UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Turn-based game for two based on classic tabletop game "Naval Battle" using Unreal Engine's multiplayer framework and Google's ARCore framework.

Author:
BOHDAN PYSKO

Supervisor:
TARAS LESKIV

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

I, BOHDAN PYSKO, declare that this thesis titled, “Turn-based game for two based on classic tabletop game "Naval Battle" using Unreal Engine’s multiplayer framework and Google’s ARCore framework.” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

*"Russian warship, go **** yourself"*

Roman Hrybov[1]

[1] According to the officials, Roman is not the real author

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Turn-based game for two based on classic tabletop game "Naval Battle" using Unreal Engine's multiplayer framework and Google's ARCore framework.

by BOHDAN PYSKO

Abstract

Game development industry is rapidly evolving nowadays, and I, as a game developer, wrote this thesis to explore the capabilities of perhaps the most dominant game engine up to date which is Unreal Engine. It is suitable for many tasks besides gamedev, particularly architectural visualisation, education, product design and so on. But one of the most complex and useful technologies it provides is the multiplayer framework. While making a major contribution to communication between users and overall UX, it is also hard to master, what makes it even more enticing to explore.
Additional motivation behind choosing UE is that UE5 was announced a year ago and was highly praised in the community[2].
To make my thesis even more interesting, I decided to add AR functionality to it.

With Augmented Reality market segment expanding, deeper integration of AR devices will be observed both in our everyday life and jobs. Besides, Apple products' LiDAR capabilities warm up my interest to AR development even more.

[2]Even though UE5 is a technological feat, this project was developed using UE4 due to UE5 being in beta stage until recently

Acknowledgements

I would like to express my gratitude to:

- Nineva Studios - for providing me with some great hardware
- Oleg Farenyuk - for teaching me the core of computer science
- Sam Pattuzzi and others for helping me learn the intricacies of Unreal Engine multiplayer
- and finally, Epic Games for letting 7 million creative people across the globe use state-of-the-art tools for free

Contents

List of Figures

List of Tables

List of Abbreviations

UE	**U**nreal **E**ngine
AR	**A**ugmented **R**eality
HMD	**H**ead **M**ounted **D**isplay
OSS	**O**nline **S**ubsystem **S**team
MMO	**M**assive **M**ultiplayer **O**nline
PVP	**P**layer **V**ersus **P**layer
LAN	**L**ocal **A**rea **N**etwork
NPC	**N**on-**P**layer **C**haracter
API	**A**pplication **P**rogramming **I**nterface
LiDAR	**Li**ght **D**etection **A**nd **R**anging
UMG	**U**nreal **M**otion **G**raphics
RPC	**R**emote **P**rocedure **C**all
BSP	**B**inary **S**pace **P**artitioning

To my family

Chapter 1

Introduction

This thesis covers many different aspects of game development, mainly:

- connection between players
- augmented reality
- real-time mesh deformation

In this section, you will see explanations why these three topics are worth investigating as of today and what are the challenges they can impose.
The project attached to this thesis is the digital implementation of my research. It is a turn-based multiplayer game in Augmented Reality, particularly handheld kind of it. It means that the build was made for Android and AR is implemented using the standard phone camera.
The game itself is based on a classic tabletop game "Naval Battle", you can take a look at the rules here:
Bradley, 1967
Although my project has fewer limitations imposed on the game, e.g. fields can vary in size, there is one more ship shape etc, the following text is written with the assumption that the reader knows the rules in mind.

1.1 Multiplayer

As mentioned above, multiplayer is one of the key aspects of modern gaming experience. It is mostly used in games nowadays, but maybe the situation will change so that more people can experience its benefits.
Multiplayer technology elevates user experience to a new level. It can facilitate player cooperation or the opposite - form a PVP match for a realistic difficulty level as well as providing creative and unpredictable opponents for everybody. Many game studios invest high amounts of resources into game AI. Enabling and NPC to behave like a human is not an easy task.

Multiplayer capability can be used in a wide variety of ways with different purposes. Also, multiplayer games can be divided into three categories based on their synchronicity and session length.

- widely known MMO games - synchronous gameplay with potentially infinite sessions and large amount of participants. Players can enter and leave the so-called world at any time.

- real-time session-based games - a good example of which can be the recently very popular app Among Us. It is a very small piece of software, does not even use 3D graphics, and encompasses a big amount of genres and gameplay mechanics. It can be mainly defined as a puzzle game, based on social interaction. You cannot win it just by learning internal mathematics and clicking quickly, it requires communication skills and understanding social interactions. A very good example of how to take advantage of multiplayer technology.
- turn-based games - originating from what may well be the oldest games ever - chess and alike. This genre can be characterised by not too long sessions and what is important - asynchronous approach.

A table to sum up:

Genre	Synchronicity	Session Length
MMO	Yes	Potentially infinite
Session-based	Yes	Usually < 1 hour
Turn-based	No	Usually < 1 hour

Now may be a good time to explain what synchronous means in multiplayer. Generally, players need to communicate somehow via the game itself to consider it a multiplayer experience. Whether it be a chess board, an interactive maze or, most commonly, a mutual level (aka world). In most cases, when players stay in the same level and *see each other* or the consequences of each other's actions - the game is *synchronous*. Maintaining synchronicity is crucial to games like this because, like in real life, actions and following events are strictly consecutive. The problem is, communication is not instant, especially when players are far from each other. With every multiplayer game the engineers try to compensate the lag by transferring only the minimal amount of data over large distances and reproducing other player's actions on the local PC. This is the beauty of multiplayer technology - and it's most complex part.
This project does not require any complex code in terms of multiplayer because it is a turn-based game (which means asynchronous, as stated above) and also LAN-based - players are relatively close and the devices communicate to a common router or even directly to each other. But the most complex part here is to coordinate players' position in the game world *as well as the real one*. More on that in the next section.

1.2 Augmented reality

8 years have passed since the collocations "AR" and "VR" became familiar to the masses, many devices have been launched too. Although these technologies have not yet been widely adopted, it is only a question of time. With the emergence of more energy efficient mobile processing units, better energy storing technologies and higher quality HMDs these types of hardware will become more and more common.
AR/VR can be applied to increase the quality of life of many poeple - doctors, students, pilots etc. Besides, Facebook has been developing these technologies rapidly over the past few years. With the launch of Metaverse people's lives can change drastically. This virtual world is a topic worth investigating, and it encompasses the same technologies as this project - AR and multiplayer. This is part of the reason why this topic was chosen for my research.
Concerning market size, AR/VR market was estimated at 17 billion USD by the end

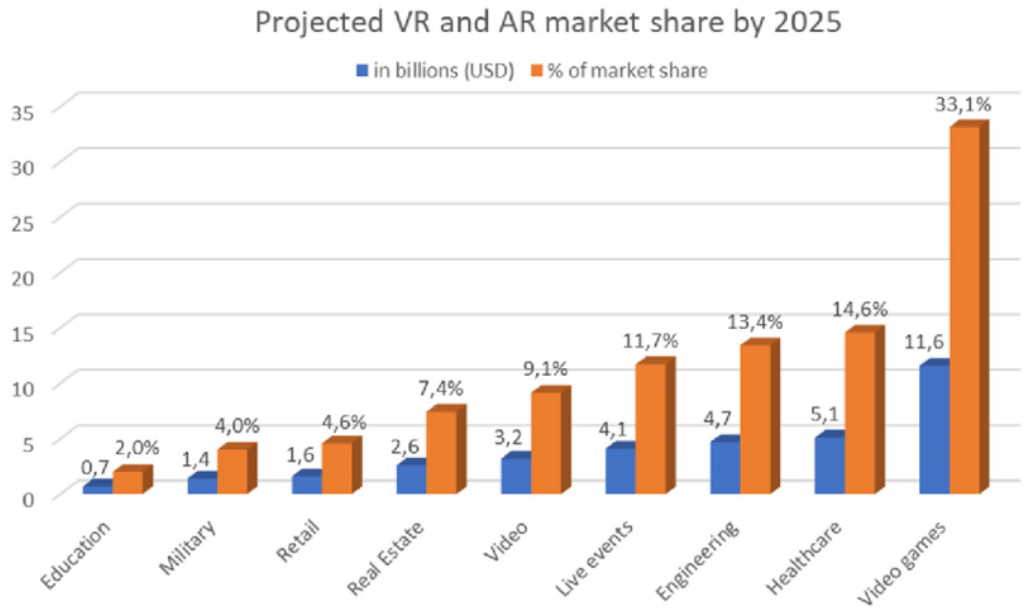

FIGURE 1.1: ResearchGate AR/VR market projection

of 2021 and is expected to reach 105 billion by the year 2028. (VantageMarketResearch, n.d.) I personally find it difficult to make predictions for 6 years from now, mainly because of the mentioned Metaverse. It is hard to overestimate its influence on the industry.
Another source (Markopoulos and Luimula, 2020) makes predictions for 2025:

1.3 Procedural mesh slicing

Real-time mesh deformation, or a subset of it - procedural mesh slicing, are both very important parts of modern graphics. They enable developers to implement dynamic surfaces like human skin, waves and trees and visual effects like disintegration, wreckage and many more.

1.4 Purpose of this thesis

This thesis aims at investigating the capabilities of Unreal Engine multiplayer technology and AR capabilities of the said engine. It is not a scientific synthesis of some state-of-the-art algorithm. While such algorithms will be discussed further with the purpose of providing theoretical background for the mobile application attached, this thesis is a description of my experience of combining two complex modern technologies to implement a game that is relatively novel by its concept.

Chapter 2

Related works

2.1 Games like the one attached and why handheld AR format was chosen

There have been many turn-based PVP games released since the term "multiplayer" became widely known. Variants of Naval Battle were among them. Although, the adoption of AR technology does not seem so widespread. I chose handheld AR for this game for the reason of simplicity of access. While dedicated AR/VR devices do not see much adoption yet, cell phones can be found in every pocket, so the game can be played anywhere and anytime, you just need a flat surface.

2.2 Theoretical background

This section describes essential to know concepts, as well as the papers used as a theoretical background for the application attached.

2.2.1 Multiplayer

While many of the first classic games could be called multiplayer, they were actually non-networked, this means, two players were playing on the same device. A classic example of a non-networked game would be Mortal Kombat - a sample of fighting genre, where two players use different joysticks, but the same computing hardware. But modern notion of multiplayer implies that players are connected via the Internet. This is much more complex connection to establish, fortunately, nowadays almost all of the networking process is hidden under numerous abstraction layers, the only aspect of it that developers need to take care of is synchronicity and lag.
While there is no need for explanations here as to how LANs or WANs work, the principles of establishing and maintaining user connection via UE and Steam (OSS) API will be described in detail in "Technical difficulties and solutions" chapter.

2.2.2 AR

There are two kinds of AR - with translucent displays (HoloLens, Google Glass) and non-translucent - Oculus devices, Handheld AR. Only the latter is the object of this investigation.
The problem with non-translucent AR is that it is fairly hard to seamlessly integrate virtual objects and the real world due to how challenging a task of measuring distances to real objects is. Proper occlusions need to be implemented, surface tangents, etc. What follows is an analysis of approaches to said task. There are 3 options when it comes to measuring distance. The device either has one camera, or 2, or a whole

LiDAR scanner. Obviously, the last one available only to Apple devices as of now.
Unreal Engine supports all of the most popular devices - Android, iOS, Oculus. This project was implemented for and tested on a Google Pixel smartphone with one camera.

One camera

Alizadeh, 2015 proposed two methods of estimating the distance to an object using a single camera.
One of them is using object's given size - which is impossible to apply in real-world handheld AR use cases; and the other is using the height of the camera and the estimated horizontal projection of the distance from camera to the object. The second method can be applied to more use cases, but it is also quite impossible to use it for Handheld AR needs due to lack of information on device height.
One of the universal approaches to this problem is machine learning - by classifying some of the scene objects, the algorithm can estimate their real size and use the first algorithm of the two mentioned above to calculate the distance. But such a program could possibly take hundreds of megabytes and require a powerful GPU, which is rarely available on mobile devices.
But there is one more option which is estimating optical flow. OF is a representation of how objects move in the scene. If two adjacent regions have nearly the same OF it means that they are probably a part of the same object or two objects moving together.
OF is a 2-dimensional matrix of 2D vectors, or a tensor. Each vector starts in the position of particular pixel and points in the direction which is basically a projection of the corresponding object's speed onto the camera sensor plane.
It is relatively easy to estimate the OF: the most straightforward way to do it is recognising patterns and their evolution over time. And *this* is a task that is very suitable for a relatively small neural network.
There are many papers written to this date on the topic of optical flow, because it is one of the crucial aspects of distinguishing one object from the other and computer vision overall.
The famous Lucas-Kanade method for estimating optical flow is described in much detail by now, I suggest that the reader take a look at this algorithm.

Here the reader can see an illustration of what OF is (IMO = independently moving object) (top left part probably illustrates the notion of OF the best):

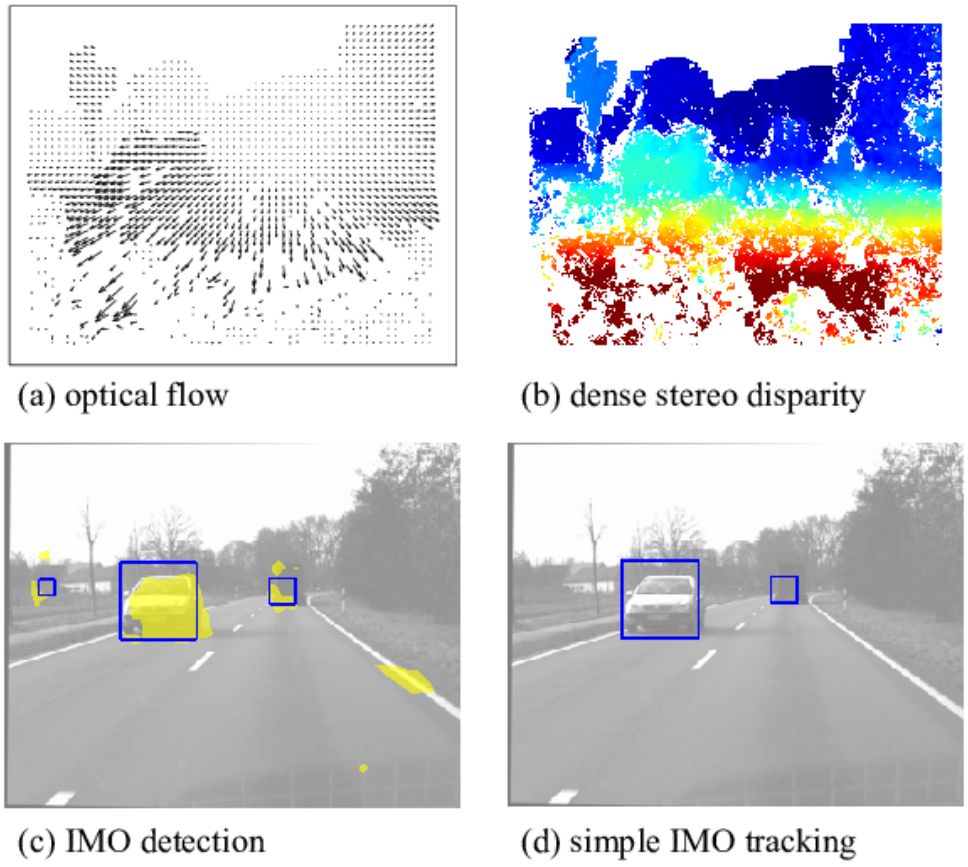

FIGURE 2.1: Example of optical flow

The advantages of this approach are obvious: it can be easily used to determine the speed of the object and hence, its distance to the AR device. The faster the object moves in the frame during camera movements (which can be determined using a gyroscope), the closer it is to the device.
The OF approach comes with limitations, though. If the camera is moving very little or is completely steady, it will be impossible to estimate the optical flow. Also, if the scene has very little steady objects, or at least objects that do not accelerate, i.e., move with a constant speed, it will be hard to figure out how much the optical flow was influenced by the AR device movement and how much - by movement of the object itself.

Two cameras

Using parallel streams of data from two cameras the algorithm can estimate distance to objects in the frame just like humans with their both eyes. In AR-capable devices market, this was successfully implemented by the first 2-camera cell phone which is iPhone 7+ in 2016.

Neshov and Manolova, 2021 described an approach to processing data from two cameras with the purpose of providing better user experience with AR devices.

Two sections above can also benefit from some sort of image segmentation based on colours like Grady et al., 2012, or even an alpha-matting algorithm. I, the author

of this thesis, have implemented the Random Walks algorithm at https://github.com/pyskonus/random_walker/tree/mask

LiDAR

The LiDAR sensor provides devices with an absolutely new type of data - it is actually just the data needed for implementing proper occlusions and other functions during an AR session. The issue is, LiDARs are quite an expensive hardware.

Still, it is worth mentioning that Kumar et al., 2020 proposed an algorithm for measuring distance in automotive industry using LiDAR and camera data together. Probably light scanners are not going to see widespread adoption in the next few years, but much cheaper ultrasonic scanners totally can.

Unreal Engine supports both non-LiDAR Google ARCore framework for Android devices, and Apple ARKit for LiDAR-enabled iPhones.

2.2.3 Mesh slicing

Parker and O'Brien, 2009 described in implementation of a system for deformation and fracture of solid objects in real-time gaming. This article is not necessary to understand the further described technical difficulties and solutions, but it can help the reader understand how objects are represented in a virtual world and what could be the implementations for mesh slicing.

Schmidt, 2020 , an employee of Epic Games provided a tutorial that I, the author, followed to get to know Unreal's API for mesh generation and deformation.

Chapter 3

Technical difficulties and solutions

UE's built apps usually take hundreds of megabytes, so I will be unable to provide you with a Git repository that contains .exe and .apk files, only source code. I will include screenshots instead, executables will be demonstrated on June, 20-24.

3.1 First investigation in the scope of this thesis

I would like to mention that on
https://github.com/pyskonus/UdemyMultiplayer
the reader can see the first attempt to investigate Unreal's multiplayer framework in the scope of this thesis.
This project will demonstrate to the reader how UE's internals work and how a session-based multiplayer app would function. It is appropriate here because session-based games are considerably better supported by Unreal than turn-based ones. Therefore, it should be much easier to understand how multiplayer works on a session-based example.

This application, just like the final project of this thesis, enables players to host an online session as well as join one. But unlike the final app, this one has a slightly less detailed user interface, where you have to type an IP address in order to connect to another player. What follows is a short description of a very simple multiplayer game in Unreal.

First, two users start a game instance on their machines. UE has a corresponding class for this. UGameInstance is a class that represents the game process. It is a singleton, one instance, or copy, of this class, is created when the game loads up and it is destroyed when the game shuts down. This class is used to implement connections between players.
Unreal Engine uses a client-server model of multiplayer, which implies, one of the game instances is authoritative and the others are not. This is used in order to exclude the possibility of cheating and, more importantly, to synchronise players.

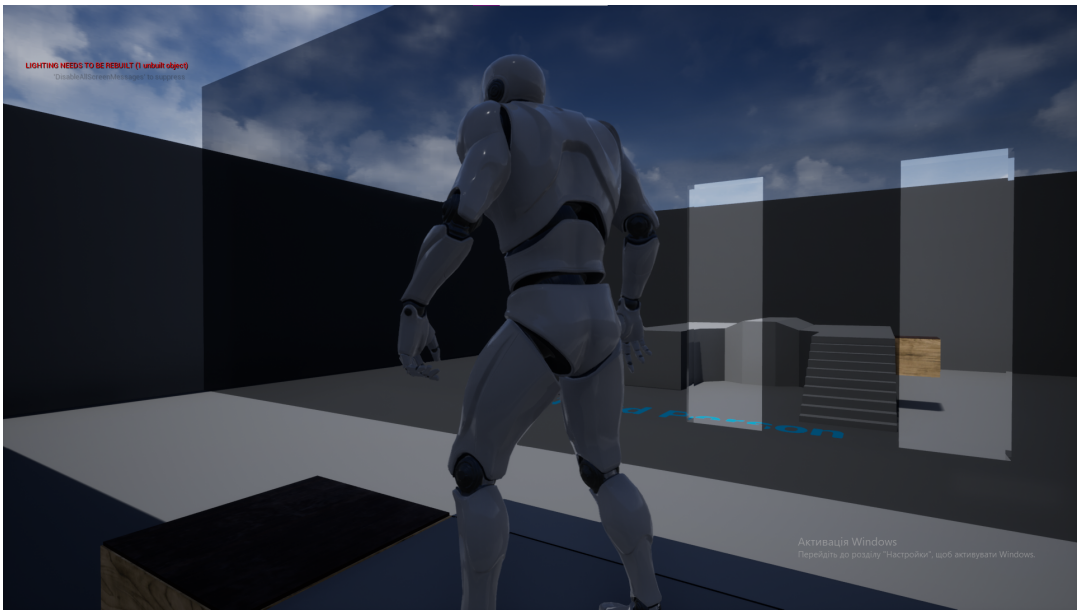

FIGURE 3.1: Host entered the common world, client - not yet

When players join, they are presented with two platforms. You can see them in bottom left corner of the image above and middle right. Each platform has a button on the floor. Each button moves the opposite platform. This way, if one player steps on their platform, another player's platform will move. The goal of this simple puzzle is to reach the opposite destination, i.e. players must step on their platforms simultaneously.

After the host has entered the world and started an online session, another player, or client, can join:

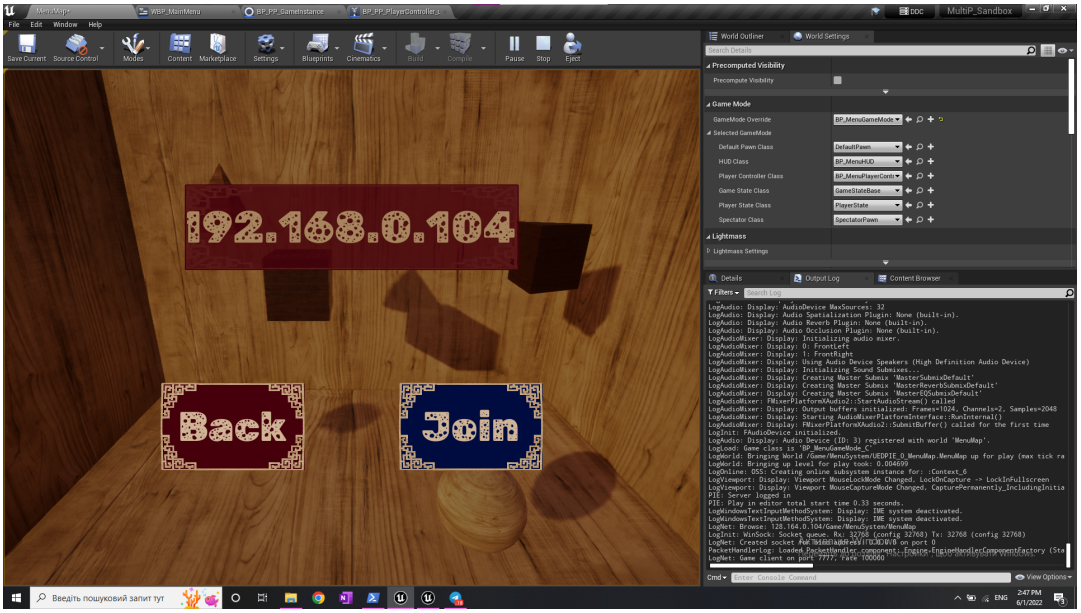

FIGURE 3.2: Joining process (final app does not require text input)

On the next screenshot you can see that the client has joined the world and is ready to interact with the level:

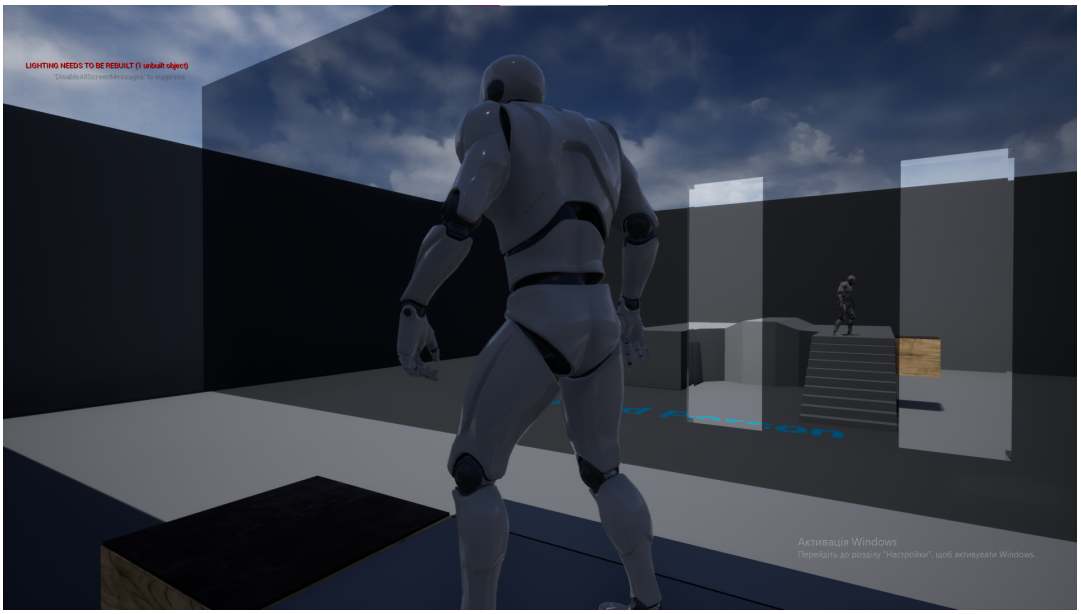

FIGURE 3.3: Now client has joined

Server has stepped onto his platform and it has called an event, aka a delegate, to the client's platform.
Delegates are similar to hardware interrupts on a lower level of abstraction.

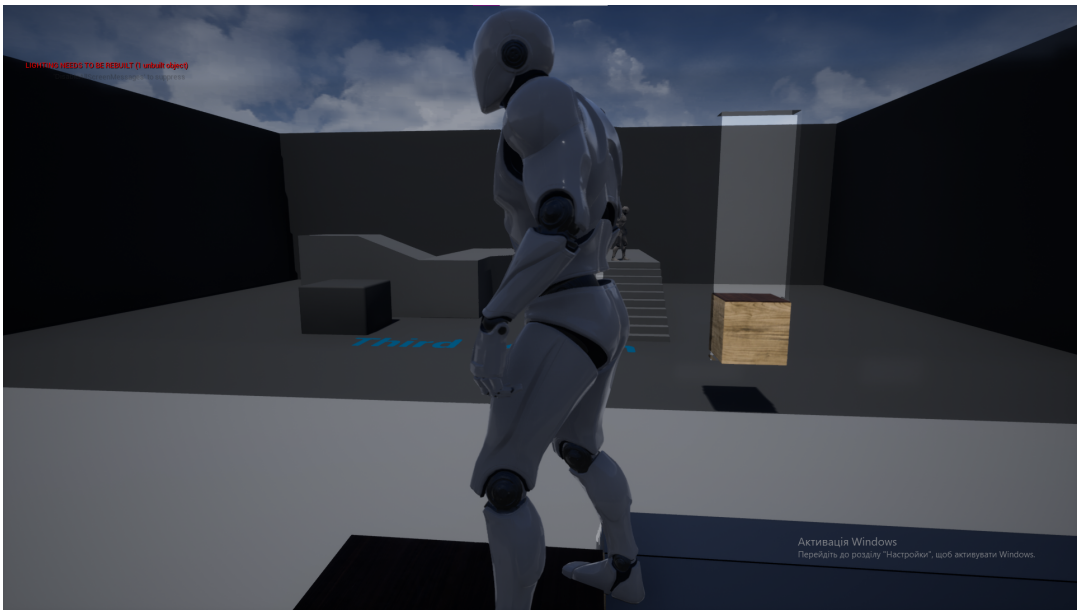

FIGURE 3.4: Client's platform moving after server stepped onto his

Now two players stepped on their platforms:

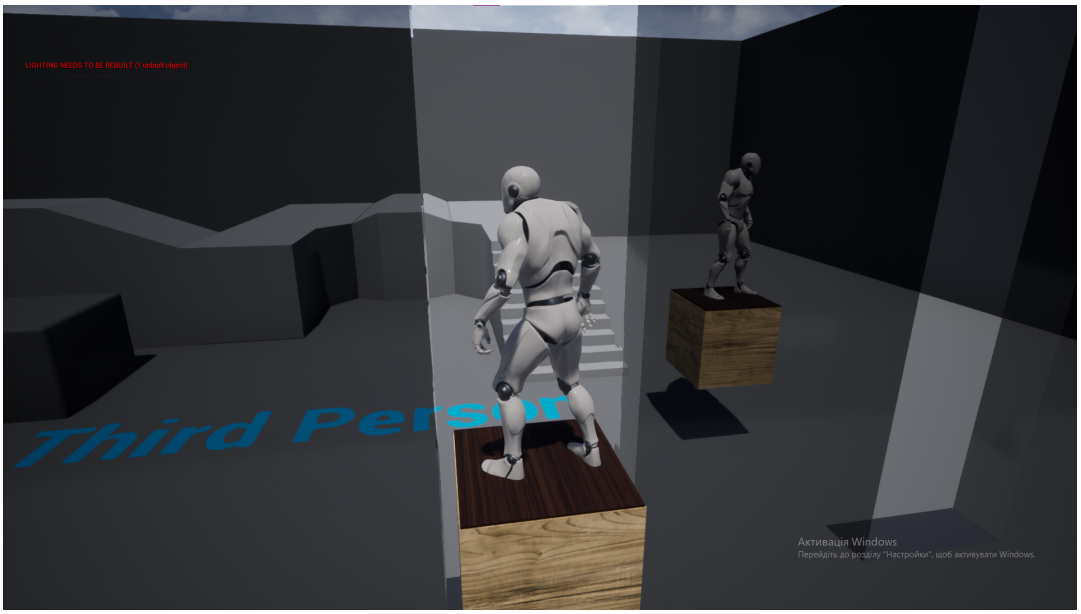

FIGURE 3.5: Both players are moving

Mow both players have reached their destinations:

FIGURE 3.6: Players have reached their destinations

Now I will very shortly describe the UE multiplayer API - what it takes to host a session, join one, and reproduce actions of players for one another.

3.1.1 Menu

Menu was implemented with the help of Unreal's UMG module. It provides developers with the necessary UI elements like buttons, text input fields etc. All these represent their corresponding classes and are capable of registering the appropriate callbacks, properties and more.
In this game, menu was implemented as a separate module that can be easily decoupled, therefore those parts of code that interact with it have been inherited from Unreal's special interface class, much like interfaces in Java.

3.1.2 UGameInstance

Host and Join functionality is implemented in this class.

```
World->ServerTravel("/Game/ThirdPersonCPP/Maps/ThirdPersonExampleMap?listen")
```

This is a function used to host a session, it takes path to the multiplayer map and ?listen at the end of it to state that this map must be hosted by a listen server.
Unreal also supports dedicated servers, which means, level is run on a remote PC and this PC does not represent any players, nor does it take any input.

```
PlayerController->ClientTravel(Address, ETravelType::TRAVEL_Absolute)
```

This function enables player to join a server, hosted by another player.

Next, I will describe how actor replication works, i.e. the platforms.

There are two different notions of replication - actor replication and property replication. Actors are updated either through property replication or remote procedure calls (RPCs). Simply put, RPCs are functions that run both on server and client at the same time. I suggest that the reader take a look at the following documentation to have a deeper understanding of how replication works in Unreal: EpicGames, 2022

As mentioned above, the server is the authoritative part of multiplayer while the client is not. This means, server is responsible for running core gameplay logic that is crucial to the game's outcome, and the client is only responsible for constructing an appropriate picture to the remote player's screen and transferring this player's input to the server.
So since distinguishing these two is so important, there is a corresponding function in the engine -

```
if (HasAuthority())     // this means, if this instance of code, or process,
{                       // is running on the server
    DoSomething();
} else
{
    DoSomethingElse();  // and this will only run on the client side
}
```

Of course, there is much more to multiplayer gaming than what is mentioned here, part of it will be described in subsequent sections, but in short, connecting players requires a special module called Online Subsystem Steam (or any other online subsystem, Unreal supports Google, Apple, Amazon and many more), replicating players' actions is not an easy task too, but this is not a topic for this paper, everything is available on GitHub and I highly recommend the reader visits at least the mentioned repository for the main project.

3.2 Second Investigation

Second project that I created while learning the necessary material for the final one is this:
https://github.com/pyskonus/KrazyKarts
It contains much more detail about multiplayer that I will shortly cover in the following text.
This project was specifically aimed at dealing with high-latency connections. It was developed by solving the simulation errors one by one and implements several complex mechanisms of reproducing another player's actions and properties correctly on the local machine.

The first problem was about client's position absolutely not corresponding to it's position on the server. Here you can see cars starting the game and the on the next screenshot they are some seconds into the session (screenshots taken from a course website because on my instance of the project these problems have already been eliminated. Illustrative purposes only):

FIGURE 3.7: starting the game - locations coincide

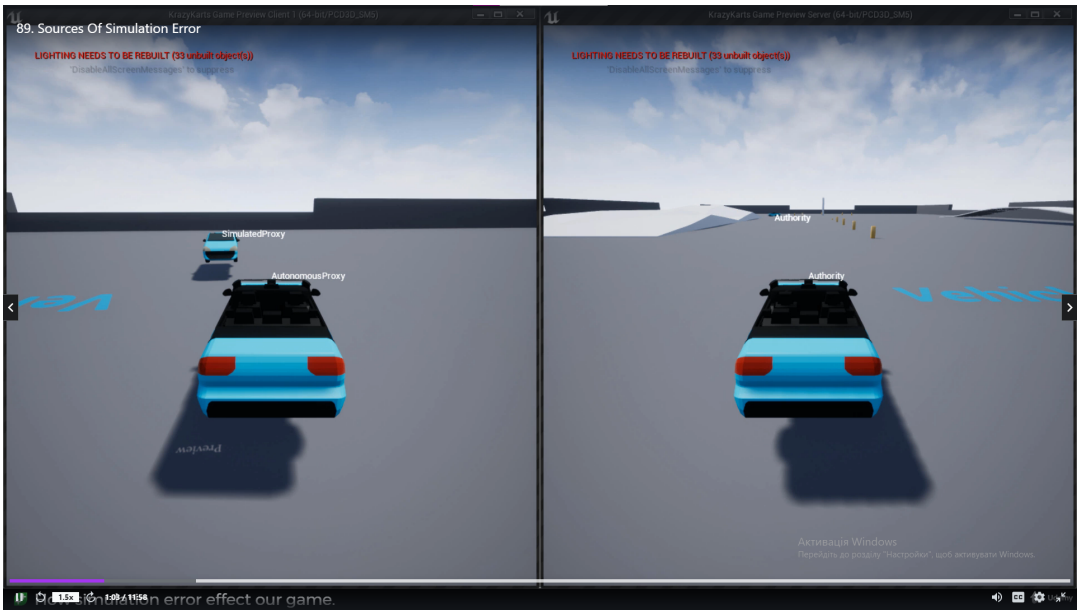

FIGURE 3.8: Not long after start problems emerge

Replicated actor location is calculated by integrating its function of speed over time, which, in turn, is obtained by calculating speed from acceleration, which is basically the user input.
This error is rooted in differences during integration, which means

1. since travelled distance is update on each frame, different time periods between frames result in different results

2. what is more important: since the lag, client's version of the game is updated only when the server sends the simulated results back two-times-the-latency seconds after the client sends its input to the said server

3. rotation input also contributes a lot to the observed error

The ultimate solution to this divergence between server and client is the following: sending not only the current input, but also periods of time between frames to the server in order to eliminate the integration error, and then, only if client's position is deemed inappropriate, update it from the server.

3.3 Final product

So, it is finally time to integrate almost all of the material used above and describe what the developed final application looks like. It is a representation of the classic Battleship tabletop guessing game.
The game takes place on a flat rectangular horizontal surface, which is detected by the Google ARCore framework and chosen by the user's tap. After this, the classical guessing game takes place in the "real" world.

After the game opens, it displays an already familiar host button and a list of other hosts that the player can join to. Joining sequence is initiated by tapping on one of the available host rows in the scrollbox:

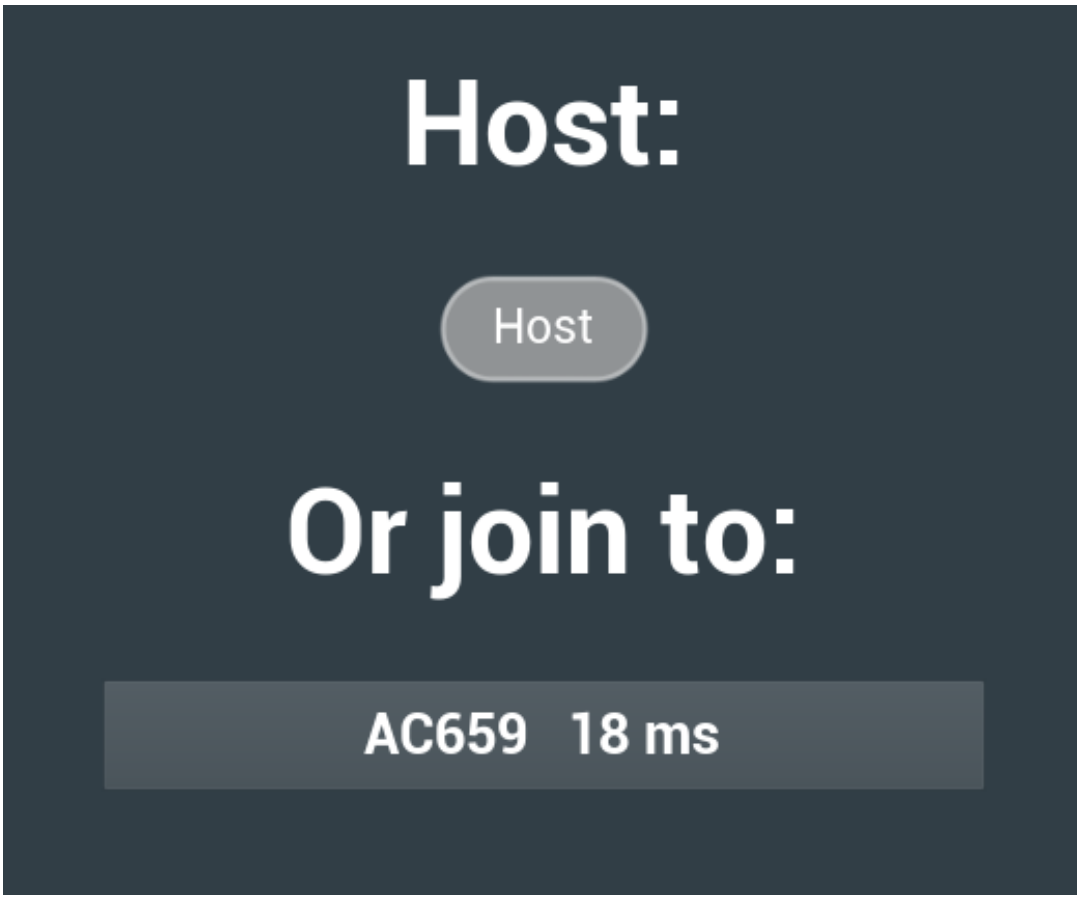

FIGURE 3.9: Final joining system on a mobile device

When players join into the world, the host chooses a plane and the game starts:

FIGURE 3.10: Selecting a plane

The code of the **final** application, developed in the scope of this thesis, can be found here:
https://github.com/pyskonus/NavalBattle

This project replicates the battlefield grid - a 2-dimensional array of each cell's states. Whether the cell contains a healthy ship part, a destroyed one or no ship at all. "Replicates" means this data must be synchronized between server and client. To avoid needless traffic usage, cell replication triggers on change only.

Some of the resources included in the project like the UI style and BP_Plane class have been copied from the official EpicGames AR template. It contains bare minimum of the code and project settings needed to successfully launch and AR project. This does not imply that I do not understand how the copied parts work. Some things just do not need to be reimplemented.

3.4 Mesh Slicing

This is a description of a task, or a function of my app, if you will, that I did not succeed to implement due to lack of understanding of UE API, but certainly plan to do so in the future updates. Still, a considerable amount of time was spent on investigation, so I will shortly describe the problem at hand here and in the following sections.
To begin with, let us define what is a mesh. It is a notion that is commonly used when describing objects in virtual worlds. A mesh is a digital representation of a solid object. Meshes are what can be drawn on the screen using 3D graphics and collided with using physics engines.
To draw a mesh, the software must establish a correspondence between parts of said mesh and pixels on the screen. This is done by dividing the simulated body into simplest 2-dimensional shapes - triangles. Even the most up-to-date virtualised geometry software uses this approach. The more triangles - the more detailed the picture is, because the main point is that every triangle must be drawn in a single color. This is imposed on the rendering software by how GPUs work.
So, why did this project need mesh slicing? The thing is, one additional function was planned to be implemented which is the radar. It had to look like a sector being cut out of the water bodies shown before and moving rapidly. There are some additional details about why this function would not be regarded a cheating and why it would come with a cost to its user, but let us skip this discussion.

3.4.1 Approaches

So, why is slicing a mesh complicated. A mesh, as stated above, is a collection of triangles, and when it is sliced or deformed, these triangles must be rearranged, often at runtime. as stated in Schmidt, 2020 , there are several approaches to this task:

- FDynamicMesh3/FDynamicMeshBuilder
- UStaticMeshComponent
- UProceduralMeshComponent

I also tried using the BSP geometry, provided by Unreal Engine mainly for level design purposes, but also, at first glance, perfectly suiting my needs since it implements *subtractive* geometry. Subtractive geometry is the first notion that comes to mind when one needs to cut out a part of a rigid body. But, as it turned out, BSP is not suitable for runtime tasks, so I needed to check the three options above. The one that suited me the best was BProceduralMeshComponent, since it easily deals with runtime fast-updating geometry which is exactly what this project needed. But, as it turned out, it is either quite hard or maybe impossible to use it in AR environment. Every way to so this that was explored during developing this project resulted in a failure.

Chapter 4

Conclusions and future work

4.1 Conclusions

In conclusion, AR is a suitable technology for multiplayer tasks. While synchronising players in AR world seems to be a hard task, multiplayer game in AR are possible and more complex activities like session-based AR communication between people is worth investigating.
Each of the two mentioned technologies are hard to master on their own, let alone their combination. But the more work will be put into this investigation, the sooner the results will emerge.

4.2 Future work

I do plan to learn multiplayer in greater detail, there is a lot of material to study.
On top of this, as mentioned earlier, I still did not succeed to implement proper mesh slicing functionality in this project. As it turned out, there is much more to mesh rendering and especially deformation than one can see at first glance. But with the emergence of technologies like Nanite and Lumen (launched with UE5) more and more engineers feel the motivation to study the complex field of computer graphics, me included.

Chapter 5

Description of used resources besides bibliography

The following materials were used mainly as theoretical background and guides to understanding the framework APIs and structure of the code libraries.

- https://docs.unrealengine.com/5.0/en-US/networking-and-multiplayer-in-unreal-engine/ - guide to networking and replication in Unreal Engine
- https://partner.steamgames.com/doc/sdk/api and https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Online/Steam/ - not used very often, official documentation on how OSS works. Used mainly for establishing connection between players.
- https://www.youtube.com/watch?v=qx1c190aGhs - a lecture on mesh drawing pipeline for UE
- https://www.pluralsight.com/blog/film-games/understanding-uvs-love-them-or-hate-them-theyre-essential-to-know - a tutorial on how UVs work. UV is a coordinate space that determines how a texture is applied to the mesh surface. Needed to understand the mesh drawing pipeline.
- https://www.youtube.com/watch?v=1zJM1gKoU14 - an entry-level tutorial on mesh slicing
- `https://cedric-neukirchen.net/Downloads/Compendium/UE4_Network_Compendium_by_Cedric_eXi_Neukirchen.pdf` - a detailed, but compact compendium on how UE multplayer works
- https://forums.unrealengine.com/ - the official community, used very frequently, stackoverflow for UE developers

`https://advances.realtimerendering.com/s2021/Karis_Nanite_SIGGRAPH_Advances_2021_final.pdf`

absolutely not essential, but provides an understanding on how supercomplex meshes can be effectively drawn at runtime. "Supercomplex" means mentioned triangles often take 1 pixel of screen space.

Bibliography

Alizadeh, Peyman (2015). "Object Distance Measurement Using a Single Camera for Robotic Applications". In: *None*. URL: https://www.mdpi.com/2073-8994/12/2/324.

Bradley, Milton (June 1967). *Battleship(game)*. https://en.wikipedia.org/wiki/Battleship_(game).

EpicGames (2022). *Actor Replication*. https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Networking/Actors/.

Grady, Leo et al. (May 2012). "Random walks for interactive alpha-matting". In.

Kumar, G Ajay et al. (2020). "LiDAR and Camera Fusion Approach for Object Distance Estimation in Self-Driving Vehicles". In: *Symmetry* 12.2. ISSN: 2073-8994. DOI: 10.3390/sym12020324. URL: https://www.mdpi.com/2073-8994/12/2/324.

Markopoulos, Evangelos and Mika Luimula (Apr. 2020). "Immersive Safe Oceans Technology: Developing Virtual Onboard Training Episodes for Maritime Safety". In: *Future Internet* 12, p. 80. DOI: 10.3390/fi12050080.

Neshov, N and Agata Manolova (Jan. 2021). "Objects distance measurement in augmented reality for providing better user experience". In: *IOP Conference Series: Materials Science and Engineering* 1032, p. 012020. DOI: 10.1088/1757-899X/1032/1/012020.

Parker, Eric and James O'Brien (Jan. 2009). "Real-time deformation and fracture in a game environment". In: *Computer Animation, Conference Proceedings*, pp. 165–175. DOI: 10.1145/1599470.1599492.

Schmidt, Ryan (Oct. 2020). *Mesh Generation and Editing at Runtime in UE4.26*. http://www.gradientspace.com/tutorials/2020/10/23/runtime-mesh-generation-in-ue426.

VantageMarketResearch (n.d.). *AR/VR Estimated Market Size*. https://www.globenewswire.com/en/news-release/2022/04/11/2420021/0/en/105-Mn-Augmented-Reality-AR-and-Virtual-Reality-VR-Market-is-Expected-to-Grow-at-a-CAGR-of-over-35-7-During-2022-2028-Vantage-Market-Research.html. Accessed: 2022-04-11.