VRIJE
UNIVERSITEIT
BRUSSEL

FINAL REPORT: REMOTE BATTLESHIP

Next Generation User Interfaces

Stijn Desloovere, Anisha Sachdeva (Group 6)

May 29, 2022

Sciences and Bio-Engineering Sciences

Contents

Introduction

This document contains the final report for the project of the course Next Generation User Interfaces. For this project, an interface had to be developed that provided a new and innovative way to interact with digital content. Our group decided to create an Augmented Reality version of the popular board game "Battleship" that also incorporates tangibles. In what follows, a more in-depth overview of the application will be given, followed by a discussing about the requirements. Furthermore, the design and features will be discussed. After that, some more implementation details and challenges are mentioned followed by an overview of the user study we conducted. Lastly, a small section with future work and the conclusion are provided.

Problem statement

The recent events of the pandemic made it hard for people to physically meet up. This let people to put some of their hobbies on hold, like for example, playing board games. While fully digital alternatives to board games do exist, it's usually not the same as playing with real, physical, pieces. We wanted to tackle this problem by creating a version of the board game Battleship which can be played with real pieces that represent the ships, but also has a digital component that makes it possible to play against other players via the internet. For this application, we want to keep as much of the original physical interactions with the game as possible. This way, user get the best of both worlds as they can play with a physical version of the game without having to be in the same physical space as their opponent. A motivation for the use of Augmented Reality in games is given in a paper by *Nilsen et al* [1] while we also kept in mind some of the guidelines related to the creation of AR games provided by *Wetzel et al* [2].

Requirements

Before starting the implementation of the project, we had to define a few requirements we wanted to meet with our program.

3.1 Functional requirements

First of all, the functional requirements which define everything a player should be able to do in order to play a round of the game. Before a game can be played, the player should be able to create an account with their username and a password of their choosing. After that, users should be able to create a room for the multiplayer session they want to start. Once a user is connected to another user, they can start a game. When the game has started, both players should be able to create their board setup with physical pieces that represent the ships. When the setup is created, the system should recognize the positions of the ships compared to the grid. Now, the players should take turns attacking the other player's board. To make the interaction more natural, an attack should be called via a voice command. When a square is attacked, this should be visualized in the player's grid as well. Both players should be informed about the outcome (hit or miss) of the attack as well. A second grid should be available for the player to mark which squares of their opponent they have already attacked. When all ships from a certain player have been destroyed, the other player should be announced as the winner.

3.2 Data requirements

Since the application doesn't have to keep track of or use much data, the data requirements are rather limited. In general, the only data that needs to be saved are the login credentials of the users. This can be done in a simple text file or an array.

3.3 Environmental requirements

The environmental requirements mainly have to do with the physical setup that is going to be needed to play the game. It is recommended for the users to play on a flat, monotonous, surface with enough space to place the grid and ships. Additional hardware pieces could be used to help keep the phone in a good position that gives an overview of the board.

3.4 User characteristics

The user group for this application is very broad since all that is required is a basic understanding of the rules of Battleship and enough technical knowledge to setup the application. In general, normal people should be able to make use of the application without many issues. The app is not focused on advanced users.

3.5 User experience

In terms of user experience goals, the idea is to provide a smooth experience for players to play a remote version of the game Battleship in Augmented Reality with tangibles. In order to do this, interactions with the application should be as intuitive as possible for the players.

Design

This section will go into more detail about the design of the application. Every feature will be discussed individually in the order that the user encounters them while playing the game.

4.1 Login and registration

The first thing the user will see when they boot up the application is the login screen which can be seen on figure 1. Here, they can enter their username and password to start playing. Of course, when the user doesn't have an account yet, they will have to create one first. This is something they can do by pressing the "go to registration" button. Doing that will bring them to the registration screen which is visible on figure 2. This screen will allow users to enter a username and password of their choosing they want to use to login again next time. Important to note here is that no dedicated database was installed for this feature. This means that login credentials will be forgotten between play sessions. The main reason for this was that installing such a database is somewhat time consuming and is less relevant to what we wanted to show with this application.

4.2 Creating a play session

When the user has logged in, they will be redirected to the game room screen. This screen can be seen on figure 3. The purpose of this screen is to either create, or join a play session. A user can create a room by entering a room name first in the top right text box. After that, when they press the "create room" button, the room will be created and they player will enter the next

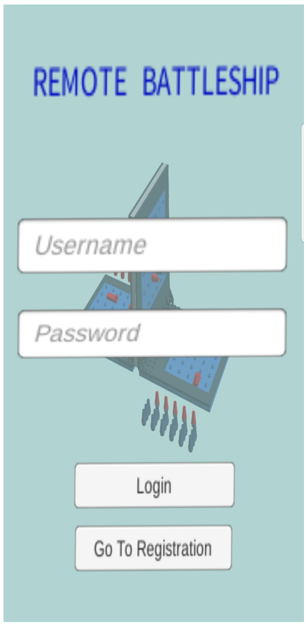


Figure 1: Login screen where users can login with their existing account.

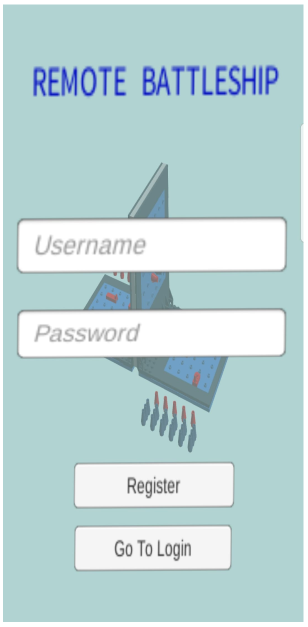


Figure 2: Registration page where users can create a new account.

scene where they are asked to create their ship setup. Apart from creating a room, there's also the option to join a room. This can be done by entering an existing room code in the bottom right text box followed by pressing the button "join room". This means that one player should always create a room and give the room code to the player they want to play against. This second player can then join the room with the same code and will also go to the scene where they are asked to create their ship setup.

4.3 Placing the grid

Once the player has joined a room, the Augmented Reality part of the application will start. This means that the camera will be activate and the application will start recognizing images and objects in the physical space. The first action which is required is placing down the grid on which the ships will be placed. This is done with a special marker with a buoy attacked to it as is shown on figure4. The user can then move this marker to a flat space in their surroundings on which they want to play the game. Once a suitable position has been found, the "Set Grid" button can be pressed to spawn a digital grid on the position of the buoy marker. The marker can now be put away.

4.4 Placing the ships

Once the grid has been placed, users can now start to add their ships. This is done by placing the tangible ships on places that correspond to squares on the digital grid. On top of the screen the position of the middle square is displayed to give players some kind of feedback as to where the ship will be placed. The position which is shown is the middle square for ship with an uneven amount length and the one when you divide the length by two and do it plus one for the ship with an even length.

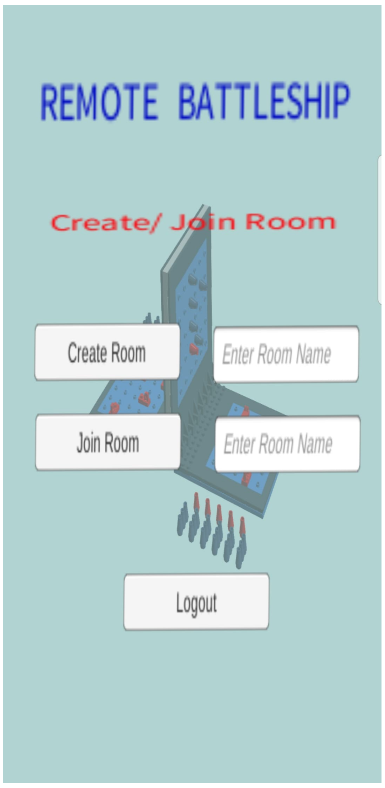


Figure 3: Create/Join room page. Users can either create their own room on this page or join an existing room.

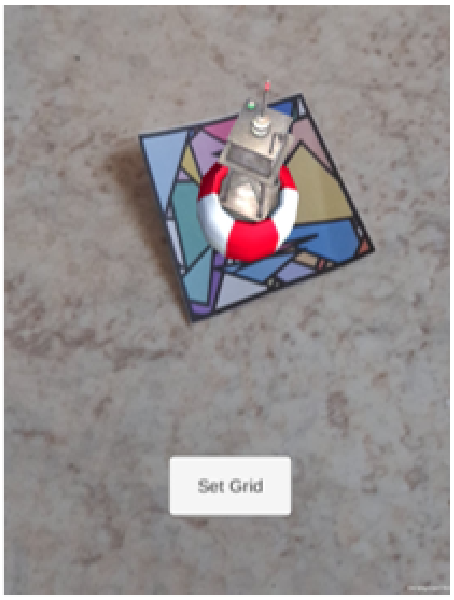


Figure 4: Marker which can be used to put down the digital grid somewhere in the physical environment.

Ships can be added with the “Add Ship” button and are added in order of their length. This means that the users will first have to place down the ship of length 2 and lock its position. Afterwards, they will have to place and lock the ship of length 3 and so forth until all ships have been placed.

4.5 Taking turns

Once the ships are placed, the ’Add Ship’ button is then changed to the ’Ready’ button. The players can then press the ready button. The one who created the room is by default the first player to take the turn and give the attack command. The first player would then press the speak button and a google pop-up would appear to listen. The attack command would then appear on the white text box which is placed just above the ’Ready’ button. Initially this white text box would contain the text ’Listening..’, this text then would change as soon as the player would press the speak button and speak. The player can say the attack command in any manner they want, example, ”I want to attack 7A” or ”Hey, I am the first player and I want to attack 7A”. The only important thing here is that which ever box the player wants to attack should be spelled like a number followed by a letter. Otherwise, the structure of the sentence may vary. Once the player checks the message on the phone screen and wants to send it to the opponent, he can press the ’send’ button. The same message will then appear on the opponent’s screen. Here the system will extract the position from the attack command and validate it. If there would be a ship on opponent’s board at the attacked position, there would be a voice output saying ”Oh no, it’s a hit”, else if there won’t be a ship at the attacked position then there would be a voice output as ”Yay, they missed”. Along with the voice output, the player’s grid will be colored. In case of a hit, the attacked position on the AR board would be colored as red and in case of a miss, it would be colored as green. The player whose all ships would sink first would lose the game and the other would win. In our scenario, the player whose all boxes on which the ships are placed will be colored red first will lose and the other would win.

Implementation

This section will discuss the implementation details of the project. Some prototypes will be discussed and a general overview of the architecture will be given. We divided the workload as followed, Stijn Desloovere focused on the AR related parts which include implementing a way to place down the grid in the environment as well as making it possible for the system to recognize the positions of the ships relative to the grid. Anisha Sachdeva mainly focused on implementing the login and registration functionality together with setting up the multiplayer environment including the voice interaction feature. She also took care of the game loop. Near the end of the project, we worked together on any remaining issues related to the multiplayer part and the game loop. Important to note is that our, initial, third group member (Ameer Hamza) was first assigned the task to take care of the multiplayer part. However, he dropped the course unexpectedly a few weeks before the final presentation. This forced us to change our focus in order to try to implement his part of the work before the final deadline while also making some trade-offs and scrapping certain features we had originally planned to implement.

5.1 Augmented Reality

In this section, the implementation details of the Augmented Reality(AR) part of the project will be discussed. First, some prototypes are discussed which have been implemented and tested, but didn't make the final application since they had some issues. Next, a description is given on how the grid placement and ship position assignment work.

5.1.1 Prototypes

For the Augmented Reality part, and more specifically the grid detection part, a few prototypes have been created and tested. The first idea was to let the application recognize the image of the physical version of the board and this way calibrate the digital board and display it perfectly above the physical board. This method, however, had the very clear issue that the board itself is too simple to be recognized by the application.

The second prototype tried to circumvent this issue by introducing 2 image markers that could be used to mark the corners of the physical board. This way, the application would have a rough idea of where the physical board is in the space without having to directly recognize it. While this fixed the issue of the first prototype, it also introduced some other issues. The main problem was based upon the fact that the system could not accurately determine the position of a ship compared to the grid that has been set. The reasons for this problem could be that the markers where not placed on the exact spots of the corners, combined with the fact that the positions of object extracted via Vuforia (The package responsible for the AR) might not be 100% accurate as well.

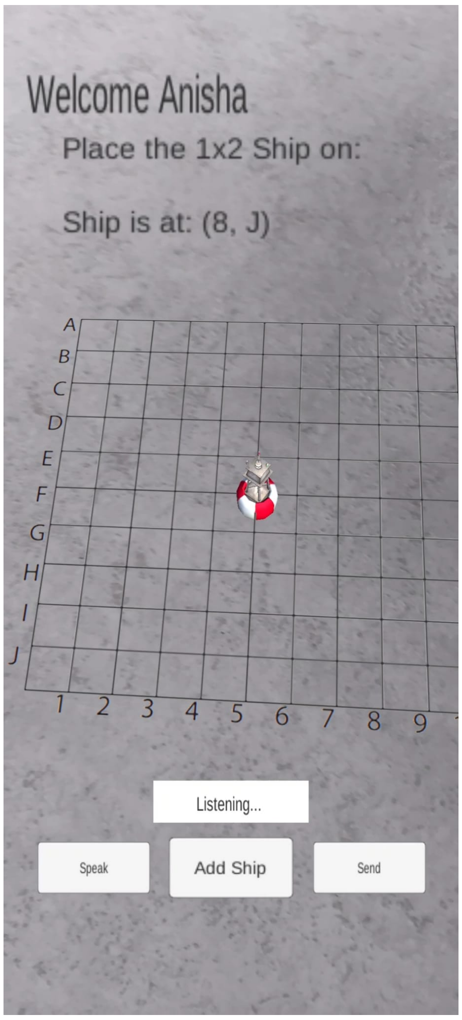
In the end the solution of a purely digital version of the grid was chosen. While this solution removes a tangible element from the experience, it was the most practical one to use in this scenario, since it does allow for fairly accurate determination of the positions of the ships. Furthermore, it also circumvents the issue that the digital and the real grid would have to align perfectly in order to not create the confusing effect of 2 grids overlapping. For the AR functionalities, the focus mostly lied on the use of image markers. These are special kinds of images with features that are particularly easy to recognize by the algorithm. An online generator [1] has been used to create the markers.

5.1.2 Final solution

A marker was used for the grid placement interaction. Important to note is that Vuforia maps Unity positions to the real world. This means that, while the markers are being tracked, they have a Unity position related to them that can be extracted. When the system recognizes the "place grid marker" it will track it and display a buoy object along with it. Once the "Set Grid" button is pressed, a digital layout of the grid is displayed and centered around the current position of the grid marker as is shown on figure 5. Important to note as well is the fact that the grid will be perfectly aligned with the x and z-axis of the coordinate space Vuforia mapped out the moment the camera was activated. This will make the mapping from a ship's position to a grid square much more easy and accurate. This marker can now be put away.

Once the grid is place, the "Set Grid" button will transform into the "Add Ship" button. Now, the player is asked to place their ships onto the grid in order from the shortest to the longest ship. These ships are tangibles in the shape of a rectangle made from metal and which are tracked by images markers that are glued on top. Since the ships are rather small (2cm wide), we opted to add more than one image marker to each ship. Because the surface of the ship objects is flat, the whole length of the ship tangible could be used for image markers. In the end, this resulted in the smallest ship (2x1 squares) being the tracked somewhat inconsistently well, while especially the longer ships have much better tracking. Each ship also has a unique 3D model attached to them so the "ugly" image markers are hidden on screen while also adding

[1]https://danilogr.github.io/AR-Marker-Generator/


Welcome Anisha

Place the 1x2 Ship on:

Ship is at: (8, J)

Figure 5: Digital grind placed on a physical space. The grid is centered around the grid marker when the “Set Grid” button is pressed.

some extra level of immersion to the game and being a way of showing the user that the tangibles are in fact being tracked. When the “Add Ship” button is pressed, the system should map the position of the ship to squares on the grid. This is done by extracting the top-left and bottom-right coordinates from the sprite that represents the grid and was placed in the previous step. These two position can now be used to create a grid in memory that can map any position in the space to a square on the grid. By knowing that a grid is 10 squares wide and 10 squares tall, the difference between the x-values of both corner positions can be divided by 10 to get the width of a square. The same can be done with the height on the z-axis. With this method, usually the middle of each ship is tracked. For ships with an uneven length, this is ideal since the middle also perfectly overlaps with the middle piece of the ship. For ship with an even length, their position needs to be shifted down a little so the position of piece that is just below the middle is being tracked (To make this more clear, a ship of length 5 occupies 5 squares on the board. Here, the middle square, number 3, could be used as its position. For a ship of length 4, the middle falls right in between the second and third square it would occupy. Its tracked position is therefore shifted a bit downwards to the third square). Since the ships have to be placed in order, the system always how many squares each placed ship should occupy. Therefore, by extracting the rotation of the ship object, the system can automatically fill in all the square the ship should occupy on the board by effectively only tracking 1 single position per ship. When all ships are placed, the “Add Ship” button transforms into a “Ready?” button that can be pressed to indicate that player has created their board layout and is waiting for their opponent to finish.

Lastly, it should be mentioned that all 3D models which have been used were taken from the Unity Asset Store.

5.2 The Multiplayer Game

In this section, we discuss the functionalities related to the Android game which we developed using Unity. We start with the basic functionality like login and registration and move towards more complex ones like multiplayer aspect and game loop. Issues, testing an implementation are discussed for each of the functionality.

5.2.1 Login and Registration

We started to make the game application by enabling the basic functionality to authorize a user i.e. the login and registration functionality. The general idea here was that a user should first be authorized before starting the game.

We began with designing a registration scene in Unity which included two input fields (to enter the username and password) and two buttons (one to confirm the registration and another to navigate the user to the login scene). Similarly a login scene was also designed. It also included two input fields (for username and password) and two buttons (to login into the game and another to go back to the registration page, if by chance a user is still unregistered). Here, we first wrote our C# script which defined how a game object (username input field, registration button etc) should behave. Herein we wrote the code according to which a user could register himself on registration page and their credentials will be stored in a text file on the local system. If there would be no user yet registered meaning no credentials file would exist, then automatically a credentials text file would be created. The username and password were stored in the form, ’username::password’. While logging in, the text file was referred to check the username, password pair in order to authenticate the user. This login functionality code was inspired from a YouTube tutorial[2].

[2] https://www.youtube.com/watch?v=SKCenhe_Osk&t=493s

However, we soon realised that the text file was being well referred if we were testing the application in Unity editor but while we were building the application in android, the text file could not be read. The reason for this behaviour was that while building the application, the build is sent as a compressed APK file to the android and as a result the application cannot access the data path for the credentials text file saved in a local system. Either we needed an additional software to see inside the jar file and obtain the required text file or the text file needed to communicate over the network. Looking for an additional software was not really a good option as the focus of the application was not on a very secured database for the login, so as a solution mentioned in the Unity documentation we tried using UnityWebRequest[3]. While trying to look for how exactly UnityWebRequest works, there were numerous of issues faced. Some online examples suggested to use WWW (UnityEngine.WWW), which is now an outdated unity functionality. Some other websites did suggest to use the UnityWebRequest, but we could not find a suitable example to look at how exactly UnityWebRequest works. There are a few examples mentioned in the Unity documentation but they were also not very clear in regard of what parameters do we use and how. An an example, in the documentation it is mentioned we can use the get request with the declaration 'public static Networking.UnityWebRequest Get(string uri)', but it lacked to inform what should be there in URI. Hence, after all the searching and experimentation we decided to go with an array list.

For the android phone application, the username-passwords were stored in an arraylist meaning that they were applicable only for the current session. As one session would end, the stored credentials would also get deleted and user would require to register again.

We had another functionality to Logout of the application which appeared on the 'Lobby Scene'. On logging out, a user would be navigated to the 'Login' page wherein he could login again using valid credentials.

5.2.2 Multiplayer Environment

To setup the multiplayer environment, we started by exploring different APIs which could be used to send messages across network. Two main APIs that we found were, the built-in Unity Networking Service and the other was Photon plug-in. Unity Networking Service is all built-in and provides different classes and components for a multiplayer set-up, however on some research we got to know it does not support all the multiplayer features, like host migration. Though we would not have needed host migration for our project as it is a two player game but we thought of going with something which has better documentation and explanation over internet. Hence, we went forward with the Photon API[4]. Photon is a real-time game development framework. In Photon there is Photon Unity Network (PUN) which is Unity specific providing services such as matchmaking, components to synchronize GameObjects, remote procedural calls (RPCs). The most crucial elements for our project were the matchmaking and sending messages over the network using RPCs. Both the conditions were full filled pretty well by PUN2.

To setup a multiplayer environment for two players, we downloaded the PUN in Unity's asset store and set up an AppId. AppId is basically an identifier for Photon cloud application. Next, we used the 'PhotonNetwork.ConnectUsingSettings()' method to connect to the Photon server. Once a user was connected to the Photon network, they would be redirected to a "Lobby" scene. In a lobby, to play the game one could either create a room or could join a room. Using the concept of rooms, it made easy for us to setup two players in the same environment. Joining the same room to play the game was the matchmaking process in our game. Again, we used the methods provided with PUN to create (PhotonNetwork.CreateRoom()) and join rooms

[3]https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html
[4]https://doc-api.photonengine.com/en/PUN/v2/index.html

(PhotonNetwork.JoinRoom()). A little detail covered here was that a room's name could never be empty (PUN allows it to be empty).

Once both the players, enter the same room they could start the playing functionality.

5.2.3 Voice Interaction

Voice interaction was another interesting feature of our game. To start with we used the Google TextToSpeech and SpeechToText API. In the starting we had two buttons, one was for speaking (the grid position to attack on) and the other one for listening (to know it the attack was a hit or a miss). When a player would press the "Speak" button, a google pop-up would come to listen to the command and the same would be displayed on player's respective screen. The player could then press the "Send" button to send the attack command to the opponent.

However, with this layout of the voice interaction we received a feedback that pressing the "Speak" button every time to speak and give the attack command would restrict the users from having a 'hands-free' experience. Ideally a user should be able to play the game without touching the phone. To enable the users with this ideal 'hands-free' experience, we could use a 'keyword recognizer' like "Hey Battleship" in our game. The voice listening API could run in background and as soon as it would hear "Hey Battleship" (or any other specified particular phrase), the pop-up to speak should be activated. To enable this feature we explored a few options. One option was to use Unity's inbuilt Keyword Recognizer (UnityEngine.Windows.Speech) but this built-in function is just valid for the Windows platform which would not help us enable the functionality on Android. As a result we dropped the idea of Unity's in-built keyword recognizer and looked for another interesting option which was, IBM Watson Speech recognition API. Though IBM Watson's speech to text API provides an option for keyword recognition, but still we could not use it as it was a paid service. The trial version did include some minutes to test the application, however we were required to fill in the bank account details even for the free trial. So, we opted out IBM Watson option as well.

We came back to the good old Google speech recognition API. As Google's API do not provide with the keyword recognition functionality, we thought of enabling the Google listening pop-up automatically whenever a player should be required to speak something. For example, as soon as both the players would be ready, the first player would get the google pop-up to speak. The speech recognizer will then compare the speech with the valid command, which would be the word "Strike" followed by a digit from 1 to 10 and a character from A to J. For example, "Strike 6H". To give the hands-experience, it would listen until a valid command is recognized by a system. As soon as a valid command would be recognized, it would appear on the player's screen and then the player could send the command to the opponent. However, a number of things could go wrong here. As an example, what if a player wanted to say "Strike 6B" and the recognizer would recognize it as "Strike 6D", there would be no possibility for the player to speak and give the command again. Also, when the google pop-up to listen appears, it takes all the space on the screen and that pop-up coming again and again on the screen until it recognizes a valid command could be little irritating to deal with.

Hence, we returned to the original solution of having a speak button. This would not only enable the user to speak as and when required but if the user's attack command is somehow recognized incorrectly, the user could speak again by pressing the button.

Interesting to mention here would be that though we implemented the speech recognition in our project but it would may be a better idea to have a chat functionality instead of speaking at all. The reason for this statement would be that right now Google speech recognition API uses some dictionary to recognize the words, however in our attack commands we generally need to speak about positions to attack like '3A' or '6D' which necessarily is not a word and it becomes difficult for the recognizer to recognize the commands correctly and efficiently at once. The API would try to match '3A' or '6D' to some word (in predefined dictionary) and show us incorrect results, which we would not wish for.

After each attack, the player could automatically listen if the attack was hit or miss. Google TextToSpeech API was used to implement the same.

As a small side remark, whenever someone would install the application in their android phone, on first installation, the application will ask for the permission to use the microphone of the phone. The request has to be allowed in order to use the voice functionality in the game.

5.2.4 Game Loop

To enable the game loop, we again used some functionality from Photon Unity Networking 2. In PUN2, there is PhotonView[5] which is identified as an object across the network with a 'ViewId' of itself. Next, there is a boolen 'IsMine' attached to the PhotonView which is always true for the owner of the room and false for others. We used this functionality to enable the game loop. As our game is a maximum two player game, the one who would create the room to play the game will be the first player with PhotonView.IsMine as 'true' and the other player who would join the room will be the second player with PhotonView.IsMine as 'false'. We start with the first player where the 'IsMine' value is true, that player's speak and send buttons are enabled. As soon as the first player, sends the attack command and the application checks and tells the opponent if it is a miss or a hit, the second player's (whose 'IsMine' is false) speak and send button will be enabled. The loop follows in order to have the multiplayer interactions.

Further, we used the Remote Procedure Calls(RPCs) provided by PUN2 to communicate the attack messages over the network. RPCs were used to send the attack message on the screen of the opponent whenever the player would press the send button.

5.2.5 Winning the Game

In our scenario the player whose all boxes containing the ships would be colored red first will lose and the other player will win. As our third member dropped the course just two or three weeks before the final submission date, we were short of time to implement this functionality. We could not implement it but the general idea of the implementation would be as followed. Till now we can correctly detect if the attack was a hit or a miss, in future in case of a hit we can check on the side of the player who was attacked, which ship was exactly hit. We could may be maintain an array to keep track of how many times a ship was attacked. In the beginning this array would be [0,0,0,0,0] since no ship would have been hit at the start. The array [0,0,0,0,0] would represent the 5 ships and could take the maximum counter as [2,3,3,4,5]. The numbers 2, 3, 4 and 5 represent the size (length) of the ships. Now we could maintain a counter for each ship and each time a particular ship would be attacked we could then increase the counter by 1. The first value in the array could reach the maximum value of 2, next as 3 and so on depending on the length of the ship. When the array of counters would become [2,3,3,4,5] we would get to know that the particular player lost and the other player would have won.

User study

We conducted a small scale user study to get some feedback of some of the interactions we created for this application. Since, at the time of the user study, not all functionalities had been implemented, the user study focused mainly on the Augmented Reality part where ships should

[5] https://doc-api.photonengine.com/en/pun/v2/class_photon_1_1_pun_1_1_photon_view.html

be placed on the digital grid as well as the voice command interaction for attacking other player's board. The results have been compiled into 4 bar charts which can be seen on figure 6. We had a total of 5 participants, all with good knowledge about technology and a basic understanding of the rules of Battleship. We asked each of them 4 questions which they could rate on the Likert scale. We also gave them an open question at the end about their general feedback for the application.

From the first question, about the intuitiveness of placing down the grid, we received generaly good feedback that this interaction felt quite natural. Likewise, the participants also indicated that using the tangibles (objects that represent ships) felt intuitive. When we asked whether placing the ships on the grid was easy we got some more neutral answers. This was mainly because, while the accuracy of this prototype was still the best, it sometimes happens that a ship is recognized to be one square higher, lower, to the left or to the right than the actual position it is placed in. This resulted in the participants having to make tiny adjustments to the position of the ship before finally locking the position. Lastly, we asked the participants if having voice input to call out attacks would be an added benefit. A few participants answered neutral while some said this would be a positive thing. Important to note is that we could not yet let the participants try out this feature yet, since the full game loop was not implemented yet.

Lastly, the open question. We summarized the answers into positive and negative feedback. As negative feedback, we got a few comments on how placing down the ships was not always accurate. This is definitely valid feedback, but it is also something that gravely depends on the AR library we used. There were also some concerns regarding the voice input as some people would find it annoying to always talk to their phone to do the attacks. Maybe, in the future, some other interactions could be used to call out attacks such as, for example, using another tangible and placing it on a certain grid in order to attack that grid. As for positive feedback, most people still liked to use the tangibles and found it a fun experience. Someone also pointed out that this was an interesting take on a classic board game.

Future work

While the application tries its best to show how a digital version of Battleship can work in AR, there are still a lot of features that are missing and improvements that could be made. This section discusses some of those.

First of all, a down side of the current application is that there is very little interaction with the tangibles once the game has started. It would be nice to introduce more tangible elements to the application. An opportunity to add some extra physical element is with the second grid the player uses to mark down the results of their attacks. Currently, a plain piece of paper with the grid layout can be used for this task, however, since the application also knows the results of these attacks, it could be translated to hardware component. The idea is to use a grid of LEDs with the same layout as the board. A small computer, such as a Raspberry Pi, could be integrated into the application to color the LEDs based on the results of the attack. For example, if a user attacks a square and they missed the ships, they would normally mark this down on their piece of paper. Now, the system could automatically color the corresponding square red on the LED grid.

Next, since this is a partly digital application, the phone interface could be used to give the players automatically generated tips while they are playing. This could help them improve their strategies. A small tutorial could also be integrated so newer players can learn the rules of Battleship via the application.

Additionally, a method for combining the digital grid with a physical version of the grid could

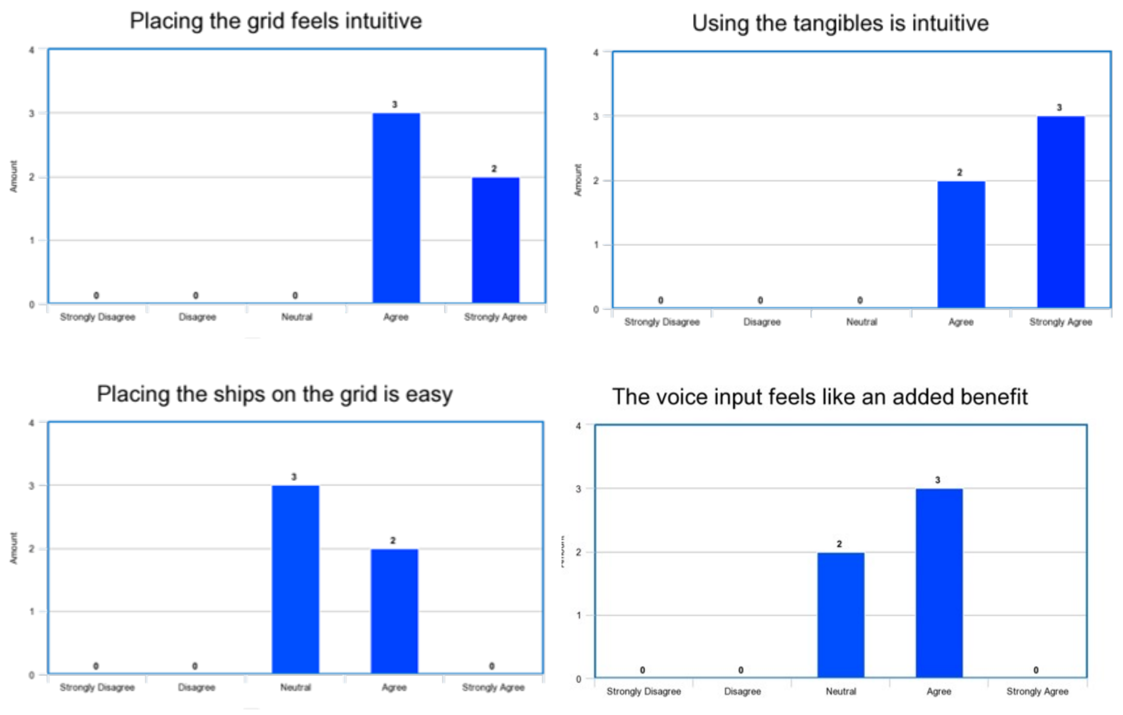


Figure 6: Results of the user study. A total of five participants have been asked to answer 4 questions each based on a Likert scale.

still be added as well. While it is currently hard to do given some limitations of the technology, it might become easier in the future.

Conclusion

We created an interactive Augmented Reality application that makes use of tangibles. The goal of this application was to let users experience the feeling of playing a real board game while not having to be in the same physical space as their opponent. In order to achieve this, we made sure to keep as much of the original interactions with the board game as possible. This includes using ship-like objects to create a board setup and using voice commands to attack the other player's board.

The source code for this project can be found on github at: `https://github.com/StijnDesloovere/ProjectRemoteBattleship`. A demo of the project is available on YouTube with the following URL: `https://www.youtube.com/watch?v=NyCTXpab3RI`.

(Note: In the demo, on one phone the microphone was not working (we suspect as it's an old phone so may be it would not support the microphone functionality) hence we hardcoded a attack command. You wont be able to listen to the attack command on Stijn's phone in the video. However, the voice interactions functionality otherwise work perfectly fine and can be verified during Anisha's POV in the demo video.)

References

[1] Trond Nilsen, Steven Linton, and Julian Looser. "Motivations for augmented reality gaming". In: *Proceedings of FUSE* 4 (2004), pp. 86–93.

[2] Richard Wetzel et al. "Guidelines for designing augmented reality games". In: *Proceedings of the 2008 Conference on Future Play: Research, Play, Share.* 2008, pp. 173–180.