Title of the project

Subtitle

Project Report

Group: SW805F20

Aalborg University
Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg East, DK

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9220 Aalborg East, DK
www.cs.aau.dk

Title:
Title of the project

Theme:
Mobility

Project Period:
Spring Semester 2020

Project Group:
SW805F20

Participant(s):
Andreas Stenshøj
Daniel Moesgaard Andersen
Frederik Valdemar Schrøder
Jens Petur Tróndarson
Rasmus Bundgaard Eduardsen
Mathias Møller Lybech

Supervisor(s):
Brian Nielsen

Copies: 1

Page Numbers: 55

Date of Completion:
May 28th, 2020

Abstract:

This is the best abstract ever written

Contents

Terms and abbreviations

Terms and abbreviations used in the report:

Pozyx : The hardware used for positioning.
AR : Augmented reality.
UWB : Ultra-wide bandwidth.
TWR : Two-way-ranging.
OSI : Open systems interconnection.
GDOP : Geometric Dilution of Precision
TCP : Transmission Control Protocol.
UDP : User Datagram Protocol.

Chapter 1

Introduction

1.1 Project idea

The idea for this project is to create a location-based competitive game using augmented reality (AR). Two teams will compete against each other to score the most goals using a ball. Each player will be equipped with a smartphone-based virtual reality headset, and these will display the playing field from a top-down 2D view. To achieve this, each player's position needs to be tracked as well as where the ball is located on the field. In the top-down view, each player needs to see the positions of the other players and the ball. They also need to see their position on the playing field and where the goals are. The players should be able to set a number of goals they need to score to win before beginning the game.

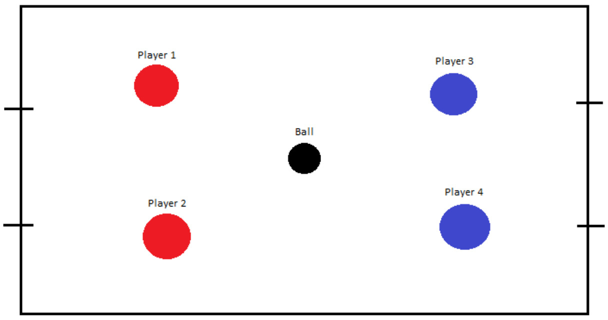

Figure 1.1: An illustration of the playing field

In Figure 1.1, an illustration of the playing field for the game is shown. There are goals at each end of the field, and the teams score goals by getting the ball between the goalposts. An alternative version of the game is suggested where, instead of goals at each end of the field, there would be virtual goal zones seen in the game which the teams need to bring the ball. These zones could even change locations as the game progressed.

1.1.1 Technical requirements for the project

Multiple pieces of hardware will be required to realize the vision of the game. First of all, each player must have a headset to hold their smartphone such that they can view a virtualized version of the playing field while playing. In order for the data to be synchronized between the players, they will also need to be equipped with a positioning device, which can transmit their location to the other players. This transfer of information will require a networking solution so that the virtualized playing field is synchronized between the players.

1.1.2 Problems to consider

The project idea proposes some problems that will need to be solved for the game to work. We will need to consider which technologies to use for the development of the visual aspect of the game which should show a top-down view for each player. As it is something we do not have experience with, it would be preferable if it is not necessary to have to build it from scratch. We will also need hardware that can track the positions of players and the ball. This must be accurate and update quickly such that the players do not run into each other, otherwise the game will not work. Another problem to consider is how the ball should be displayed in a 2D view. For the players to be able to find the ball on the field, it either has to be quite large to make it easier to find from the top-down view, or the game will need some metric to display how far the ball is from the ground. The game will also need to be able to track when the ball has crossed the goal line and then give feedback to the players. Another problem to solve is how to keep the positional data synchronized across all the players' devices, as it will be difficult to play the game without accurate data.

1.2 Essence

For the process of project development, we have chosen to work with Essence. Previous semesters we have worked with an agile approach inspired by Scrum, however, this semester we are attempting to apply the Essence approach. The basic idea of Essence is to encourage diverse thinking in the team, even though all members of our team share a similar background as bachelors in Software.

Essence uses two strategies to support value creation:

- *A systematic use of diverse viewpoints.* Values, views, and roles are used frequently in Essence. By using different views and roles to represent problems and solutions, Essence tries to facilitate a range of viewpoints on how a problem needs to be understood and solved.

- *A focus on idea maturation more than idea generation.* Essence applies the concept that ideas develop over time and tries to stimulate the team to evaluate and refine ideas [1].

1.2.1 Four variants of innovation

Essence tries to support innovation in software development, and hereby it defines four different variants of innovation, which are: [1]:

- *Product innovation* is new or radically changed software products or services.
- *Process innovation* is software solutions offering the user new or radically improved ways to produce products or services.
- *Project innovation* is fitting software solutions from earlier projects into new application domains
- *Paradigm innovation* is about software solutions coupled with changes in the mental model of what a business is, who the users are or what the market is.

1.2.2 Paradigms

There are two well-established software development paradigms: the document-oriented paradigm, which we know from the waterfall approach and agile paradigm which we know from for example extreme programming and Scrum. The author of Essence considers the new emerging paradigm called the pragmatic paradigm.

The document-oriented paradigm portrayed software developers as being document-oriented. The requirements are static and allow for a top-down waterfall approach to software development, which pays small attention to creativity and innovation.

The agile paradigm sees software development as user-oriented. Requirements are presumed dynamic as customers learn about options and constraints within the course of the project. This leads to incremental software development.

The pragmatic paradigm is problem-oriented. Systems are becoming more complex and it is more difficult to separate systems from each other. The amount of data, software libraries and hardware components available is steadily increasing, leading to hypercomplex software projects. Hypercomplexity is a degree of complexity that makes it impossible to make rational decisions within a reasonable time constraint. The most important features of this paradigm are that requirements are not completely known when the projects start. Ideas evolve in the process of the project, and during the project the requirements for the project are negotiable [1].

1.2.3 Core concerns

All software projects involve these four core concerns:

- Do we know what to build?
- Do we understand the solution?

- Do we understand the problem?
- Should we pivot or persevere?

1.2.4 Team organization

Within the team organization, in Essence, roles are used to create heterogeneity in teams, to ensure diverse points on views and to ensure cohesion despite diversity. The focus of these roles is to increase learning with personal interaction by sharing insights and experiences. The roles also ensure that the team understands the problem domain, and see potentials in the technology domain.

As a rule of thumb, the roles are persistent meaning that a member will have the same role for the duration of the project. The roles in Essence are compatible with agile software development, making it possible to combine Essence with other processes like Scrum.

There are four roles in essence:

- Child
- Responder
- Challenger
- Anchor

The role of *Child* can ask any questions and make propositions that are opposite of previous decisions. The rest of the team is not allowed to criticize the child, but they are however allowed to ignore the person's suggestions. The child is one of the main sources of ideas and other perspectives on the project. Outsiders are also allowed to take this role.

Responders are the developers in the team, and they are usually the majority within the team. Responders work closely together with the *Challenger*, so that the most important features are developed first.

Challenger is the customer or customer representative. The challenger can be compared to the *Product Owner* in Scrum. This role formulates and explains the Challenge, prioritizes features and accepts the solutions. There can be more than one Challenger, but if there are they must agree on the product vision.

Anchor is the one responsible for leading evaluations but does not decide the consequences. If necessary, the anchor can intervene and remove threats to the team's ability to develop ambitious responses. A potential threat could be something that results in productivity issues.

Chapter 2

Sprint 1

2.1 Sprint goal and introduction

The goal for this sprint is to explore the project idea and look for possible solutions to some of the problems that the project idea introduces. In this sprint, we want to learn more about visualizing the game for the users so they can see themselves in the game as well as the playing field.

We will look into using Pozyx to define the corners of the playing field as well as get the players' positions in an accurate way with fast updates, and to determine whether or not Pozyx is even viable for use in the context of this project an experiment will be conducted where we look into the accuracy of the positional precision of the tags. As this is the first sprint of the project, the main focus is to gain more knowledge about the various aspects of the project and to gain a shared vision amongst the group members as to how the game should work. This initial shared vision will be achieved by making prototypes of the game as well as architectural diagrams of how different components in the game will work together.

2.2 Our process

As described in section 1.2, a secondary focus of this project will be to attempt using the Essence process model as taught by the Software Innovation course. We have been working with a ScrumBut approach in previous semesters and at the beginning of this semester, but we decided that we wanted to try a new organization this semester, and as we learned more about the Essence approach we decided to pivot, changing our process.

However, we have chosen to keep sprints, stand-up meetings and retrospective meetings from Scrum as these can complement Essence. To make the report fit this format, it has been split into the 4 sprints that we will go through during the semester, where each sprint has a length of 3 weeks.

2.2.1 Roles

We have divided the roles that were described in subsection 1.2.4 between the members of the group. One person has the *Challenger* role, meaning that they are accountable for prioritizing the tasks in the backlog. The process of prioritizing tasks is further described in the following section. Another person has the *Anchor* role, and is responsible for changes to the process and is in charge of leading evaluations of the process. The rest of the group functions as *Responders*, which is the role for the developers of the project. The *Child* role fluctuates between members of the group. Everyone can add suggestions to improve an idea and give other perspectives on the project.
Due to our team size, the challenger and anchor will also work as developers during the duration of the project.

2.2.2 Prioritizing tasks

Our backlog is saved as a board on Jira, which can be seen on Figure 2.1. The leftmost column is the *Suggested* column. Everyone can make suggestions for tasks that they find useful for the project. After each stand-up meeting, the challenger will present new suggestions that seem relevant to work on in the near future. This presentation will include the definition of done, and all members of the group will then vote on how valuable it is for the project and how time-consuming it is. The priority of a task is then calculated as $reward - time$, which is an arbitrary number to indicate how important it is.

The challenger then chooses the most important features from the *Discussed* column, often based on the highest priority, as *Chosen for Development*. Responders then have the opportunity to take tasks from this column and move it to the next column *In progress* when they start working on it. When the task has been completed, reviewed and merged into the develop branch, it is automatically moved to the column *Done*.

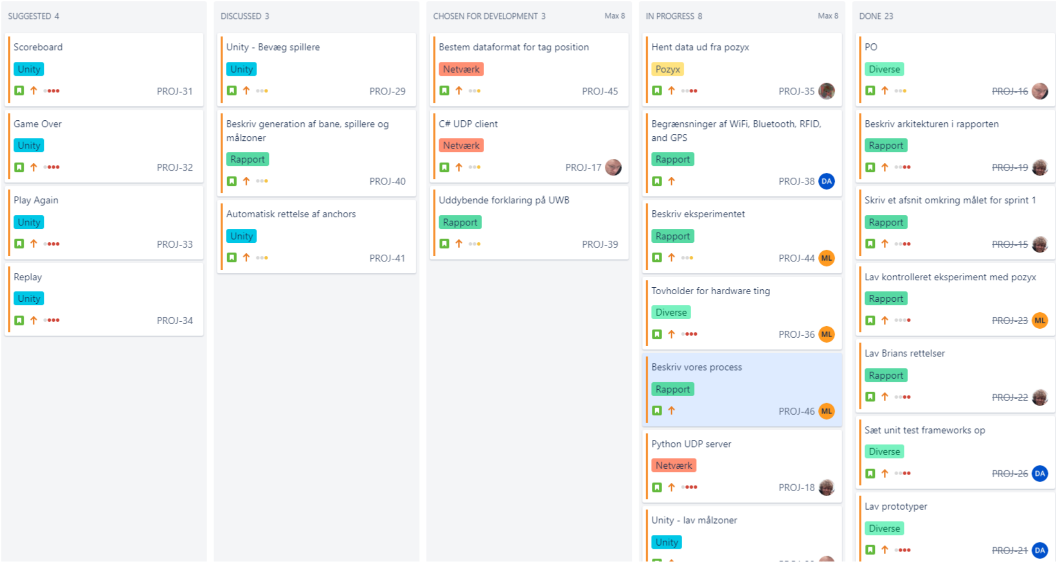

Figure 2.1: The board for tasks.

2.2.3 Pair reviews

To ensure quality and make sure that people follow the test plan, everything undergoes a formal review before it is added to the project. The test plan is our description and an agreement of what needs to be tested and how thoroughly it should be tested. More details about the test plan can be found in section 6.3. For report tasks, two reviewers are assigned to review and accept it before it can be merged. For coding tasks, two people are likewise assigned to review it, but the review has to be done as a pair, meaning that they have to physically sit together and go through the code on a shared screen. These pair reviews are a good way to share knowledge about the implementation through the group, as people have to understand it to be able to discuss it. In previous semesters people have reviewed the code separately, which leads to fewer comments as the code was not discussed between reviewers.

2.3 Prototypes

The first step in the project is to arrive at a common vision about the layout and functionality of the system. To accomplish this, a series of wireframe prototypes were created. A wireframe is a lo-fi prototyping technique where you create a grey box schematic that outlines the primary features of the user interface, unlike hi-fi prototypes which generally have a higher amount of detail and are used to test the interaction with the application. The primary focus of these prototypes is not to be used for implementation, but rather for comparing opinions about how the flow in the system should be created. Since the user interface is mostly focused on the mobile devices that the players will be wearing, it was decided that the host computer should simply have a text-based interface, as seen on Figure 2.2.

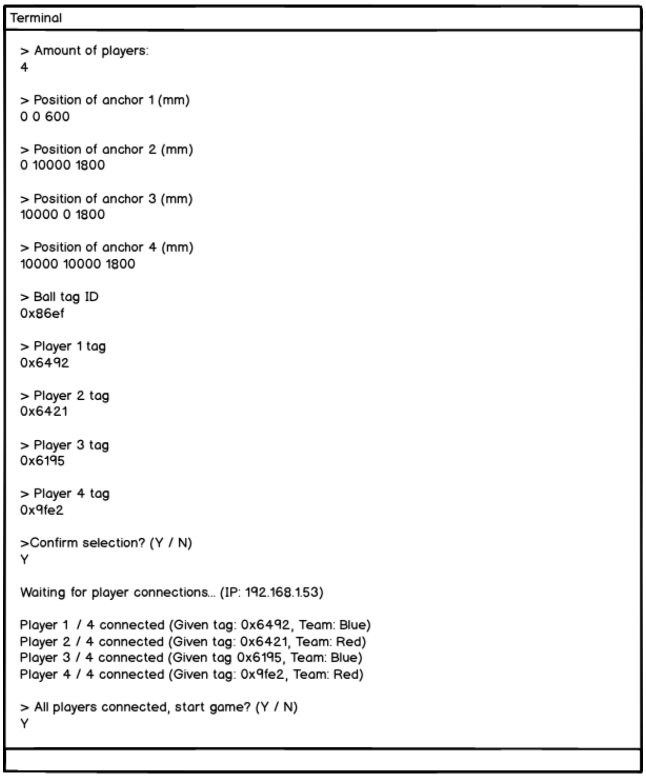

Figure 2.2: Prototype of hosting interface

When a user starts the application, they will be greeted with an input field, where they will specify the IP address of the host machine, as seen on Figure 2.3.

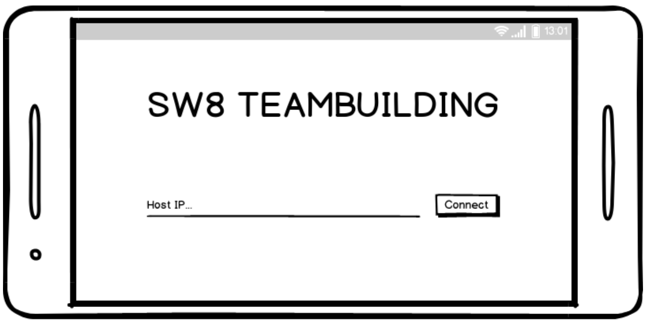

Figure 2.3: Prototype of game menu

After inputting an IP and confirming, they will be redirected to a page where they can see how many users have connected to the host, as seen on Figure 2.4.

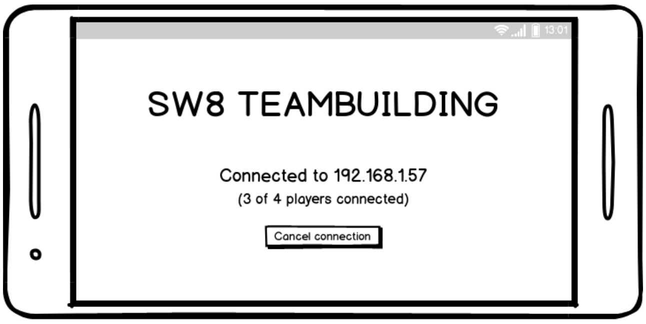

Figure 2.4: Prototype of screen where a user has connected to the host

When all users have connected, the host can start the game, and they will now see the virtual game field, as seen on Figure 2.5. In this prototype, the players' icons are represented by small squares and the goals are represented by larger rectangles and have the same color as the players they belong to.
The current player's icon is highlighted by having a solid color, whereas the other players are just shown as outlines. The players are divided into two teams which are represented by the color of the player icons. In the middle of the screen is the ball in a designated starting area to make the game fair for both teams.

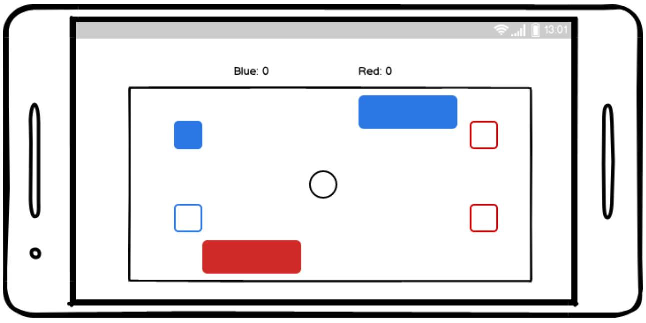

Figure 2.5: Prototype of in-game screen

2.4 Architecture of the game

In addition to the prototypes, which served as a broad overview of how the flow in the application should be, we will look at the overall architecture of the different components involved in the game from a more technical perspective.

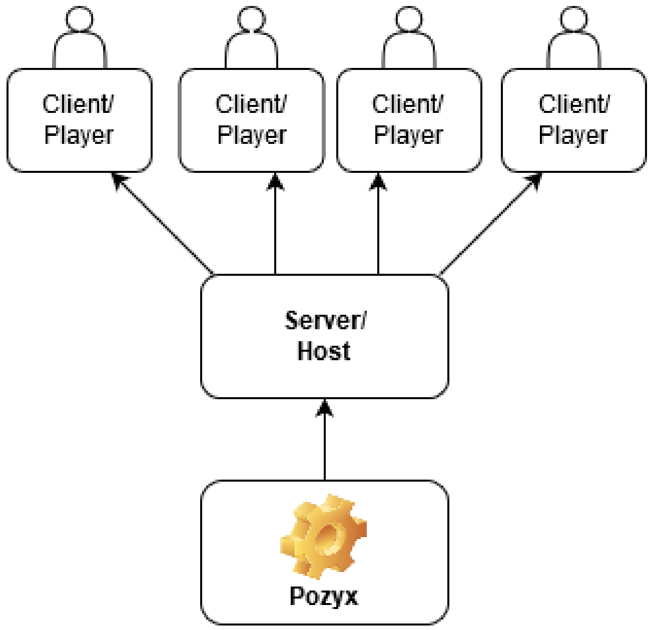

Figure 2.6: The different components of the game

In Figure 2.6 we see an illustration of the initial idea of how the different components of the game are going to work together. The arrows in the diagram show the flow of data in the game. The Pozyx component is responsible for providing the positional data to the host, this will be described in further depth in section 2.5. Each client/-player is equipped with one of the tags that track their positions on the field. The playing field's dimensions will be set to match the distances between the anchors.
The host has the positional data for each client, but it will also need to know which of the tags each client is equipped with, such that it can continuously transfer positional data to the clients. This information will also include which of the tag ids belong to which clients. Each client will be running an instance of the game, and with the use of the positional data, it will render the playing field, the ball and each of the players. The host is responsible for checking if the ball has crossed the goal line and it should then provide this information to the players. In general, the clients' instances of the game should only be responsible for rendering.
Any game logic should happen on the server-side to make sure that the game is synchronized for each player.

2.5 Pozyx

Since positioning is a major part of the project, it is important to have accurate positioning information. Pozyx is a hardware/software solution that is used to provide positioning with an accuracy of down to 10 cm [7]. It makes use of ultra-wide band (UWB) in combination with machine learning for positioning, which according to

their documentation is more precise and efficient than traditional positioning systems such as WiFi, Bluetooth, RFID, and GPS.
Since the two major requirements for positioning in this project are precision and a high update rate to ensure that the players can have reliable data available, the Pozyx system seems like a good place to start.
The Pozyx tags support update rates of up to 125 Hz for a single tag [7]. The Creator system from Pozyx is sold with 4 anchors and 5 locatable tags. An anchor is a stationary sensor used by the moveable tags to get their exact position.

2.5.1 Finding the location of anchors

A trilateration method is used for finding the position of a given tag using the anchors. This method uses basic geometry to estimate the position by measuring the distance to the anchors of which we know the position. With this distance estimate, it is possible to draw a circle with a given radius. If we use two anchors, we will have two intersection points which are the possible positions of the tag. This means that to find a two-dimensional location, we will need at least three anchors, which will lead to only a single point where all three circles intersect, as seen on Figure 2.7.

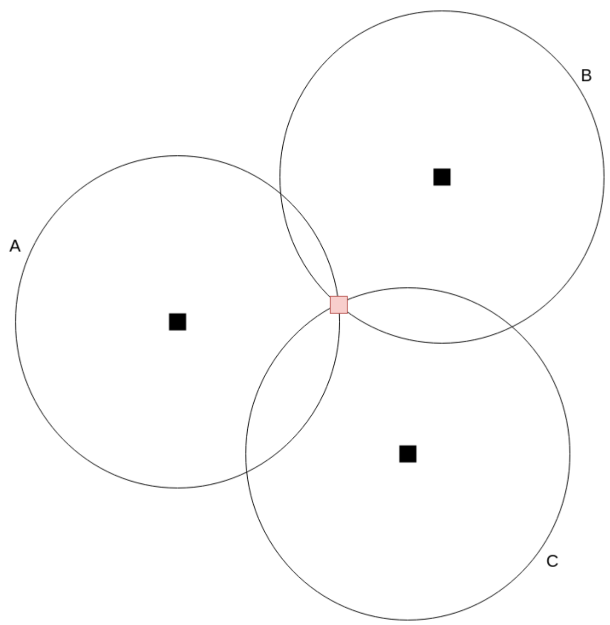

Figure 2.7: An example of trilateration using three anchors

The issue with this approach is that the measurements are not perfect, which might cause the circles to not intersect at exactly one point which will make the positional data seem to jitter.

2.5.2 Using UWB

To find the position of the tags, Pozyx makes use of radio waves. Radio waves travel at the speed of light, so by dividing the time of travel between anchors with the speed of light, the distance between them can be found.
Because the speed of light is so fast the time measure needs to be very accurate to get the correct distance. To achieve this the anchors make use of UWB [9]. UWB is a technology for transmitting data over a wide bandwidth usually wider than 500 MHz. The Pozyx sensors use exactly 500 MHz which means they just qualify as a UWB system. Because the bandwidth is this wide it makes the wavelength very short, and by combining multiple sinusoidal signals with slightly different frequencies the Pozyx tags can create a pulse with a peak which is very narrow. Measuring a narrow peak results in a more accurate timing, allowing the Pozyx to be much more precise than other technologies like Bluetooth and WiFi. High bandwidth means faster data transfer, which most people would prefer, but if everyone were to use the same frequency the signals would interfere with each other, therefore the use of high-frequency signals is tightly regulated[8]. To reduce the amount of interference the regulations say that UWB systems may only transmit at very low power. This means that a UWB transmission can not travel very far and therefore the chance of it interfering with another system is very low. Because of this we are not able to build huge playing fields with the limited amount of anchors we have if we also want it to be precise or even playable. Building a playing field that is too big could cause some tags to not reach all anchors and therefore not calculate a position. This could be combated somewhat by having more anchors, but as will be explained in 2.5.4 the slave tags might not be able to communicate with the master tag because the distance between them is too large.

2.5.3 Alternatives to UWB

Having a high accuracy is essential for the users' experience since a low accuracy could result in the in-game information being incorrect or just too imprecise to make it a joyful experience. While Pozyx uses UWB for positioning, it is interesting to take a look at the primary alternatives to indoor positioning and to see if any of them can compete with Pozyx' high accuracy [7].

GPS

The obvious alternative for positioning data is GPS, which is frequently used for outdoors positioning. In optimal outdoor situations, GPS can provide accuracy within 4.9 meters [15]. However, the GPS positioning accuracy can be degraded due to buildings, bridges, and trees, making it a sub-optimal choice for indoor positioning.

Wi-Fi

Since the project already utilizes networking for data transfer, Wi-Fi may be worth considering to decrease the number of technologies needed. However, Wi-Fi only provides an accuracy of between 5 and 15 meters depending on the hardware chosen for access points and clients.
In addition to this, iOS devices are blocked from using Wi-Fi for indoor navigation purposes, since it usually makes use of a technology called fingerprinting, which only functions with Android devices due to technical restrictions [10].

Bluetooth

Using Bluetooth for indoor location is similar to how Pozyx works with anchors and tags. Instead of anchors, Bluetooth beacons are able to send out signals with a range of up to 30 meters [11]. With Bluetooth, you get a significantly higher accuracy compared to GPS and Wi-Fi. In the optimal settings, this solution can provide an accuracy of up to one meter. Unlike Wi-Fi, this solution would work on both iOS and Android devices but has a lower range than the Pozyx solution.

RFID

Finally, we have RFID which uses radio waves to wirelessly transmit the identity of an object. Unlike the other solutions, RFID offers high accuracy, but a very limited range of less than a meter [12].
For a game like what we are building, it would be possible to use RFID for the location of the ball to check which player is currently holding it, but due to the limited range, it would not make sense to use it for localizing the players.
Generally, using Pozyx with their UWB technology for tracking the position of the players and the ball appears to be the most optimal solution.

2.5.4 Two-way-ranging

We are using the Pozyx Creator Kit Lite which uses the Two-way-ranging (TWR) protocol for positioning [16].
A tag calculates its position by communicating with the anchors one by one, getting the distance from the anchor to itself. Once it has the distance from 3 anchors it can compute its position utilizing trilateration.
If multiple tags are being used at once, one tag is made the master tag and the other tags become the slave tags. The master tag instructs the slave tags to report their position to the master tag one by one. The master tag is then usually connected to a computer that can use the position data. This technique does not scale well as all the slave tags have to be within the radio range of the master tag so spreading them across huge areas is not possible. Instead of a tag being the master it is possible to use an anchor.

This makes it easier to have a computer attached, as the anchors are stationary, unlike the tags.

2.6 Unity introduction

As defined in section 1.1, this project aims to create a location-based augmented reality game. This means the project has to have a game component - an application to display the objectives of the game, the play area, and the players. To create this, a game engine can be used, such as `Unity`.
A game engine is a piece of software that provides creators with the necessary set of features to develop games quickly and efficiently [20]. This means that a game engine is a collection of reusable components, abstracted away from the game developer. This can include tools to help with, for example, graphics, physics, networking or audio. These tools would expose certain functionality to a developer to make use of, and hide the specific implementation details for that functionality, ensuring the developer can focus on more pressing issues. Unity supports the C# language for development [17].

The Unity game engine supports development for different game platforms. Of particular interest to this project is the support for both `Android` and `iOS` devices, as well as `Google Cardboard` [18].
Unity was chosen for the development of the game aspect of this project since this facilitates that a greater amount of time can be spent on the other aspects of the project rather than the low-level details of game development, and it allows for easier inclusion of multiple platforms.

2.7 Networking

The following section will examine the different possibilities regarding transmitting player position data from the Pozyx tags to the Unity applications used to visualize the game.

2.7.1 Possible networking solutions

Unity will be used for the creation of the game aspect of this project as described in section 2.6. Unity includes a proprietary networking solution known as UNet [19]. This solution allows developers to use a high-level API, giving access to commands that cover many common requirements for multiplayer games, without worrying about the low-level details.
Since the solution is developed alongside the actual game engine, it has a higher level of integration with the Unity Editor and Engine, which allows for certain components and visual aids to aid the building of the game. As of the beginning of this project, the UNet solution has been deprecated for a while, and the Unity developers are

actively working to create a new system to replace it.
The current UNet iteration is usable but will be removed in the future. Other third-party solutions for Unity-based games also exist, such as Photon Engine. Photon provides functionality for the developers to make use of to create multiplayer games in the same way as UNet, exposing higher-level functionality. Photon supports multiple platforms outside of just Unity, with both Android and iOS support [5].

The advantage of using a library that is built for Unity such as UNet or Photon is that it is easy to set up with the Unity engine compared to a solution built from scratch. UNet provides a higher level of abstraction with functions to control the networked state of the game, send and receive messages between server and clients and much more.
UNet and Photon also allow for "client-hosted" games that act like lobbies. So any client can host a game that other clients can connect to. This allows the clients to send and receive data between each other [19]. A disadvantage of these libraries is that they are generalized and thus would not be able to achieve the same efficiency as a custom solution tailored to the specific needs of the game.

ZeroMQ is also a possible solution. ZeroMQ is an asynchronous messaging library. It can carry messages across various transport formats and is available in many different programming languages [24]. It aims to be a high-performance library to be used in distributed or concurrent applications that are reliable. According to the getting started guide provided by ZeroMQ, certain issues tend to arise when developers attempt to create a networking solution using sockets [25]. These are:

- How to handle I/O?
- How are dynamic components handled? What happens if a component disappears temporarily?
- How are messages represented? Different sizes and different content can change representations
- How are messages that cannot be delivered immediately handled?
- Where should message queues be stored?
- How are lost messages handled?
- What if the network transport changes, for example, TCP to UDP?
- How do messages get routed? Can the same message be sent to multiple peers?
- How to write an API for another language?
- How to represent data such that it can be read between different architectures? How much of this should be the messaging system's job?

- How do network errors get handled?

These issues are mostly applicable to general solutions that need to accommodate changing requirements or be reusable. However, for this project, not all of these issues are relevant. In terms of problems to overcome, this project should only be concerned with handling dynamic components, handling lost messages, routing messages and handling network errors.
If a player closes the game application it can lead to dynamic component issues. A message can be lost during the playing of the game. Messages should be delivered to all players to ensure that they all have the same information. Finally, a player might suddenly disconnect from the network.

The alternative to making use of a pre-existing solution is creating a custom solution. A custom solution entails a need to establish a familiarity with the required knowledge to construct such a solution. A custom solution would involve sockets, which are a network API that allows programs to communicate with each other [4].

2.7.2 Choosing a solution

There are certain pros and cons associated with both approaches of using either a pre-existing solution or a custom solution. Table 2.1 shows some of the considerations made when deciding an approach for this project. The criteria that were considered when evaluating which solution to use were:

- Customizability
  - How much we can customize the solution to our specific use case. If the customizability is low the project needs to be built around the networking solution, whereas if the customizability is high the networking solution can be customized to our needs.
- Requirements
  - How much knowledge about the subject is needed to use the solution.
- Optimization
  - How much optimization are we able to do ourselves if we use this solution.
- Learning reward
  - How much will we learn if we use this solution.

	Pre-existing Unity-based	Custom	ZeroMQ
Customizability	Consists of a set of pre-defined functionalities	Can have any functionality implemented	Has pre-defined functionalities, but these are lower level than a pre-existing solution
Requirements	Familiarity with the solution	Familiarity with the knowledge required to implement a usable solution	Needs familiarity with a mix of pre-existing and custom solutions
Optimization	Lower-level details are obscured, optimized for general use	Lower-level details are freely available, can be optimized for a specific purpose	Focuses on performance, but the solution is general
Learning reward	Most of it is already implemented, so the learning reward is low	Learning reward is high because we have to implement everything ourselves	Have to customize some of it ourselves, so we will have to get familiar with the subject

Table 2.1: A comparison of the pros and cons of the possible solutions

Based on these considerations, it was decided that a custom solution should be created to handle networking in this project. This choice was based on two major factors: the lack of transparency in a pre-existing solution as well as the need for fast communication and the opportunity to learn more about networking at a low level.
For the game to be playable and enjoyable, the location data collected by the Pozyx system needs to be transmitted to all the clients as quickly as possible such that they always have an up to date view of the positions of the players.
To achieve this, it would be preferable to build a solution capable of performing the minimum amount of work as quickly as possible. Pre-existing solutions cannot be guaranteed to do the minimum amount of work as lower-level details are obscured from the developers. With a custom solution, the data sent across the network can be guaranteed to be exactly what is needed.
ZeroMQ was also a possible choice based on the performance needs, but its generalized approach concerning itself with reusability and issues unlikely to be a big factor in this project meant it was dismissed, in favor of a custom solution in which the problems defined in the previous section are handled. Additionally, we have not previously worked with networking at a low level, but wanted to learn more about

it, and decided that working without a framework would provide the best learning experience.

2.7.3 Introduction to sockets

In order to construct a network solution, a familiarity with the layers of a network is needed to gain an intuition of what sockets are. A common way to describe these layers is through the *open systems interconnection* (OSI) model for communication. This model is illustrated in Figure 2.8, along with approximate mappings of the technologies used for each layer.

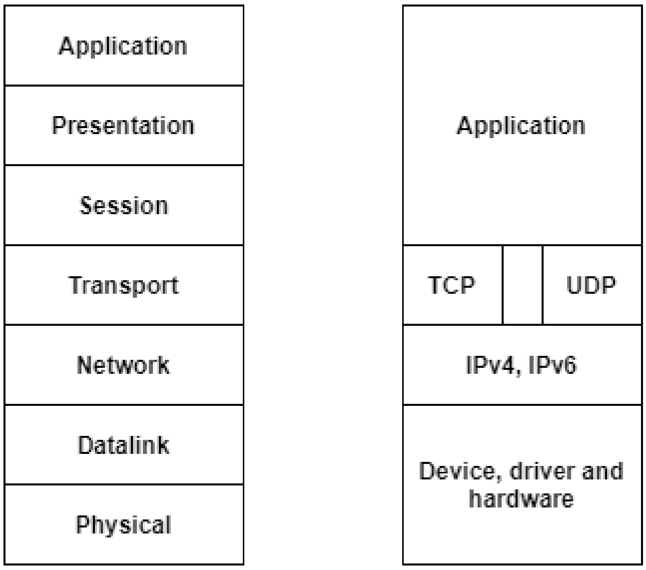

Figure 2.8: An illustration of the OSI-layer, and the corresponding technology for each layer.

As shown, the relevant layers for the purpose of creating a networking solution through sockets is the third and fourth layers, transport and network. The network layer is handled by the IPv4 and IPv6 protocols, which will be discussed in subsection 2.7.6, and the transport layer is handled by either TCP or UDP, which is described in subsection 2.7.4.
The reason for the gap between TCP and UDP is to illustrate that it is possible to bypass this layer and use IPv4 or IPv6 directly [4]. Sockets provide the interface from the upper application layers to the transport layer. The upper layer handles details about the application, and the lower layers handle details relating to communication.

Programs that communicate across a computer network need an agreement on how those programs will communicate. This is known as a protocol. Generally, before defining the design details of the protocol, a decision should be made as to which program is expected to initiate communication. One way of defining this is through the client-server architecture illustrated in Figure 2.9. This split is used by most network-aware applications [4]. The most common method of initiating communication when using the client-server architecture is to have the client initiate requests. This tends to simplify the protocol and the programs themselves [4].

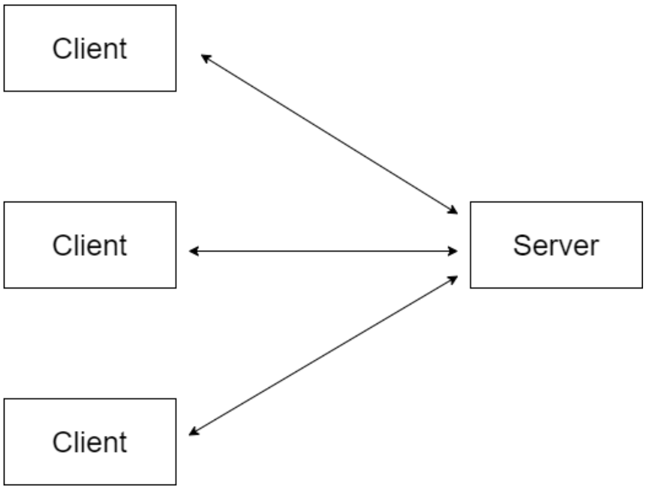

Figure 2.9: An illustration of the client-server architecture with multiple clients

2.7.4 TCP and UDP

The following section introduces two different protocols for the transport layer - Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Both of these protocols use a network-layer protocol known as IP, which can be either the protocol IPv4 or IPv6.

TCP

TCP provides connections between clients and servers. A TCP client establishes a connection with a given server, then it receives or sends data to that server across the network and closes the connection. TCP provides reliability. When TCP sends data, it requires an acknowledgment from the receiver that the data has been received. If it does not receive such an acknowledgment, TCP automatically retransmits the data and then waits for an acknowledgment for the retransmission. After a certain number of retransmissions, TCP gives up.
Based on implementation TCP will typically attempt to send data for 4-10 minutes. This does not guarantee that the receiver will receive the data, but the guarantee is that it will deliver the data if possible, or notify the user that the connection has been broken without an acknowledgment from the receiver. To know how long to wait for acknowledgments, TCP contains algorithms to estimate the *round-trip time* between the client and server dynamically. It also performs these estimations continuously, as the result can be affected by variations in the network traffic. TCP sequences data by associating bytes and sequence numbers.

For example, if an application writes 2048 bytes to a TCP socket, it would be sent in two segments, with the first containing data with the sequence number 1-1024, and the second containing data with the sequence number 1025-2048. If they arrive in the wrong order, the receiving TCP reorders the segments based on the sequence numbers before passing the data to the receiving application. If the receiver receives duplicate data this can also be detected through the sequence numbers, and the duplicate data can be deleted.
TCP provides flow control by telling clients how many bytes of data can be accepted at any time, known as the window. The size of the window decreases as data is received, and decreases as the receiver reads data from its buffer.

UDP

UDP is a simple transport-layer protocol [4]. An application writes a message to a UDP socket, which gets encapsulated in a UDP datagram, which further gets encapsulated in an IP datagram and then sent to the destination. A datagram is a self-contained entity of data carrying information to be routed from the source to the destination nodes without reliance on earlier exchanges between the nodes and the transporting network [6].
A UDP datagram is not guaranteed to reach its final destination, nor is it guaranteed that order will be preserved across the network, or that datagrams arrive only once. This means that the UDP protocol is unreliable. If a datagram is lost on the network and not delivered to the UDP socket, it will not be automatically retransmitted. UDP also does not provide an acknowledgment that datagrams were received, sequence numbers to ensure data can be ordered, *round-trip time* estimation or timeouts.

UDP has no notion of flow control, meaning a fast UDP sender can transmit data at a rate the receiver is unable to keep up with. As such, it does not provide the same reliability as TCP. If reliability is a requirement, it has to be built through features such as timeouts, retransmissions and adding acknowledgments from the receiving end. A UDP datagram has a length, which is passed to the receiving application along with the data. UDP is considered connectionless, as there does not need to be a long-term relationship between a UDP client and the server.
A UDP client can create a socket and send a datagram to a server, and then immediately send another datagram on the same socket to a different server. A UDP server can receive several datagrams on a single UDP socket, each from different clients.

Choosing between UDP or TCP

Windowed flow control might not be necessary for transactions where both ends agree on the maximum size of a request or a reply [4]. For this project, the most important type of message is the player location data. This message will always be formatted in the same way, and as such, a maximum size can be agreed to mean the

flow control aspect of TCP is not needed. Another aspect of TDP that is not needed for this project is the automatic retransmission of messages. While this can provide reliability, it is of no importance for the game.
The players of the game are only concerned with the most recent updates of their position. As such, if a message were to not be received, it would not make sense to continually delay subsequent messages to attempt to retransmit a message containing position data that is more and more likely to be outdated.
For the position data to be as recent as possible, the messages should be sent as frequently as possible. It is also not necessary to provide an acknowledgment that the message has been received. The receiving applications should simply update their locations to comply with the most recently received data. The sender should not be concerned that a message was received, it should just continue to send the next message, which is likely to be more recent. Duplicate messages also do not pose much of an issue. If the applications were to receive the same location data multiple times, it would not impact the overall functionality of the program, rather just the speed at which the next updates would be received.
UDP has no connection setup or teardown costs. UDP only requires two packets to exchange a request and a reply, whereas TDP requires about 10 packets [4], if a new TCP connection is established for each exchange. In terms of transaction time, the minimum time for a UDP request-reply is the round-trip time + server processing time, and the minimum time for TDP is 2× round-trip time + server processing time [4].

Because of the limited scope and uniformity of what is going to be transmitted via the networking solution, a lot of the features included in TCP are unnecessary. UDP is slightly faster because of its lack of reliability and other benefits but might require some extra work to implement some of the functionality that is missing when compared to TCP if this were to become necessary. Because of the reasons discussed, UDP seems to fit the needs of this project more than TCP and is the protocol chosen for the networking solution featured in this project.
An issue with the choice of UDP could present itself in that messages are not guaranteed to arrive in order. It could pose a problem if a player in the game received a message with recent location data, and then another message afterward with outdated data. This could cause the player objects in the game to be at positions in which they were in the past, but not currently in the present.

2.7.5 Introduction to UDP sockets

UDP is a connectionless, unreliable datagram protocol. Figure 2.10 shows an illustration of the client-server architecture using UDP. The client-side creates a socket and sends a request to the server as illustrated.
Once the request has been sent the client transitions to a state of awaiting a reply. Once the reply is received the client can send another request, or the socket can be closed. The server side also creates a socket, and then binds the socket to a

port. Once bound, the server can await a request from the client. When a request is received, the server processes it, and then sends it to the client after which it can return to awaiting requests.

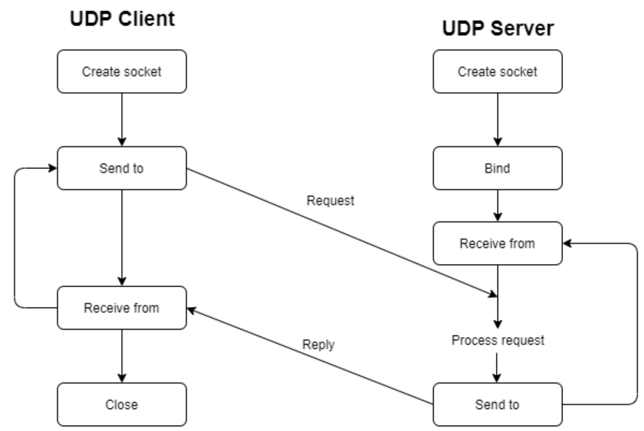

Figure 2.10: An illustration of the client-server architecture with UDP

For the purpose of this project, it might not be optimal to have client request the server. The server does not need information from the client or acknowledgment, meaning the clients do not have to send messages. As such, it might be better to use a publisher-subscriber approach.
The server would then act as a publisher, constantly sending messages to the clients that would be subscribed to the publisher. An illustration of this concept using UDP can be seen in Figure 2.11.

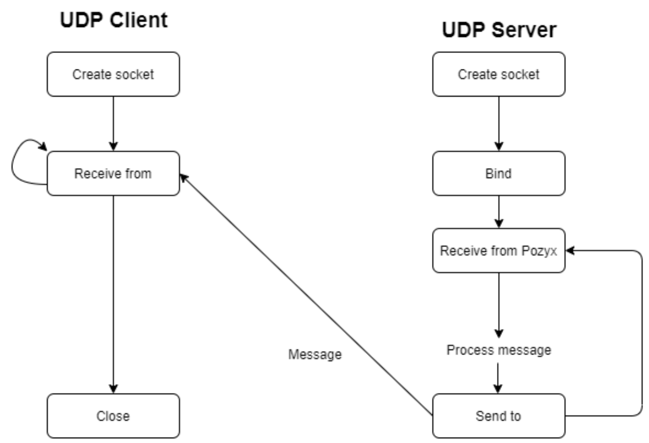

Figure 2.11: An illustration of a publisher-subscriber architecture with UDP

Possible issues with using this architecture is that there would need to be a way to ensure all clients receive a sufficient amount of messages, and that the receivers are not overloaded with messages.

Transmission in UDP

When working with UDP, there are three types of possible transmissions: Unicast, multicast or broadcast. Unicast is a one-to-one communication with a single source sending information to a single receiver, while a broadcast transmits the information to all nodes on a network.
The benefit of unicast is that it has been in use for a long time and utilizes well-established protocols, it is known from applications like HTTP, FTP, and Telnet [3]. For this project, however, the major drawback of unicast is that to transmit the message to multiple nodes, it has to send multiple unicasts messages addressed to each receiver, which also requires us to know the exact IP address of each destination device.
Looking at broadcast, we can instead transmit the information to all nodes on a network, which ensures that all nodes on the network receive the message. This could be useful if we only had the players of the game on our network, but since they might be connected to a larger network with a large number of clients, this could lead to the data being sent to clients that are not a part of the ongoing game session. Luckily, we have multicast in the middle of the two extremes, where you do not send from one client to another or one client to all others.
Instead, the data is sent to as many destinations as express an interest in receiving it [3]. This one-to-many approach seems suitable for this project since it would intuitively lead to better bandwidth utilization and does not require the receivers' addresses to be known.

2.7.6 IPv4 and IPv6

Both IPv4 and IPv6 provide packet delivery service for TCP and UDP. The major difference in IPv4 and IPv6 is the addresses they use. IPv4 uses 32-bit addresses, whereas IPv6, being a newer version, uses larger addresses of 128 bits [4]. IPv4 addresses are usually written as four decimal numbers separated by ".". This is known as *dotted-decimal notation*.
Each decimal number represents one of the four bytes of the 32-bit address. The first of the four numbers represent the address type. IPv6 addresses are usually written as eight 16-bit hexadecimal numbers [4]. The higher-order bits of the 128-address imply the type. For this project, this should not have a significant impact, and either can be used.

2.8 Experiment with Pozyx

To determine the accuracy of the Pozyx tags an experiment was conducted. The primary goal of the experiment was to test the accuracy, but a secondary goal of the experiment was to determine the frequencies of updates for each tag.

2.8.1 Purpose

According to research regarding latency in VR [22] and interactive systems [14], the number of movement errors increase as the latency increases. An example of this is when a tag is moved, and the tag first sends a new position after 3 seconds, then the movement error could be high as the user does not know their exact position for these 3 seconds. According to experiments with VR, the motor performance and sense of body ownership start to decrease at latencies above 75 ms.
This means that the optimal results from these experiments would indicate that the Pozyx system can send position updates fast enough which allows the system to operate with a latency of less than 75 ms between their physical movement and the in-game reflection of this movement.

2.8.2 Setup

The tags were set to transmit with the highest bitrate with the longest preamble length and with the ranging mode set to precise [2].
According to the documentation, these values lower the tags update rate to about 9hz which is then divided by the number of tags used in the system as described in subsection 2.5.4. As the main goal of the experiment was to test the accuracy these settings were chosen to gives the most accurate positioning. The settings can be changed at a later point to try to increase the update rate to a level where the latency for the users is acceptable.
The experiment was set up as shown on Figure 2.12. The experiment was conducted indoors on our campus in the building Novi 9. The anchors `0x632b` and `0x676e` were mounted on a wall 240 centimeters apart, and the remaining anchors `0x6738` and `0x676c` were mounted on a bulletin board.
The number of centimeters accompanying the hexadecimal number of each anchor is the height at which the anchor was mounted during the experiment. Different heights were chosen as Pozyx documentation suggests that not all anchors should have the same height [23]. The reason for this is the principle of geometric dilution of precision (GDOP), which can cause the error on range measurements to be amplified.

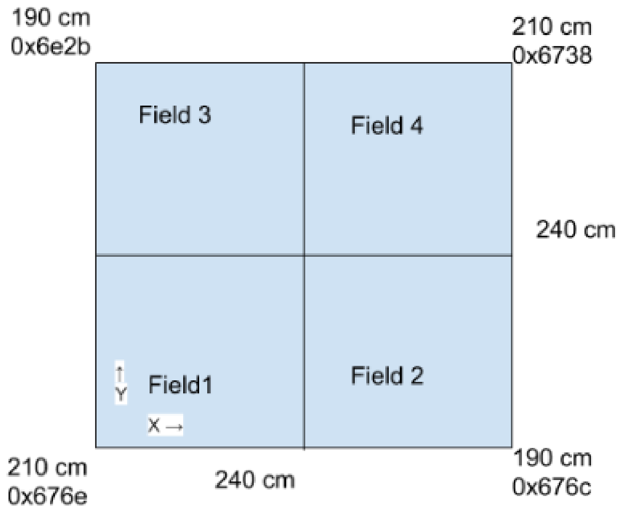

Figure 2.12: The setup of the experiment with the anchors and the height at which they were placed in the corners. The hexadecimal number is the anchor, and above that is the height at which the anchors were placed.

The fields were created by a whiteboard on which lines were drawn every 10 centimeters to know the actual position as seen on Figure 2.13. This whiteboard was moved intermittently to act as respectively fields 1, 2, 3, and 4.

Figure 2.13: The whiteboard with the drawn positions.

The procedure for the experiment was based on this whiteboard. The whiteboard would be placed in one of the fields defined in Figure 2.12, and the tags would be placed in certain positions on the board and record the accuracy with which the position was reported. The tags would be placed in a position, remain there for five seconds, then be moved to the next position over the next five seconds and remain in this position for five seconds before being moved again. This procedure was repeated until a satisfactory number of measurements had been made. This amount was usually 5 measurements. Once one field had been tested, the whiteboard was moved to the next field and the process repeated.

2.8.3 Analyzing the data

In section 6.1 the data for the experiment can be seen. For each test the average grid was calculated, and the minimum and maximum values for x, y and z were found. Likewise, the average deviation was calculated from the actual grid to the average grid to get a better view of the deviation. The min and max values were noted, as they may be useful for future analysis. The z values were very inconsistent throughout the entire test, and due to this and the z-axis being largely irrelevant to the game, we will not focus on analyzing them.

Precision with 1 tag

The average of all deviations is 10.54 cm. There were spikes where the max and min values spiked a lot. For `(30, 60)` the minimum x value was -33.3, which is a deviation of around 60 cm and for `(120, 0)` the minimum y value was -54.5 cm, which is also 54.5 cm away from the actual data point. There are more cases where the deviation of a single data point is far away.

Precision with 3 tag

For this part of the experiment, the accuracy was tested with three tags. The tags were positioned at three different x-coordinates. 140, 180 and 220 were chosen as the x coordinates and the y coordinate was moved up by 30 cm each time starting from 0 to 120.

Tag 26467 The overall average deviation for this tag was 11.2 cm. There was, however, a large deviation at `140, 90` and `(140, 120)` with 20.63 cm and 20.46 cm, respectively. The min and max values for x and y were good, and there were never any larger spikes from the actual data point than around 25 cm.

Tag 26895 The overall average deviation for this tag was 6.31 cm. The average deviation was more accurate than expected for this tag. For this tag, there were also no spikes larger than around 20 cm for the x and y values.

Tag 24622 The overall average deviation for this tag was 17.41 cm. The first three points the results seemed fairly accurate, however, the grid `(220, 90)` and `(220, 120)` had a large deviation of 37.53 cm and 24.4 cm respectively. The min and max values for x and y were greater in this than the previous, with around 40 cm deviations.

Precision with 5 tag

For this experiment, it was noted how many data points were sent in the 5-second intervals, as well as the average deviation. The tags were positioned at five different x-coordinates. 140, 160, 180, 200 and 220 were chosen as the x coordinates and the y coordinate was moved up by 20 cm each time starting from 0 to 120.

Tag 24622 The average deviation for this tag was 8.42 cm and the average amount of data points every 5 seconds was around 4.28 data points. However, at `(180, 120)` we only received 1 data point. The min and max values of x and y were lower than 25 cm.

Tag 26467 For this tag, the average deviation was 11.20 cm and the average amount of data points was 3.28. At `140, 40` there was only one update, and at `(140, 80)` there were only 2 updates every 5 seconds. The max `x` value was around 30 cm away from the actual data point, but otherwise, they did not spike larger than that.

Tag 26895 Tag 26895 had an average deviation of 14.78 cm. The deviation was higher than the two previous, however, there was an average update rate of 9.57, which is more than twice as high as the two previous tags. The min and max values for min `y` and max `y` deviated at `(160, 20)`, where the min value for x was -28 cm, and max value was 49.4 cm.

Tag 26901 The average deviation of tag 26901 was 8.56 cm, and it sent an average of 10 data points every 5 seconds. This tag was quite stable, as there were 10 data points every 5 seconds almost every time, and the average deviation never spiked higher than 12 cm. Around `(220, 100)` the min y value was 51.3 cm, which is quite far away from the actual position. There were a few instances of large gaps between min and max values and the actual grid.

Tag 27001 The average deviation of this tag is 9.61cm and the average amount of data points every 5 seconds was 5.57. This was also quite consistent with the average deviation. The max values for x and y were also quite consistent and there were no large deviations. The min and max deviation of the x coordinate were quite accurate with the largest distance of around 20 cm, but mostly less than 10 cm. The y coordinate fluctuated more but was still less than 25 cm.

2.8.4 Possible influences on the test

One thing that could have affected the tags is that a whiteboard was used as a measure for positions. As there is metal in the whiteboard this could have affected the precision on the tags. Metals are conductors, which can lead to the signal having less power and reduced range, and the signal might spend extra time trying to get through the metal. Since Pozyx positioning relies on calculating the time of flight, having the signal spend extra time traveling reduces accuracy [21].

If we look at the z coordinate in the test experiment data, it can be seen that z fluctuates a lot more than the x and y coordinates. The max z coordinate for multiple experiments is often more than 3 meters and the average z coordinate is also often 1 meter higher than the actual coordinate. This could be due to the height difference between the anchors not being sufficiently large.

During the test with 1 tag, the 0 value of the x coordinate coincided with the wall. This could have been an influence on the positioning result in that it could increase uncertainty, as it was difficult to center the tag over the x coordinate since the coordinate collided with the wall.

While the experiment was running the tags could throw errors instead of giving a position if something went wrong. Mainly during the multi-tag tests, some tags would report back with errors such as unable to get firmware version, flash memory corrupted or an error message saying there was no error. At that point in the experiment, it was not clear whether it was a hardware error with some of the tags or if it was code related. The tags would throw these errors randomly and then report back their position as normal at the next update. This resulted in some of the tags having periods of update rates of less than 0.2 Hz. It was decided that further investigation into this issue should be conducted at a later point.

2.8.5 Conclusion on the experiment

Our conclusion of this experiment is that the precision is satisfactory, but the update rate is not optimal as that does not meet the standards of the refresh rate needed.

We also need to consider the spikes in the coordinates, such that the user does not jump around on the screen, even though the user might not be moving.

The results of this experiment shows that a three dimensional game is not viable with the errors that we had on the z coordinate. This is due to the height difference between the anchors only being 20 cm. If we, later on, choose to make use of the z coordinate, then it is necessary to conduct another experiment with the Pozyx with different, larger height differences, but as it currently is only a two-dimensional game it is not necessary.

It also seemed like some tags were not able to send as many data points as others. This was also due to getting errors from some of the tags. This is something that should be investigated.

Chapter 3

Sprint 2

3.1 Sprint goal and introduction

The goal for this sprint is to explore the networking side of the project and implement initial versions of the UDP client and server based on what was learned in section 2.7. Additionally, the first version of the game should be created, which allows seeing the game through a VR headset and can show the players moving based on the position of the tags.

3.2 Deployment Diagram

section 2.4 introduced the different components of the system in Figure 2.6. In order to further elaborate on the different components, a deployment diagram is constructed.

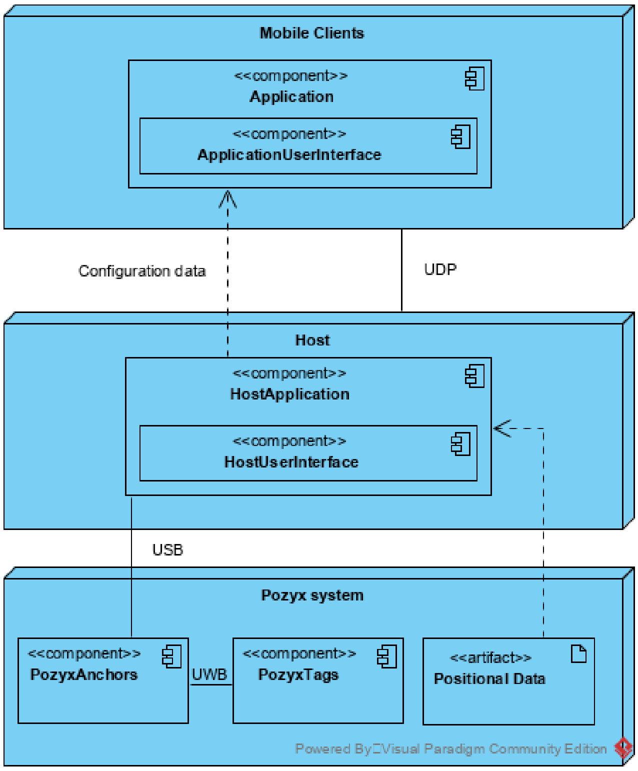

Figure 3.1: A deployment diagram for the system.

A deployed system will contain three nodes:

- Pozyx system
- Host
- Mobile Clients

Nodes are represented by cubes, and are entities executing components. The Pozyx system node contains two components. These are the anchors and the tags needed to generate positioning data with the Pozyx hardware. The system contains multiples of each. These Pozyx components generate artifacts in terms of positional data for the location of the players and the ball. Anchors and tags are associated through a UWB connection in order to generate the artifacts. The host node is dependent on the positional data artifact, the component receives the positional data and transforms it for communication. Positional data is transferred from the Pozyx system to the host through a USB association. One anchor is associated with the host application, and this anchor is responsible for collecting data from all tags. The host application component contains a user interface component, which is the interface that the person using the host component will interact with, in the form of a terminal application for the early version of the system. The host node is associated with the clients through a

UDP connection in order to communicate both the positional data and configuration data to perform game setup. As such, the mobile client application component is dependent on the host application component, since it needs the configuration data. The mobile clients also contain user interface components, which is how the users interact with that part of the system. This user interface is the virtual playing field generated in Unity that the users view in the headset.

3.3 Accessing games on the network

Different methods of transmission were discussed in subsubsection 2.7.5, and multicasting was selected as the optimal solution for this project. In order to make proper use of this in an eventual deployment of the game, it would not make sense for the users to input the IP address to which they want to connect. As such, a way to access hosted games without this needs to be implemented. In order to support multiple games being played at the same time, the system also needs to properly make use of multicast groups based on the game being joined. Clients should only receive data from one specific host in their specific multicast group.

To do this, the game must include some form of LAN discoverability. One way of achieving this is to have clients searching for a game broadcast a message to available hosts via the LAN. The hosts then reply if they are available along with data for a multicast group, and the client can then join that group. Another method could be to make use of Internet Group Management Protocol (IGMP), which is a communications protocol used on IPv4 networks to establish multicast groups. The IP address 224.0.0.1 is a notable IPv4 address reserved for IP multicasting, which is the multicast group address for all hosts on the same network [13]. All hosts should join that group on start-up, and clients could then message the group that they are looking for an available game.

Another way would be to have the host continually broadcast messages that it is available for players. This would, however, lead to the host needing to repeatedly send messages for as long as it is not filled with players or has not started. This would likely be a less elegant solution than having players searching for hosts message a couple of times.

Having the players send a message to the multicast group address for all hosts is the preferable solution, as it will avoid unnecessary overhead caused by broadcast messages being sent to unrelated machines on the network. In order to select a game to join as a client, the game would need a game selection screen. Figure 2.3 and Figure 2.4 show the initial prototypes for joining a host-based on their IP and waiting for the game to begin. If a lobby were to be implemented, Figure 2.3 would need to be updated, and there would need to be an extra step before Figure 2.4 could be shown. Figure 3.2 and Figure 3.3 show new prototypes for this purpose. Choosing a lobby on Figure 3.3 would lead to Figure 2.4.

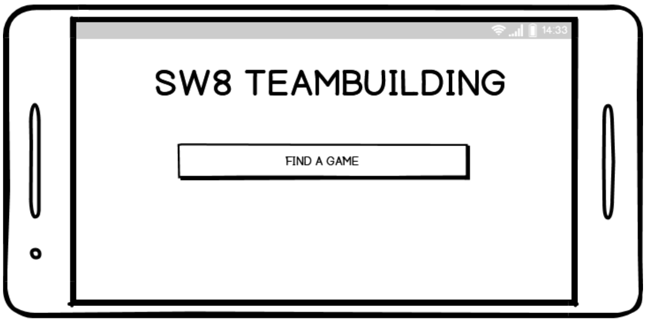

Figure 3.2: Prototype of the game menu when players can choose a game to join. They need to either search for games or host one.

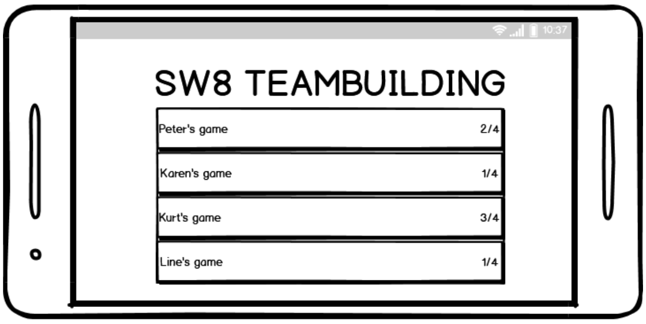

Figure 3.3: Prototype of a lobby menu. Available games are shown as well as the number of players. Players can choose one to join.

3.4 Process changes

The process required some unexpected pivots in the second sprint due to the coronavirus, which lead to the university being locked down for a period.
The daily group work was moved from the physical group room to a virtual setting facilitated by Discord. One of the major differences was that it was previously possible to sit down and do the pair reviews in the morning, which meant that they would go through review quickly. However, while we tried to still get the pair reviews done first thing in the morning, there was a noticeable delay before the new features were reviewed and merged.
Another minor change was how we assigned points (cost, reward, priority) to tasks. This worked out well with online, as people could simultaneously post their point number in a chat, as opposed to showing a number of fingers while sitting around a table.

Since most of the project work is managed on Jira and completed by individual group members, there was not a noticeable change in the amount of work getting done. A slight side effect of working with only voice chat has been that it was occasionally difficult to reach the other members when their help was needed since they might be temporarily away from the computer, or too distracted or focused to be paying attention to the voice chat. This also resulted in some information not getting spread to the entire group, as not everyone may have been listening in while some discussion was happening, which lead to minor confusion in further discussions on the subject.

3.5 Current product

To give an overview of the progression of the project this section will provide an overview of what has been created during the sprint.

This overview will be split into two categories, to reflect the structure of the project: Networking and game.

3.5.1 Networking

The network aspect has been a big focus this sprint and has led to a better understanding of how the data should flow through the system (section 3.2, and how we can use the knowledge from sprint 1 about networking to automatically find on-going games on the local network (section 3.3).

From an implementation perspective, this sprint has introduced an initial version of the UDP host which is capable of transmitting Pozyx location data from the host computer to the game clients. For the host, an initial terminal-based setup has been created, where the user can input the number of players, the location of the anchors and specify which tags are used in the game. Additionally, the host can automatically re-order the list of entered anchors to ensure that they are sent to the clients in a clockwise manner, such that Unity can create a playing field mesh based on the coordinates.

3.5.2 Game

For the game aspect of the project, the most crucial parts have been implemented this sprint:
First of all, the game has been set up to support VR glasses by splitting the game view into two halves, one for each eye, as seen on Figure 3.4.

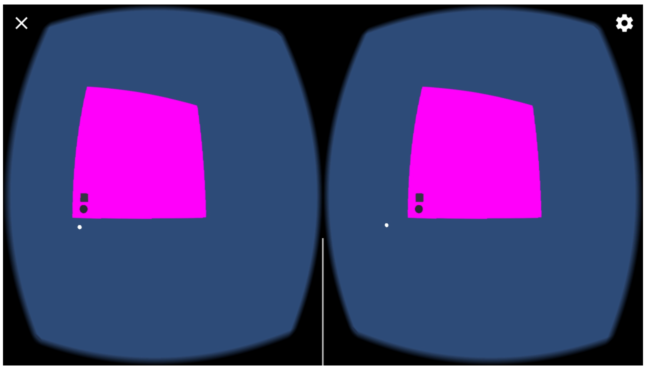

Figure 3.4: The first implementation of VR

The game supports moving the players upon receiving new information from the host, but the connection between the two was not coupled together in this sprint.

Finally, an algorithm was implemented to generate the goal zones and ensure that they are placed fairly on the playing field.

Chapter 4

Sprint 3

Chapter 5

Sprint 4

Chapter 6

Appendix

6.1 Results for experiment

6.1.1 Experiment with one tag

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(0, 0)	(0.9, 1.1, 117.9)	(0.0)	(34.0)	(-23.10)	(18.8)	(0.0)	(346.50)
(30, 0)	(36.4, 9.0, 156.6)	(26.6)	(51.2)	(1.3)	(21.30)	(55.10)	(336.7)
(60, 0)	(67.0, 11.3, 203.3)	(55.7)	(79.8)	(0.7)	(28.1)	(63.2)	(203.35)
(90, 0)	(99.2, 4.5, 248.6)	(83.4)	(108.2)	(-37.3)	(16.3)	(79.3)	(322.6)
(120, 0)	(133.0, 4.5, 249.0)	(109.4)	(172.1)	(-54.5)	(14.0)	(48.2)	(305)
(0, 30)	(22.8, 41.3, 220.7)	(4.6)	(35.2)	(10.0)	(62.7)	(24.5)	(36.2)
(30, 30)	(24.4, 38.5, 225.2)	(8.3)	(57.0)	(26.0)	(67.7)	(28.4)	(33.6)
(60, 30)	(66.8, 33.0, 149.2)	(55.4)	(85.4)	(19.3)	(50.1)	(52.5)	(33.1)
(90, 30)	(99.9, 26.5, 292.9)	(89.2)	(105.8)	(18.5)	(33.0)	(84.9)	(321.2)
(120, 30)	(122.1, 33.9, 283.4)	(104.8)	(136.8)	(14.1)	(53.3)	(64.3)	(31.3)
(0, 60)	(22.9, 41.3, 220.8)	(4.6)	(35.2)	(10.0)	(62.7)	(24.5)	(362.3)
(30, 60)	(34.6, 64.0, 64.0)	(-33.3)	(84.6)	(46.9)	(110.1)	(7.8)	(334.9)
(60, 60)	(62.1, 78.0, 282.4)	(47.0)	(81.5)	(59.0)	(97.5)	(70.5)	(338.1)
(90, 60)	(90.2, 63.9, 106.6)	(80.7)	(97.1)	(56.8)	(70.1)	(81.8)	(317.1)
(120, 60)	(105.6, 50.3, 150.0)	(62.2)	(135.5)	(13.8)	(74.3)	(76.2)	(336.5)
(0, 90)	(18.4, 98.1, 193.8)	(4.9)	(42.1)	(73.8)	(122.9)	(47.9)	(347.2)
(30, 90)	(33.0, 91.8, 261.7)	(-5.2)	(54.4)	(69.7)	(106.5)	(61.4)	(330)
(60, 90)	(64.0, 88.5, 120.6)	(54.1)	(76.7)	(76.2)	(98.8)	(68.9)	(317.6)
(90, 90)	(88.3, 92.5, 99.2)	(80.0)	(103.3)	(75.2)	(107.7)	(82.1)	(313.5)
(120, 90)	(114.1, 86.3, 124.0)	(88.1)	(136.4)	(69.7)	(112.7)	(65.0)	(314.8)
(0, 120)	(22.7, 118.1, 77.4)	(11.6)	(58.9)	(62.2)	(133.8)	(61.8)	(365.4)
(30, 120)	(34.9, 119.5, 197.5)	(15.9)	(56.4)	(96.8)	(138.6)	(82.1)	(333.1)
(60, 120)	(67.0, 124.2, 242.3)	(45.1)	(83.2)	(108.3)	(143.1)	(82.2)	(313.6)
(90, 120)	(91.3, 121.5, 104.5)	(79.4)	(105)	(114.3)	(130.0)	(76.5)	(315.2)
(120, 120)	(130.7, 117.9, 70.0)	(119.9)	(142.4)	(105.6)	(126.7)	(62.90)	(81.9)

Table 6.1: Table with grids for experiment with 1 tag. Tag: 26895

6.1.2 Experiment with three tags

Position	Average deviation
(0, 0)	1.42
(30, 0)	11.04
(60, 0)	13.29
(90, 0)	10.24
(120, 0)	13.75
(0, 30)	25.44
(30, 30)	10.17
(60, 30)	7.43
(90, 30)	10.34
(120, 30)	4.42
(0, 60)	25.53
(30, 60)	6.09
(60, 60)	18.12
(90, 60)	3.90
(120, 60)	17.36
(0, 90)	20.10
(30, 90)	3.49
(60, 90)	4.27
(90, 90)	3.02
(120, 90)	6.96
(0, 120)	22.77
(30, 120)	3.49
(60, 120)	8.16
(90, 120)	1.98
(120, 120)	10.90

Table 6.2: Average deviation for each position with tag 26895

Position	Average deviation
(140, 0)	9.06
(140, 30)	2.02
(140, 60)	3.84
(140, 90)	20.63
(140, 120)	20.46

Table 6.4: Average deviation for tag 26467

Position	Average deviation
(180, 0)	4.56
(180, 30)	12.57
(180, 60)	3.67
(180, 90)	5.51
(180, 120)	5.24

Table 6.6: Average deviation for tag 26895

Tag 26467

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(140, 0)	(141.1, -9.0, 272.2)	(134.5)	(147.7)	(-15.1)	(1.6)	(79.1)	(325.5)
(140, 30)	(141.9, 29.3, 161.3)	(134.1)	(150.4)	(23.1)	(37.5)	(77.5)	(306.8)
(140, 60)	(133.8, 60.6, 89.0)	(132.3)	(135.3)	(59.9)	(61.2)	(87.5)	(90.5)
(140, 90)	(129.6, 72.8, 161.5)	(123.8)	(137.5)	(55.0)	(89.2)	(85.7)	(308.4)
(140, 120)	(123.8, 107.5, 91.7)	(114.7)	(128.9)	(90.9)	(120.3)	(80.2)	(105.0)

Table 6.3: Table with grids for experiment with three tags. Tag: 26467

Tag 26895

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(180, 0)	(184.1, -2.0, 242.2)	(172.6)	(192.7)	(-10.7)	(18.2)	(86.6)	(304.9)
(180, 30)	(185.3, 18.6, 165.4)	(174.5)	(202.1)	(9.3)	(29.1)	(79.7)	(295.4)
(180, 60)	(181.4, 56.6, 173.2)	(173.7)	(192.9)	(48.7)	(62.8)	(84.9)	(307.2)
(180, 90)	(176.2, 94.0, 175.1)	(167.0)	(190.3)	(85.8)	(104.6)	(85.7)	(320.5)
(180, 120)	(175.5, 118.3, 82.4)	(168.6)	(180.5)	(112.8)	(122.7)	(74.0)	(94.5)

Table 6.5: Table with grids for experiment with thee tags. Tag: 26895

Position	Average deviation
(220, 0)	8.82
(220, 30)	9.00
(220, 60)	7.32
(220, 90)	37.53
(220, 120)	24.40

Table 6.8: Average deviation for tag 24622

Tag 24622

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(220, 0)	(228.4, 2.7, 270.8)	(210.8)	(236.1)	(-7.9)	(10.2)	(58.2)	(325.9)
(220, 30)	(228.8, 31.9, 261.7)	(219.0)	(232.0)	(25.3)	(51.3)	(96.7)	(298.2)
(220, 60)	(224.2, 54.0, 193.1)	(222.5)	(226.0)	(51.3)	(56.7)	(97.1)	(289.1)
(220, 90)	(226.9, 53.1, 139.5)	(205.8)	(255.1)	(49.6)	(61.0)	(76.5)	(276.5)
(220, 120)	(206.2, 101.4, 230.0)	(192.8)	(213.4)	(83.8)	(125.6)	(89.0)	(326.8)

Table 6.7: Table with grids for experiment with three tags. Tag: 24622

6.2 Experiment with five tags

Tag 24622

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(180, 0)	(179.8, 14.8, 222.8)	(170.7)	(192.4)	(-4.7)	(35.0)	(92.3)	(298.9)
(180, 20)	(180.3, 37.5, 94.3)	(176.3)	(185.8)	(27.7)	(44.9)	(84.6)	(110.6)
(180, 40)	(184.7, 56.3, 94.6)	(179.6)	(192.2)	(39.6)	(64.1)	(84.4)	(100.9)
(180, 60)	(188.3, 76.5, 174.9)	(180.9)	(203.7)	(62.9)	(91.3)	(92.7)	(301.9)
(180, 80)	(192.6, 80.7, 159.6)	(187.8)	(196.3)	(70.4)	(87.0)	(91.6)	(295.3)
(180, 100)	(179.5, 110.8, 271.3)	(177.4)	(182.1)	(96.4)	(125.8)	(67.5)	(338.4)
(180, 120)	(181.6, 123.4, 315.3)	(181.6)	(181.6)	(123.4)	(123.4)	(315.3)	(315.3)

Table 6.9: Table with grids for experiment with five tags. Tag 24622

Position	Amount of data points updates	Average deviation
(180, 0)	3	4.80 cm
(180, 20)	7	2.57 cm
(180, 40)	5	5.98 cm
(180, 60)	5	18.46 cm
(180, 80)	4	12.61 cm
(180, 100)	5	10.81 cm
(180, 120)	1	3.75 cm

Table 6.10: Amount of data points with 5 seconds and the average deviation for tag 24622

Tag 26467

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(140, 0)	(166.8, 12.4, 67.7)	(159.9)	(172.0)	(7.3)	(18.7)	(63.7)	(70.9)
(140, 20)	(136.2, 15.6, 76.5)	(113.9)	(147.6)	(7.6)	(25.1)	(68.2)	(85.6)
(140, 40)	(132.2, 23.0, 89.6)	(129.4)	(134.1)	(19.8)	(25.8)	(86.4)	(94.0)
(140, 60)	(127.0, 58.1, 206.4)	(118.5)	(135.6)	(43.8)	(72.4)	(88.8)	(323.9)
(140, 80)	(130.4, 78.2, 183.5)	(125.3)	(136.6)	(60.9)	(102.9)	(88.3)	(332.8)
(140, 100)	(145.0, 97.4, 86.2)	(144.8)	(145.3)	(97.1)	(97.7)	(86.0)	(86.3)
(140, 120)	(143.0, 116.6, 98.4)	(143.0)	(143.0)	(116.6)	(116.6)	(98.4)	(98.4)

Table 6.11: Table with grids for experiment with five tags. Tag 26467

Position	Amount of data points updates	Average deviation
(140, 0)	6	20.88 cm
(140, 20)	3	5.81 cm
(140, 40)	1	18.70 cm
(140, 60)	3	13.13 cm
(140, 80)	2	9.76 cm
(140, 100)	4	5.63 cm
(140, 120)	4	4.53 cm

Table 6.12: Amount of data points with 5 seconds intervals and the average deviation for tag 26467

Tag 26895

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(160, 0)	(165.5, 3.9, 113.2)	(157.8)	(181.1)	(-11.2)	(14.3)	(79.6)	(295.6)
(160, 20)	(160.1, 18.2, 129.4)	(137.7)	(173.6)	(-28.0)	(49.4)	(63.4)	(326.7)
(160, 40)	(164.7, 63.5, 91.5)	(156.0)	(177.5)	(57.6)	(72.2)	(84.0)	(99.2)
(160, 60)	(165.1, 87.3, 109.9)	(160.0)	(175.8)	(81.3)	(94.8)	(86.6)	(282.9)
(160, 80)	(171.1, 88.7, 144.7)	(159.7)	(179.9)	(73.5)	(110.5)	(82.8)	(323.3)
(160, 100)	(168.6, 107.9, 204.7)	(150.8)	(190.3)	(90.1)	(143.1)	(69.4)	(318.9)
(160, 120)	(164.7, 136.8, 174.4)	(161.5)	(167.6)	(132.0)	(141.0)	(78.8)	(313.9)

Table 6.13: Table with grids for experiment with five tags. Tag 26895

Position	Amount of data points updates	Average deviation
(160, 0)	9	6.74 cm
(160, 20)	10	1.80 cm
(160, 40)	10	23.96 cm
(160, 60)	11	27.77 cm
(160, 80)	11	14.10 cm
(160, 100)	11	11.67 cm
(160, 120)	5	17.44 cm

Table 6.14: Amount of data points with 5 seconds intervals and the average deviation for tag 26467

Tag 26901

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(220, 0)	(229.9, 4.9, 233.6)	(214.6)	(246.5)	(-6.9)	(15.7)	(72.4)	(318.3)
(220, 20)	(227.5, 14.8, 151.4)	(221.7)	(233.4)	(3.0)	(30.8)	(87.7)	(296.1)
(220, 40)	(228.6, 42.4, 111.9)	(221.6)	(237.5)	(31.9)	(58.0)	(90.6)	(286.5)
(220, 60)	(227.5, 61.0, 132.9)	(220.5)	(241.1)	(43.7)	(75.2)	(87.1)	(293.6)
(220, 80)	(222.8, 77.6, 86.2)	(218.4)	(235.3)	(60.2)	(90.3)	(75.3)	(91.9)
(220, 100)	(220.2, 92.2, 165.6)	(180.5)	(228.8)	(51.3)	(115.2)	(22.4)	(342.8)
(220, 120)	(222.4, 108.4, 105.6)	(215.6)	(228.7)	(97.7)	(115.5)	(73.9)	(319.7)

Table 6.15: Table with grids for experiment with five tags. Tag 26901

Position	Amount of data points updates	Average deviation
(220, 0)	9	11.04 cm
(220, 20)	10	9.12 cm
(220, 40)	10	8.92 cm
(220, 60)	10	7.56 cm
(220, 80)	11	3.68 cm
(220, 100)	10	7.80 cm
(220, 120)	10	11.84 cm

Table 6.16: Amount of data points with 5 seconds intervals and the average deviation for tag 26467

Tag 27001

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(200, 0)	(198.3, 0.3, 133.9)	(194.0)	(204.5)	(-5.8)	(3.8)	(92.7)	(289.2)
(200, 20)	(208.3, 11.7, 181.5)	(201.1)	(220.9)	(-1.7)	(24.2)	(102.7)	(294.6)
(200, 40)	(201.8, 31.8, 113.4)	(195.5)	(206.7)	(16.9)	(42.0)	(108.2)	(118.5)
(200, 60)	(205.6, 53.0, 140.8)	(200.3)	(213.5)	(37.7)	(60.0)	(112.4)	(272.8)
(200, 80)	(203.6, 71.5, 116.4)	(200.1)	(207.9)	(57.4)	(77.4)	(112.5)	(118.9)
(200, 100)	(201.6, 93.2, 112.0)	(197.5)	(205.8)	(90.1)	(96.8)	(109.7)	(115.5)
(200, 120)	(204.6, 110.8, 105.9)	(200.7)	(212.4)	(101.0)	(119.0)	(95.0)	(116.6)

Table 6.17: Table with grids for experiment with five tags. Tag 27001

Position	Amount of data points	Average deviation
(200, 0)	5	11.70 cm
(200, 20)	7	11.73 cm
(200, 40)	4	8.39 cm
(200, 60)	6	8.96 cm
(200, 80)	5	9.23 cm
(200, 100)	6	6.98 cm
(200, 120)	6	10.28 cm

Table 6.18: Amount of data points with 5 seconds intervals and the average deviation for tag 27001

6.3 Test plan

6.3.1 Documentation

All functions should be documented. This is done by writing comments in the code following the standards of the language.

For C#:

`https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/documentation-comments`

For Python:
`https://www.geeksforgeeks.org/python-docstrings/`

These comments should explain what the parameters of the function are and a short summary of what the function does.

If the function that is being implemented deals with more complex concepts, additional comments should be written in the code to explain what is happening in the function.

6.3.2 Testing

What to test

All code that can influence the user's experience should be tested. This means that every function that would influence the program negatively if it had an incorrect calculation should be tested to try to avoid this.
All public functions should always be tested.

How to test

- Unit tests
  - Functions are tested in isolation.
  - If the function needs data from outside the function it should be manually generated using mocks.
  - More information about unit testing
- Integration tests
  - Testing if different modules integrate correctly.
  - An example of this is when we are fetching data from the python server into the C# client.
  - More information about integration testing

Unit testing in unity

There are two ways of running unit tests in Unity: play mode and editor mode. Play mode requires the framework to load the scenes before testing code while editor mode can test the scripts in the project without loading the scenes. When the scenes are loaded in play mode, the objects are instantiated with the predefined data in the scene.

Since this makes it more difficult to control the state of the program, the focus will be on testing the functions isolated in editor mode. This has the consequence that some code that utilizes the Unity API and functions on presentational objects in Unity can not be tested without using the play mode.
To make it easier to unit test your code you should try to separate logic that depends on Unity features and logic that does not into different functions.

How much to test

100% code coverage is not necessary, but if there are no unit tests for a part of the system it is not enough. The focus should be on the quality of tests rather than the number of tests.
All functionality in a function that can affect the output of the function should be tested. If a bug is fixed and it took more than a couple of minutes to fix it, unit tests should be created for this bug to prevent it from being reintroduced later on.

Bibliography

[1] Ivan Aaen. *Essence. Problem Based Digital Innovation.* 2020.

[2] *Creator System Performance.* URL: https://www.pozyx.io/products-and-services/creator/system-performance.

[3] Finjan Cybersecurity. URL: https://blog.finjan.com/unicast-broadcast-multicast-data-transmissions/ (visited on 03/01/2020).

[4] Bill Fenner, Andrew M. Rudoff, and Richards W. Stevens. *UNIX Network Programming, Volume 1. The Sockets Networking API.* 3. edition. Addison-Wesley Professional, 2003.

[5] Exit Games. URL: https://www.photonengine.com/en-US/PUN (visited on 02/14/2019).

[6] Ilya Grigorik. *High Performance Browser Networking. What Every Develoepr Should Know About Networking And Web Performance.* 1. edition. O'Reilly Media, 2013.

[7] *Home: Pozyx.* URL: https://www.pozyx.io/.

[8] *Home: Pozyx.* URL: https://www.taitradioacademy.com/courses/basic-radio-awareness/.

[9] *How does ultra-wideband work?* URL: https://www.pozyx.io/technology/how-does-uwb-work.

[10] infsoft. URL: https://www.infsoft.com/technology/positioning-technologies/wi-fi (visited on 03/02/2020).

[11] infsoft. URL: https://www.infsoft.com/blog/indoor-navigation-indoor-positioning-using-bluetooth (visited on 03/02/2020).

[12] infsoft. URL: https://www.infsoft.com/technology/positioning-technologies/rfid (visited on 03/02/2020).

[13] *IPv4 Multicast Address Space Registry.* URL: https://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml#multicast-addresses-1.

[14] I. Scott MacKenzie and Colin Ware. “Lag as a Determinant of Human Performance in Interactive Systems”. In: *Proceedings of the INTERACT 93 and CHI 93 Conference on Human Factors in Computing Systems*. CHI 93. Amsterdam, The Netherlands: Association for Computing Machinery, 1993, pp. 488493. ISBN: 0897915755. DOI: 10.1145/169059.169431. URL: https://doi.org/10.1145/169059.169431.

[15] US National Coordination Office for Space-based Positioning Navigation and Timing. URL: https://www.gps.gov/systems/gps/performance/accuracy/ (visited on 03/02/2020).

[16] *Positioning protocols explained*. URL: https://www.pozyx.io/technology/positioning-protocols-explained.

[17] Unity Technologies. URL: https://unity.com/how-to/programming-unity (visited on 02/13/2019).

[18] Unity Technologies. URL: https://unity3d.com/unity/features/multiplatform (visited on 02/13/2019).

[19] Unity Technologies. URL: https://docs.unity3d.com/Manual/UNet.html (visited on 02/14/2019).

[20] Unity Technologies. *Game engines - how do they work?* URL: https://unity3d.com/what-is-a-game-engine (visited on 02/13/2019).

[21] *Ultra-wideband and obstacles*. URL: https://www.pozyx.io/technology/uwb-and-obstacles.

[22] Thomas Waltemate et al. “The impact of latency on perceptual judgments and motor performance in closed-loop interaction in virtual reality”. eng. In: *Proceedings of the 22nd ACM Conference on virtual reality software and technology*. Vol. 02-04-. VRST ’16. ACM, 2016, pp. 27–35. ISBN: 9781450344913.

[23] *Where to place the anchors*. URL: https://www.pozyx.io/technology/where-to-place-the-anchors.

[24] ZeroMQ. URL: https://zeromq.org/get-started/ (visited on 02/18/2019).

[25] ZeroMQ. URL: https://zguide.zeromq.org/page:all (visited on 02/18/2019).

List of Figures

List of Tables

Listings