 81–93.

[15] N. A. Miedema, J. Vermeer, S. Lukosch, and R. Bidarra. "Superhuman sports in mixed reality: The multi-player game League of Lasers". In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. 2019, pp. 1819–1825.

[16] M. Kegeleers, S. Miglani, G. M. Reichert, N. Z. Salamon, J. T. Balint, S. G. Lukosch, and R. Bidarra. "Star: superhuman training in augmented reality". In: *Proceedings of the First Superhuman Sports Design Challenge: First International Symposium on Amplifying Capabilities and Competing in Mixed Realities*. 2018, pp. 1–6.

[17] S. Kohen, C. Elvezio, and S. Feiner. "HoloFight: An Augmented Reality Fighting Game". In: *ACM SIGGRAPH 2021 Immersive Pavilion*. 2021, pp. 1–2.

[18] *microsoftMixedRealityToolkitUnity Mixed Reality Toolkit MRTK provides a set of components and features to accelerate crossplatform MR app development in Unity*. `https://github.com/microsoft/MixedRealityToolkit-Unity`. (Accessed: 14/3/2022). May 2018.

[19] D. Jara, F. Márquez, and A. Barrientos. "ARSOCCER: Development of a Multiplayer Real Time Videogame with Augmented Reality and Knowledge Based Agents". In: *2021 IEEE Engineering International Research Conference (EIRCON)*. IEEE. 2021, pp. 1–4.

[20] A. S. Markovits and A. I. Green. "FIFA, the video game: a major vehicle for soccer's popularization in the United States". In: *Sport in Society* 20.5-6 (2017), pp. 716–734.

[21] A. R. Bharambe, J. Pang, and S. Seshan. "Colyseus: A Distributed Architecture for Online Multiplayer Games." In: *NSDI*. Vol. 6. 2006, pp. 12–12.

[22] M. Bonfert, I. Lehne, R. Morawe, M. Cahnbley, G. Zachmann, and J. Schöning. "Augmented invaders: A mixed reality multiplayer outdoor game". In: *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*. 2017, pp. 1–2.

[23] *UNet - Unity Manual Multiplayer and Networking*. `https://docs.unity3d.com/Manual/UNet.html`. Accessed: 2022-04-10. June 2015.

[24] *Vuforia Developer Portal*. `https://developer.vuforia.com/`. (undefined 7/6/2022 20:15). Dec. 2012.

[25] P. Bhattacharyya, Y. Jo, K. Jadhav, R. Nath, and J. Hammer. "Brick: A Synchronous Multiplayer Augmented Reality Game for Mobile Phones". In: *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–4.

[26] M. Lanham. *Learn ARCore-Fundamentals of Google ARCore: Learn to build augmented reality apps for Android, Unity, and the web with Google ARCore 1.0*. Packt Publishing Ltd, 2018.

[27] R. Nystrom. *Game programming patterns*. Genever Benning, 2014.

[28] R. Nystrom. *Component Decoupling Patterns - Game Programming Patterns*. https://gameprogrammingpatterns.com/component.html. Accessed: 2022-04-09. Feb. 2010.

[29] E. Gamma, R. Helm, R. Johnson, R. E. Johnson, J. Vlissides, et al. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[30] R. Nystrom. *Observer Design Patterns Revisited - Game Programming Patterns*. https://gameprogrammingpatterns.com/observer.html. Accessed: 2022-04-09. Jan. 2014.

[31] R. Nystrom. *Singleton - Design Patterns Revisited Game Programming Patterns*. https://gameprogrammingpatterns.com/singleton.html. Accessed: 2022-04-09. Sept. 2009.

[32] M. Seemann. *Dependency injection in. NET*. Manning New York, 2012.

[33] R. Nystrom. *Command Design Patterns Revisited - Game Programming Patterns*. https://gameprogrammingpatterns.com/command.html. Accessed: 2022-04-09. Oct. 2013.

[34] J. Gregory. *Game engine architecture*. AK Peters/CRC Press, 2018.

[35] U. Technologies. URL: https://unity.com/.

[36] J. Koetsier. *With billions of people and millions of apps, can unity create the metaverse?* Dec. 2021. URL: https://www.forbes.com/sites/johnkoetsier/2021/12/07/with-billions-of-people-and-millions-of-apps-can-unity-create-the-metaverse/.

[37] *AR Foundation - Unity Documentation*. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html. (Accessed: 13/2/2022). Mar. 2021.

[38] A. Sanders. *An introduction to Unreal engine 4*. AK Peters/CRC Press, 2016.

[39] *Augmented Reality Overview - Unreal Engine Documentation*. https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/XRDevelopment/AR/HandheldAR/AROverview/. (Accessed: 13/2/2022). Mar. 2022.

[40] A. Vieira. *How smartphones and AR are changing the way we see and interact with the world*. (Accessed: 15/3/2022). Sept. 2020.

[41] S. Yoo and C. Parker. "Controller-less interaction methods for Google cardboard". In: *Proceedings of the 3rd ACM Symposium on Spatial User Interaction*. 2015, pp. 127–127.

[42] I. E. Sutherland. "A head-mounted three dimensional display". In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. 1968, pp. 757–764.

[43] T. Sasaki, K.-H. Liu, T. Hasegawa, A. Hiyama, and M. Inami. "Virtual super-leaping: Immersive extreme jumping in VR". In: *Proceedings of the 10th Augmented Human International Conference 2019*. 2019, pp. 1–8.

[44] C. Eichhorn, A. Jadid, D. A. Plecher, S. Weber, G. Klinker, and Y. Itoh. "Catching the Drone-A Tangible Augmented Reality Game in Superhuman Sports". In: *2020 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE. 2020, pp. 24–29.

[45] *Raspberry Pi Documentation - Getting Started*. https://www.raspberrypi.com/documentation/computers/getting-started.html. Accessed: 2022-05-09.

[46] *Arduino Home*. https://www.arduino.cc/. Accessed: 2022-05-09.

[47] *Build new augmented reality experiences that seamlessly blend the digital and physical worlds nbsp;| nbsp; arcore nbsp;| nbsp; google developers*. URL: https://developers.google.com/ar.

[48] D. Burke. *Arcore: Augmented reality at Android Scale*. Aug. 2017. URL: https://blog.google/products/google-ar-vr/arcore-augmented-reality-android-scale/.

[49] *ARKit Developer Documentation*. URL: https://developer.apple.com/documentation/arkit.

[50] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[51] P. Nowacki and M. Woda. "Capabilities of arcore and arkit platforms for ar/vr applications". In: *International Conference on Dependability and Complex Systems*. Springer. 2019, pp. 358–370.

[52] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov. "Real-time computer vision with OpenCV". In: *Communications of the ACM* 55.6 (2012), pp. 61–69.

[53] S. Weber, M. Ludwig, and G. Klinker. "Ubi-Interact: A modular approach to connecting systems". In: *EAI Endorsed Transactions on Mobile Communications and Applications* 6.19 (2021), e5.

[54] *Mirror Networking Open Source Networking for Unity*. https://mirror-networking.com/. (Accessed: 14/2/2022). Oct. 2019.

[55] *Delegate Events VS UnityEvent, which one is superior?* https://www.reddit.com/r/Unity3D/comments/35oekm/delegate_events_vs_unityevent_which_one_is/. Accessed: 2022-05-10.

[56] *What's new in Unity 2020.3 - Unity Technologies Website*. https://unity3d.com/unity/whats-new/2020.3.0. Accessed: 2022-05-09.

[57] *Image Tracking - Unity | Magic Leap*. https://developer.magicleap.com/en-us/learn/guides/image-tracking-overview-unity. Accessed: 2022-04-10.

[58] *PhysicsRaycast - Unity Scripting API*. https://docs.unity3d.com/ScriptReference/Physics.Raycast.html. Accessed: 2022-04-09. May 2014.

[59] *RaycastHit - Unity Scripting API*. https://docs.unity3d.com/ScriptReference/RaycastHit.html. Accessed: 2022-04-09. June 2014.

[60] *WebGL Wiki page*. https://www.khronos.org/webgl/wiki/Main_Page. Accessed: 2022-04-10.

[61] *Raspbian Documentation and Support*. https://www.raspbian.org/. Accessed: 2022-05-09.

[62] M. Schmidt. *Ubi-Interact Python Framework*. https://ubii-node-python.readthedocs.io/en/develop/index.html. Accessed: 2022-05-09.

[63] *OpenCV plus Unity - Unity Asset Store*. https://assetstore.unity.com/packages/tools/integration/opencv-plus-unity-85928#description. Accessed: 2022-05-10.

[64] *AR Marker Detector - Unity Asset Store*. https://assetstore.unity.com/packages/templates/tutorials/ar-marker-detector-80086#description. Accessed: 2022-05-10.

[65] *Odin Inspector*. https://odininspector.com/. Accessed: 2022-05-10.