Web-Based Integration of Mobile Devices Into Public Space Games

Michael Temper

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2018

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, June 26, 2018

Michael Temper

Contents

Abstract

This thesis discusses the novel concept of enhancing a public space game with smart devices via a suitable network technology. To provide a satisfying user experience, a performant connection between server and client had to be established in terms of latency, scalability and network robustness. Furthermore, a suitable interface for the interaction with the game had to be implemented under the constraints of mobile web browsers. Existing projects already use similar approaches, though they are either designed for private usage or require an installation of additional software and hence the attempt of connecting a theoretically unlimited amount of users with the system is rather unique. Therefore, *WebSockets* were used for the communication of all endpoints as they are already in use for other real time applications and further showed satisfying results in early playtests. Multiple people with varying amount of players tested the final prototype for the evaluation of the chosen web technology, followed by completing a game experience survey. Furthermore, network-related data was logged for a more precise technical evaluation. The outcome of this thesis should reveal the potential of *WebSockets* for this unique setup and furthermore, show appropriate practices to overcome presumably upcoming barriers and challenges.

Kurzfassung

Diese Arbeit setzt sich mit dem neuartigen Konzept einer Erweiterung eines Spiels für öffentliche Räume mit Smart Devices über eine passende Netzwerktechnologie auseinander. Um eine zufriedenstellende Nutzererfahrung bieten zu können musste eine performante Verbindung zwischen dem Server und Client hergestellt werden in Bezug auf Latenz, Skalierbarkeit und Netzwerk Robustheit. Außerdem musste eine passende Schnittstelle für die Interaktion implementiert werden unter Berücksichtigung der Einschränkungen von mobilen Browsern. Existierende Projekte verwenden bereits ähnliche Methoden, jedoch sind diese entweder für den privaten Gebrauch gedacht oder setzen die Installation zusätzlicher Software voraus und daher ist der Ansatz eine theoretisch unbegrenzte Anzahl an Nutzern mit dem System zu verbinden einzigartig. Daher wurden *WebSockets* für die Kommunikation aller Endpunkte verwendet, da diese bereits für andere Echtzeit Anwendungen verwendet werden und darüber hinaus zufriedenstellende Resultate in frühen Spieltests zeigten. Mehrere Personen mit unterschiedlicher Anzahl an Spielern testeten den finalen Prototyp für die Evaluierung der ausgewählten Webtechnologie, gefolgt vom Ausfüllen eines Fragebogens zum Spielerlebnis. Weiterhin wurden netzwerkbezogene Daten aufgezeichnet für eine präzisere technische Evaluierung. Die Ergebnisse dieser Arbeit sollen das Potenzial von *WebSockets* für dieses einzigartige Setup aufzeigen und zusätzlich angemessene Methoden zur Überwindung von voraussichtlich aufkommenden Barrieren und Herausforderungen zeigen.

Chapter 1

Introduction

1.1 Motivation

In recent years, playing games in public spaces has become more and more popular, such as in the case of multiplayer Augmented Reality games or on large public displays. Conventionally, such games require the installation of a dedicated application on a smartphone or other mobile device. In many public space installations, spectators may be interested in the game being played, but are not willing to install an unknown app to be able to participate. Since smartphone technologies provide a wide range of input and output modalities, and web technologies can now access a large number of these functionalities, a solution without the need to install an app would be possible. Such an approach would simplify the process of joining a public space game and providing a larger potential player base.

1.2 Initial Goals

Several points have to be accomplished to create such a system. Firstly, each endpoint of the network has to be implemented, the server and the clients. For the clients a distinction needs to be made between the mobile and the desktop version. Secondly, the controller of the mobile devices should support several input possibilities, like buttons, a joystick, swipe gestures and data of the gyroscope, to explore the constraints of common web technologies in modern mobile browsers. Thirdly, the system has to be performant even when it is used by multiple players, so it is important to choose a convenient web technology for the connection between server and clients. Fourthly, the framework has to be implemented in a way that it is independent of the game which it is used for. And finally, a game prototype has to be implemented to test the resulting framework.

1.3 Document Structure

Chapter 2 gives a basic idea of what public space games are in general and shows which technical and also social challenges have to be accomplished for designing a game for this unique setup.

Chapter 3 shows how smart devices have been used in several projects to enhance

their overall system in various ways. Furthermore, it describes the barriers which can keep people from joining the game and lastly, this chapter deals with the challenges that are entailed by this enhancement.

Chapter 4 deals with the constraints for web applications in modern mobile web browsers. In addition to common problems like the battery life, it also focuses especially on the challenge of using a smart device as a game controller such as how to design the user interface or network related aspects.

Chapter 5 explains technologies which can be used to establish a connection between a server and a client. Furthermore, the benefits and drawbacks of each of these technologies will be described, as well as the reason for the decision of which technology was chosen for the connection of the endpoints.

Chapter 6 describes the overall system in detail. The first section shows which requirements have been defined in the beginning of the project to provide a satisfying user experience. The second section describes the tools that have been chosen for the single components, like the clients and the server, and for what reason. Furthermore, the gameplay, the controls and the goal of the implemented game prototype will be explained. Lastly will be shown how each of these components have been implemented and which aspects have to be considered for establishing the connection between server and clients.

Chapter 7 shows the evaluation of the prototype. First, it will be explained how the system was evaluated in general and then, how the chosen web technology for connecting server and clients has been evaluated.

The final chapter will give a conclusion about this thesis. In the first section the results of the previously mentioned evaluation will be analized in more detail. Then, the second section shows how the project could be improved upon in future developments and lastly, the third section concludes what has been learned in this master's thesis.

Chapter 2

Public Space Games

Over the last few years, two basic forms of multiplayer gaming have evolved: a co-located form with one shared display and multiple players and a typically network-based form in which each player has an individual display. Public space games, often also referred to as urban space games or co-located games, are included in the first type of multiplayer games. The advantages of them are the theoretically unlimited number of players due to the large display and the easy way of joining them, but because of this unique setup, there are multiple design challenges which have to be considered.

2.1 Game Design Challenges

When it comes to design, there are multiple different aspects which have to be considered – most importantly technical and social implications. Various studies have evaluated the gameplay of public space games in terms of player behavior [21], additional input modalities [11] and a larger amount of players [16]. The following section will cover the most common aspects which have to be considered when designing a game for such a scenario.

Physical Space, Health and Safety

Due to the fact that public space games take place in urban areas and the theoretically unlimited number of players, it always has to be kept in mind that they can attract a lot of people. It often happens for example when certain sport events are broadcasted on public displays that people create a big crowd in front of them. This is even a bigger problem with interactive applications like games, since physical interaction, like throwing a ball, can potentially interfere with other people who are around the players. These aspects can lead to include extra policing, erection of temporary boundaries, or the use of a professional host. The final decision will be made by a screen manager who is responsible for appropriate health and safety measurements when it comes to the use of interactive applications in urban areas. So for example the use of a professional emcee and temporary physical barriers were necessary for a game during a study in Birmingham as shown in Figure 2.1, due to the physical movements by multiple players and the crowded area of the used location. In contrast to this, no barriers or professional emcees were used to accompany the game in Bradford, since the used area was not a

Figure 2.1: People playing and watching a public game at an urban area [21].

busy walkway. Further precautions, which can be made beforehand to ensure a safer gameplay, are to slow down the game pace, to increase the size of the players' bounding boxes and to try to avoid distracting players' vision in general.

All these aspects are important for the development of the final gameplay, since it can incur financial costs impacting the economics of screening these games and the frequency with which they can be made available [11, 21]. This is where the use of smart devices comes in quite handy as they can help to set constraints for the physical interaction of players without restricting the interactivity of the game.

Initiating Participation

The start of a public space game can be quite difficult due to social challenges. According to Kenton's study [21] where people were asked about the use of a game in a urban space, it often happens that people are afraid of being judged by others while playing a game on a big public screen. A lack of understanding and the exploration of the game mechanics through trial and error can be socially awkward. An observation showed that two specific aspects can help to get over these psychological barriers.

One of these aspects is the use of an emcee who can help in several ways. Firstly, he or she can explain people what is going on in general, the purpose of the game, how to play the game and where to play the game. Even when the game mechanics are simple, these explanations are a resource of understanding the activity, which can help to reduce uncertainties about how to get involved. Secondly, he can directly invite people to play the game. Lastly, the presence and actions of the emcee legitimizes the activity, which is especially important for older people who would be otherwise uncertain about their participation of the game due to their age.

The second aspect is the accidental interaction in the game. By using any kind of motion sensor or visual tracking systems it can happen that people accidentally interact with the game. These interactions can already explain simple use mechanics which helps to join the game.

Organizing Collaborative Play

It can be quite difficult to achieve a coordinated game play among strangers. Studies have shown that it comes to collaboration between the players more often when they join the game as a group due to a specific relation to each other, like friendship or family. A common practice is to design a game which can be played completely alone and also with multiple players. Furthermore, the use of smart devices can help to improve the collaboration between the players since it is not necessary to establish close physical proximity to strangers.

Spectatorship, Audience and Performance

Public games in urban areas have effects on both participant types, on the performer as well as on the spectators. The player can be motivated for his or her performance in the game by applause of the audience or motivational quotes by family and friends. On the other hand, it can also be entertaining to the audience, especially when the game has a high level of expressivity in terms of visible manipulations and visible effects, which means that the audience can see players move in the physical environment while simultaneously viewing them on the screen. Another good approach to attract more people is to foster competition between an inner group and an outer group, like often in sports, as this motivates the people to win the game [21].

Instant Drop-In and Drop-Out

For co-located games in public areas it is important that the joining and leaving of the game works as easily as possible without interrupting the general gameplay. This is necessary as not to lower the confidence of the players who want to join the game and to avoid an interruption of the game flow of people who are already playing. Preferred solutions are either a setup which does not require further hardware for the user, like motion sensors and other tracking techniques or technologies, such as QR-Codes and (shortened) links, which do not require the user to install an extra application. Furthermore, the system has to handle disconnections correctly, even when the user looses the connection in another way than originally intended, like through incoming calls or simply walking away from the system and its network.

Perspective

When it comes to the use of large displays, which are often used for public space games, the perspective can become a problem since people tend to have struggles between being close to the display to interact with the game and getting further away from it for a better overview [11]. A solution to this problem can be the use of split screens. This can even improve the gameplay since it is possible to show unique information to the players, although this also enables screen watching. Still, a single shared gameplay area also has several advantages, like a game area that scales directly with the resolution of the display and that all players focus on the same view, which can provide a greater level of collaboration and for richer social interactions. The developer has to carefully weigh the benefits and disadvantages of both possibilities and chooses the one which fits the used gameplay best [16].

Lack of Physical Barriers

It can be quite difficult to restrict areas of public space games to the player since he or she can freely walk around in the real environment. A physical barrier is not possible since it would have a negative influence on the safety of the players. A possible solution would be the use of digital barriers, like a wall in the game. This prevents the character from moving into restricted areas. The problem with this approach, however, is that immersion can be highly dependent of the direct mapping between the player and the character, which would be influenced by the disconnection of both positions.

Anonymity of the Player

Every player who joins a Co-Located Game is basically anonymous – seen from a technical point of view. This prevents the use of user specific data like highscores and names. Therefore, some kind of identification system has to be used. Normally a registration and login system can be used here, but because of the unique setup the joining has to be kept as simple as possible and so the users have to be identified in the background. This can be done by saving a specific identifier for each player on a persistent data storage, such as a data base.

Explicit Player Actions

By using visual tracking systems for urban space games, the only possibility for input can be physical movements and gestures. The problem is that these gestures are often interpreted wrongly or not detected at all. Furthermore, these gestures have to be learned from the players before they can join the game. A solution to this problem can be the use of virtual triggers like buttons. The disadvantage of this workaround is that the trigger can be quite imprecisely due to the fact that the players first have to move physically to the element before they can activate it.

Individual Player Feedback

A big shared display makes it difficult to give single players individual feedback. Since there is only one source for output, the feedback can be easily overseen and overheard. It is important to solve this problem as the feedback shows the player the resulting effects of his or her actions in the game. This is necessary to prevent social inhibition as the uncertainty of the player's action can feel awkward to the users and in turn keep them of from joining and playing the game [11].

Public space games entail a whole new series of challenges due to their unique setup. To provide the players a satisfying user experience, each of them has to be handled in a good manner and to accomplish this, the use of smart devices can be quite helpful. The following chapter will show several projects where games have been enhanced with smart devices and how this influences the overall setup.

Chapter 3

Smart Gaming

One of the fastest growing novel technologies in the mobile phone market is the smartphone. Its main purpose is to fulfill tasks both at home and at work [1]. Furthermore, they can be used for entertainment, like gaming, but besides playing games directly on the smartphone it is also possible to use them to enhance other systems. Using a smart device for extending a public space setup will be referred to as smart gaming in this thesis due to its variety of usable interfaces. Smart devices contain several sensors, like the touch sensor and the gyroscope, which offer the user various input modalities, even though it is important to note that some of these sensors should not be used for precise tasks as a drift of the correct data can occur. In addition to multiple input possibilities, individual player feedback can be given to the player by the use of available output components, like the speakers, the vibration motor or the display itself. Furthermore even location-based data can be accessed by the GPS sensor of the device and lastly, most smart devices include a network interface and can therefore be connected to any other network participant, like servers or other clients. All these functionalities offer people a whole new gaming experience since they can, in the best case, easily join the game without the need of further hard- and software such as additional sensors or specific applications.

3.1 Related Projects

Numerous projects use smartphones as an extension of their final products. They either use it as primary controller, to enlarge their number of interaction modalities or to give the users new controlling mechanisms. This section will describe these projects in general and show the positive as well as the negative aspects of each of them.

3.1.1 MoCo

This project comes from the master's thesis “Integration of Mobile Devices into a Floor-Based Game to Increase Player Dynamic” which shows the diversity of co-located floor-based video games and elaborates on how the user interface in such games can be extended. A particular focus lies on how to give the player more control over the game and more ways to perform explicit actions during the game [11]. The final result is a

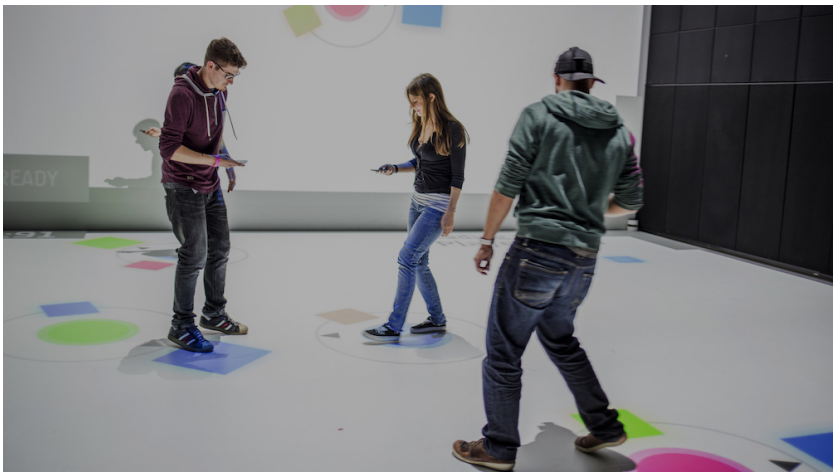


Figure 3.1: People playing *Lazor Arena* based on *MoCo*.

framework called *MoCo*[1]. It is implemented in the game development platform *Unity*[2] and gives users more input and output possibilities for two specific floor-based games, *Lazor Lab*[1] and *Lazor Arena*[1], via the use of a smartphone as shown in Figure 3.1.

Both games only last for a few minutes, but offer two different gameplay styles. *Lazor Arena* has a competitive mode where the goal is to shoot as many players as possible while not getting shot by others to keep them off from getting points. Shooting is possible after energy is collected by picking up digital cubes, while the use of a shield for protection limited by a certain time threshold. *Lazor Lab* has a more collaborative gameplay where all players have to defend the base from enemies for as long as the game lasts. Enemies can be destroyed by either shooting them as in *Lazor Arena* or by shooting them with a stationary, more powerful laser. The latter mentioned option is especially necessary for better protected enemies and only available if three different kinds of energy have been collected. Furthermore, the players have to repair bricks on the border of the game area to reduce the amount of incoming enemies.

The project is used in the *DeepSpace*[3] at the *Ars Electronica*[4] museum in Linz, which is a multifunctional room for interactive media, games and presentations. This unique setup includes a laser-based tracking system for interactions and eight projectors for two rather large displays on the floor and on a wall. The projection on the wall can be configured to either mirror the floor projection or to show additional content [11].

The general idea of the project is that users step into the tracking area while holding a smart device which is connected to the local network. Then, the user connects to the game server either by entering a specific IP address or is automatically connected with the help of a broadcast by the server. After the player has been assigned with the correct tracking ID the smart device acts as an extension of the game as the actual game still takes place on the projections. This extension entails several advantages to the whole

[1] http://lazorlab.rohschinken.at/en/
[2] https://unity3d.com/
[3] https://www.aec.at/center/ausstellungen/deep-space/
[4] https://www.aec.at/

Figure 3.2: User interface of mobile devices in *Lazor Arena* [11].

system. Firstly, individual player feedback is possible as personalized audio and tactile feedback is given to the user as soon as any kind of interaction appears. Secondly, the user can send explicit actions such as a permanent transmission of sensor data by using the smartphones built in gyroscope for aiming in the game and input commands to trigger actions like grabbing energy and activating the shield as seen in Figure 3.2. Lastly, player profiles can be created and therefore the player's nickname and score of individual games can be saved on the smartphone.

However, there are still open issues as *MoCo* is more of a prototypic proof of concept than a final product. First of all, the player has to calibrate his device since both game mechanics are depend of the gyroscope. During a play test, the author of the master's thesis in which *MoCo* was presented could experience by himself that some devices gyroscope drift apart from the original calibrated values during the gameplay. So the player has to calibrate his device repeatedly since otherwise the aiming does not work in the correct way. So both of them, the constant recalibration and the wrong aiming, can lead to frustration after some time, which in turn leads to a bad user experience. Moreover, the input provided by the smartphone is reduced to two modalities, the buttons and the gyroscope. This is not exactly a problem since the game already works with these controls, yet there is still space for improvement by adding additional input possibilities, like for example swipe gestures. And last, the whole project was implemented in *Unity* which is a barrier for joining the game as players have to install an additional application on their smartphones before they can start playing.

3.1.2 Super Sync Sports

The *Chrome Experiment*[5] *Super Sync Sports*[6] is a collection of three web-based mini games which uses smartphones and tablets as game controllers. The connection is based

[5] https://experiments.withgoogle.com/collection/chrome
[6] https://experiments.withgoogle.com/chrome/super-sync-sports

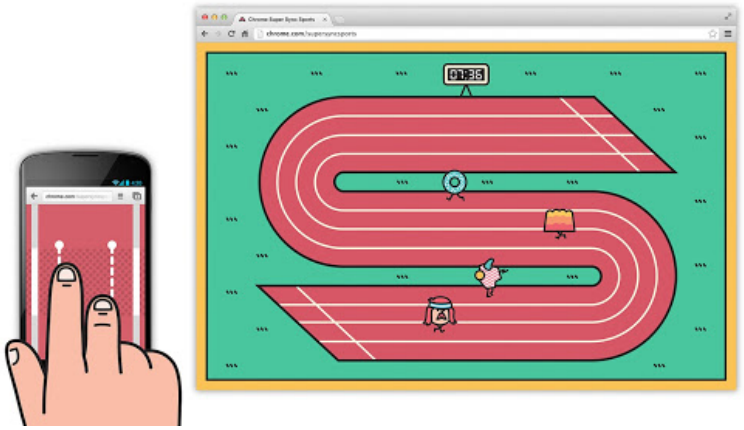


Figure 3.3: Game preview of *Super Sync Sports* [33].

on *WebSockets*[7] and the controls work with swipe gestures and buttons.

First, the player has to start the game in the browser on a desktop computer. Then, the user has to connect his smart device by entering a specific code for synchronization on the website of the game. When the synchronization was successful, the player can decide to play either alone in the single player mode or play with up to three other people in the multiplayer mode. Afterwards each player can choose one of 50 unique characters and can decide to either run, swim or cycle against each other. The tracks of each game are rather short and so are the game rounds. The players can control their virtual characters with swipe gestures on the smart device, as mentioned before. So for example to move the player, two fingers have to swipe alternately from top to bottom on the mobile screen, as seen in Figure 3.3 or swipe with two fingers simultaneously in a circular manner to ride the bike. In the end the players can see their final rank on their smart devices as also a ranking of the best three players, including their measured time, on the desktop computer.

As mentioned before there are still several restrictions of this setup. Firstly, there are only three games available, whereas the duration of each of them is rather short which can quickly lead to boredom of the player. Secondly, the maximum player count is up to only four players. And lastly, the input can only be done with two modalities where, in turn, the controls of the character during the game are even restricted to only one input possibility.

3.1.3 AirConsole

AirConsole[8] is an online platform that transforms the browser into a video game console and smartphones into gamepads. It is a commercial project that can either be used for free with advertisements between the game rounds or with premium services that have to be paid for.

[7] https://www.websocket.org/
[8] https://www.airconsole.com/

Figure 3.4: *AirConsole* game prototype.

Various games are offered by an online library and developers can use an application programming interface (API) to create their own game and controller. They even provide a simulator for debugging. It is a good way to test your game since you can start multiple virtual mobile clients. Furthermore there are plugins for *Unity*, Construct2/Construct3 and several libraries, like a controller generator, which can be used for the implementation.

As a first step of the project implemented for this master's thesis the *AirConsole* API was used for a game prototype as shown in Figure 3.4, to see how the connection between the smartphone and browser is handled and how the performance is in terms of latency. The result was satisfactory, the API offers a lot of methods for messaging between the controller and the screen, device states, user profile handling and more. The latency was barely noticeable even with multiple players. The problem with this approach is, however, that there are no insights into the implementation behind the API and so it is only possible to stick completely to the structure and the interface it offers.

3.1.4 PlayLink

One of the latest developments in the area of smart gaming is *Sony's PlayLink*[9] for the *Playstation 4*[10]. It describes a series of exclusive games which can be controlled by smart devices instead of the usual gamepads. *PlayLink* was released in the year 2017 with the game called *That's You!*[11] as shown in Figure 3.5, which is a comedy quiz for multiple players. Several tasks have to be accomplished in the game, like answering questions about each other, drawing specific activities, capturing photos with given facial expressions and more.

For the connection the *Playstation* establishes a WiFi-Hotspot, which makes it possible to play the games even without an internet connection. Each game has its own

[9] https://www.playstation.com/en-gb/explore/ps4/games/playlink/
[10] https://www.playstation.com/en-us/explore/ps4/
[11] https://www.playstation.com/en-gb/games/thats-you-ps4/

Figure 3.5: In-game content of *Sony's That's You!* [37].

application which has to be installed on the personal smart devices first, before the games can begin. The last step is to connect to the previously mentioned hotspot and then everyone can start playing.

However, this project has a major drawback. Since it is a commercial product, additional hard- and software has to be bought before actual gamplay is possible. For the whole setup a personal television, smart device, console and one of the games is necessary, which makes it a rather expensive setup for the users compared to similar projects.

3.2 Cross-Device Interaction Barriers

Interactive public displays in urban areas bring up several problems, which have to be solved. A study has shown that these problems can be divided into three different categories as shown in Figure 3.6: *Attraction*, *Interaction* and *Management* [8]. This section will describe these issues and shows approaches on how to solve them.

3.2.1 Attraction

To initiate any kind of interaction, the display has to be firstly visible to the users, secondly attract them, thirdly represent as an interactive medium and lastly also show the connection to smart devices.

A solution to the problem of people not seeing the display is to react to the presence of passersby if tracking capabilities are available. This can even lead to accidental interaction, which causes sometimes that people start interacting with the content shown in the display [21]. Then, it can come to the *honeypot effect* which means that people watching another person playing the game leads them to even join the game by themself [4, 8, 11]. Also, music and sound effects can be used to attract the users attention, but it is noteworthy to not overwhelm the people with too much visual or auditive effects since this could otherwise lead to the oposite effect. To show the users that the display is interactive, signs or instruction labels can be provided next to the display. These static instructions can either be enhanced by digital ones like videos and animations or be re-

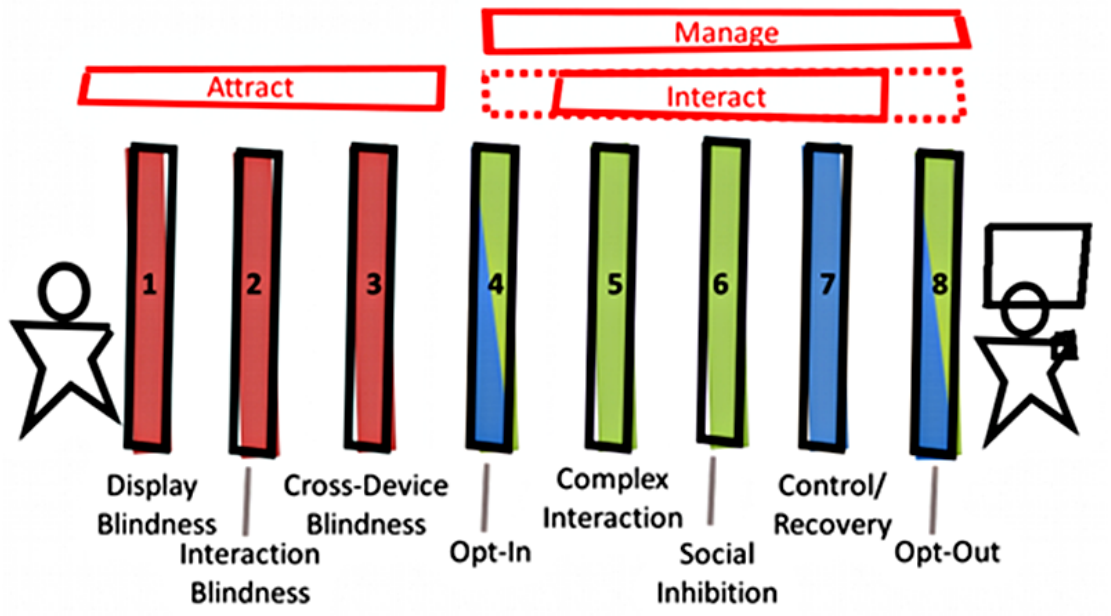


Figure 3.6: The ordered cross-device barriers [8].

placed by them. Another approach to explain the audience how the whole system works is to assign an on-site instructor. This solution delivered already good results in another study. Lastly, it is important to show the users that they can use their smartphone for interacting with the display, which can either be done by showing a symbol or a video of someone using the smartphone with the display can be shown or even simpler, by showing a QR-Code which is implicitly a hint for users who have used this technology already before [8, 11].

3.2.2 Interaction

The requirements of the system should be kept as low as possible to increase the probability of people joining it. This means that, on the one hand, the system should be uncomplicated in general and on the other hand the joining should be as easy as possible, which means that the scanning of an QR code or the entering of an (shortened) URL would be preferable over the installation of a specific app. Another important aspect is to show the user all input possibilities of the system and how they work. Since mobile devices, as the large displays themselves, can have several different input modalities, such as touch sensors, accelerometer or full body gestures, it is important to provide multiple ways to interact with the system, because of the different hardware equipment of each device. For example, if a game uses the accelerometer of the smartphone to control a character, then the movement should be also possible by touch in case that the smart device is not equipped with an accelerometer [2, 8]. To prevent social inhibition, which often keeps people from joining the system, it is important to provide users feedback during display interaction, device connection, or connection failures. Otherwise people could be afraid of being socially embarrassed.

Another way to convey screen interaction possibilities to the user, is to first let them test the actual gameplay, which means that they have a short time to test all input possibilities, like in the game *Super Sync Sports* by *Google*[12] mentioned in the previous section [8].

3.2.3 Management

To support multiple connected smartphones at the same time, the necessary bandwidth should be minimized, which means that the amount of sent messages between all devices should kept as low as possible [8]. For example, in some games, depending on the gameplay, it is not necessary to constantly update the position of the player via the network, when it is also possible to just send movement commands instead. Minimizing the bandwidth is especially important for games where responsiveness is a key factor for playability and user experience [11]. Another important requirement for the system is that the users should only join the game on their own volition. Any kind of automatic download or opening of an app could otherwise break the user's trust [13]. This can also prevent accidental interaction, which can, on the one hand, attract people to join the game [21], but, on the other hand, also become an issue when someone is already playing the game. Lastly, it is important that proper recovery and control mechanisms exist as people do not always disconnect from the game as intended, due to inactivity or long distances from the display. Also, the reconnection to the system should be possible since mobile devices can also be interrupted by incoming calls or connection failures. For example, in games the character of players who are either disconnected or be inactive for a specific amount of time should be put in some kind of idle mode, like freezing it or removing it temporarily from the game scene, since otherwise negative consequences, like loosing points, could appear [8, 11].

3.3 Design Challenges of Smart Gaming

Adding a second display to games by using smart devices entails additional cognitive and social challenges which have to be accomplished. The following section describes the main characteristics of them and shows approaches how they can be solved.

3.3.1 Cognition Challenges

Using multiple screens can lead to a higher cognitive load for the players, due to constant need to refocus their attention. Furthermore, this also depends on how much the user is overwhelmed by the information shown on the screens and the uncertainty about focusing on the correct display. To prevent this problem the amount and complexity of the game content and information shown on the screen should be reduced. Additionally, the most important parts should be highlighted in any way so that the players immediately pay attention to them. Furthermore, the user interface should be as consistent as possible to avoid that the user has to look for essential game elements over and over again. For example, buttons for any actions should stay at the same position during the whole gameplay. Another way of showing the user on which display he or she has to

[12] https://www.google.com/

focus is the use of audio signals, since they are independent of where the user is actually looking at. Another solution can be the implementation of a gameplay with explicit temporary phases for each display.

An additional cognitive challenge is information distribution, which can be positive in terms of enhancing the game by showing different private content to each player, but also entails several aspects that have to be considered. Firstly, players have to know if a information and its value is private or public at any time. Secondly, the game has to be balanced and it has to be carefully considered which player receives which information and lastly the distribution has to be as transparent as possible to avoid that players feel treated in an unfair way [10].

3.3.2 Social Challenges

As soon as more than one person is involved in the game – which probably is the case in the majority of Smart Gaming settings – the playing session becomes a social situation. In such situations, the game experience is supposed to be significantly influenced by the interaction between the people who take part in the playing session [3, 10].

Game designers have to keep in mind that decisions during the implementation can indirectly influence the resulting social interactions. This can lead to rather neutral processes such as bluffing, bragging and trash talking in competitive settings or team building, construction of shared awareness and helping in collaborative games, but also to negative ones such as insulting other players, which have to be avoided.

Another challenge is the variety of the players. Since public games are have no specific target group to attract as many people as possible, they are independent of the age, gender and, most importantly, the experience of each player. This can lead to two problems, either experienced players get bored because of the low difficulty of the game or casual and completely new players are overwhelmed by the complex gameplay. So it is important to find a good balance between both extremes. A solution to this problem can be the introduction of various player roles, to offer different levels of difficulty. Furthermore, it is also possible to either let the player choose his own difficulty or detect a fitting level of difficulty by an algorithm depending on the users behavior and performance [6, 10].

Enhancing games with smart devices is a good way to overcome problems of public space games as shown in the previous chapter. Several projects use this technique for various purposes like increasing the amount of input modalities for floor based games or using smartphones as game controllers. However, it entails again a new set of problems. Besides cognitive and social challenges, there are additional constraints in terms of technology. Due to the unique approach of a completely network-based setup, they will be discussed in detail in the following chapter.

Chapter 4

Constraints of Mobile Web Browsers

When it comes to web development for mobile devices compared to desktop computers, there are some differences which have to be considered [20]. Especially the unique setup of using the smart device as a game controller leads to even more problematic aspects. In this section the most common differences and problems, which appeared during the implementation of the final project of this thesis will be discussed. Furthermore, it will be shown how these problems were handled to avoid them in future projects.

4.1 User Interface

There are some constraints for a well designed user interface of mobile web pages. Since the screen size is rather small, it is important that each element is easily readable and interactable. The use of less but bigger elements, as shown in Figure 4.1, is preferable so that the players can also use the device without looking at the actual screen of the smart device. During play tests it showed up that in specific situations some kind of workarounds would be preferred. For example, the buttons were enlarged without changing the size of the used sprite. This means that the buttons look normal to the users in general, but they can also be pressed by touching the screen above or below them, which helps that the user can play the game without looking directly on the screen of the mobile device.

Another problem is the wide variety of smartphones and their different screen sizes. For all elements percentages should be used instead of hardcoded values in terms of size and position so that the content of the screen looks the same on every device.

The last aspect which has to be considered in terms of the user interface are the browser's *Bars*[1]. Unfortunately it is not yet possible to hide them on all of the most popular mobile web browsers, which means that elements should not be placed directly at the top of the screen, to avoid that players accidentally click on any bar while they play the game.

[1] http://www.newmediabytes.com/2008/12/28/understanding-web-browser-anatomy-terms/

Figure 4.1: User interface of the used controller in this thesis project.

4.2 Touch Sensor and Default Operations

Touchscreens offer an easy way to control the interface of a mobile device, can detect the touch of one or multiple fingers and can even track the movement of them. The data from the touch sensor is accessed in JavaScript using the *W3C*[2], an international community which develops *Web standards*[3], Touch Events API. This provides users completely new functionalities on mobile web pages like drawing on the screen with a finger or navigating through different elements by swiping on the screen.

There are event handlers for the start of a touch, for the movement of a finger and the release of a finger which can be used to create the desired behavior of a web application. It is even possible to detect which page element was touched, which can be useful for buttons for example, or to the detect the position of the finger on the screen in general, which can be useful for a joystick functionality for example.

Depending on the used mobile browser, there are different default operations which will be fired for specific actions. These operations can become annoying for the unique use case of using the smartphone as a controller and can even destroy the gameplay. One of these problem is the possibility of zooming in browsers. There are two different ways how a user can zoom, either with using pinch-to-zoom with two fingers or by double tapping the screen. Both of these actions appeared during play tests, either by using multiple elements, like joystick and buttons, simultaneously or by hitting buttons multiple times in a short amount of time. The problem is that the user then only sees a small part of the whole controller which makes it impossible to use all elements on the screen and this finally impedes the complete gameplay. Fortunately, it is possible to prevent zooming, but it is important to know that this prevention has to be used for every single element on the screen. Another problematic default operation is the functionality of scrolling. Since the smartphone is used as a controller it is important that the user knows where he has to press so that he can play the game without even looking at the screen of the smart device. The use of scrolling would make this impossible since the elements would be displaced all the time. The last default action which should be

[2] https://www.w3.org/Consortium/
[3] https://www.w3.org/standards/

prevented is the highlighting of elements. In some browsers an element gets highlighted by default, which means that it gets surrounded by a colored rectangle after it got pressed. This does not directly affect the gameplay, but it still should be prevented since it could distract or confuse the user.

4.3 Battery Life

Current smart devices have a battery life which last for one or two days. Thus, it is important to create websites and applications that do not consume too much energy. There are several aspects which help to save energy.

Firstly, it is a good practice to update the display only when needed. Secondly, JavaScript code should not be executed often, so it is preferable for example to use CSS3 transitions and animations, which are way more CPU-efficient, instead of JavaScript animations. Lastly, it is important to use the sensors of mobile devices, such as geolocation or orientation rarely. This applies also to network related code. So instead of constantly updating the position of a player via the network, the gameplay could be also implemented in a way that the movement and the resulting position will be made directly in the game.

4.4 Network Bandwidth Speeds and Latency

The speed of mobile networks relies on multiple various factors, like the distance to the cell tower mast, the amount of people connected to the same mast and also environmental aspects like the density of surrounding buildings and the weather. Another important factor is latency, which is defined as the time between sending and receiving a package to and from the server. Especially in games it is important that the actions feel very responsive. This is ensured in local wireless and wired networks due to the low latency, but since public space games are placed in urban areas, often a mobile network is used, which is, like mentioned before, influenced by various factors. High latency affects every request and therefore JavaScript files should be minified and concatenated. The same applies to CSS files. Furthermore, sprites with all images combined should be used instead of single images. Another helpful technique would be the use of *parallelization*. That means assets will be downloaded via multiple domains, due to the simultaneous download limits of a single domain. This process should be used wisely, since each DNS lookup takes time which can lead to the opposite effect of what was initially intended.

4.5 On-Board Memory Size

Normally, web developers do not have to worry about on-board memory, since most entry-level desktop computers have at least 2 GB on-board memory. This changes when it comes to web development of mobile devices, due to the rather small memory of around 512 MB, whereas only the half of it is available for the running application because of the operating system and background applications. Especially when it comes to images, which are saved as pixel data in an uncompressed form, it is noteworthy that even when they get resized after receiving them in a large size, they still need the

same amount of memory as the larger image. This also applies to JavaScript files which are transmitted in a compressed way but will be uncompressed again on the device. Hence, it is important to use as few and small files as possible to present a website as each of them consumes memory. Otherwise it is necessary to close background tasks and applications to free up memory, which will affect the user's mobile experience in a negative way.

4.6 Operating System Responsiveness

Users should be provided by a certain degree of responsiveness since they expect it of applications. To prevent a negative user experiences, there should be a visible reaction after the amount of 300 ms when an action was made. For example, when the player clicks a button, which triggers a networked related task, and there won't be any response due to high latency or connection speed, the user should see that his action was detected and that it is still in progress. A way of showing it can be the use of a spinning wheel indicator. This problem becomes exaggerated by the fact that the standard click event handler gets triggered after 300 ms, which is necessary to detect double-taps. Otherwise every action would be interpreted as a single tap.

4.7 Orientation and Direction Sensors

The orientation sensor provides data about how the device is rotated in the x-, y- and z-axes. The rotation around them may respectively be referred to as roll, pitch and yaw, or expressed in degrees of beta, gamma, and alpha rotation as shown in Figure 4.2.

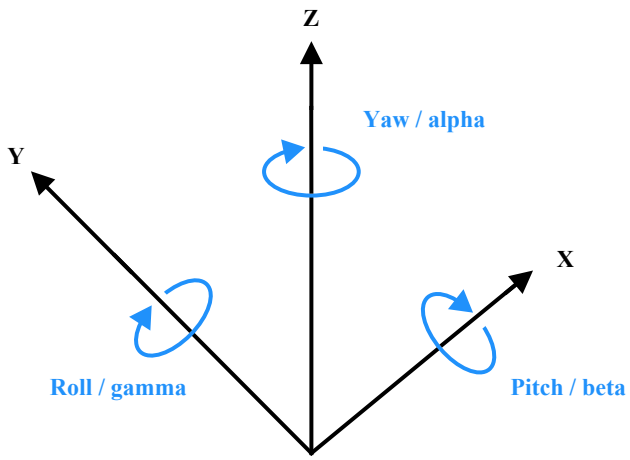


Figure 4.2: Rotation around the x-, y-, z-axes of a mobile device.

The orientation changes can also become laborious when it comes to the unique use case of using the smart device as a controller. This showed up during play tests while using the gyroscope based controller, which means that the player movement is dependent of the current device orientation. The main problem is that the smartphone automatically switches from landscape mode to portrait mode when the mobile device

reaches a specific angle on the x-axis. This can lead to a completely different layout of the shown interface, which makes it impossible to continue playing until the device gets rotated back to the landscape mode. A good practice is to keep the borders for the implemented orientation events rather low. So for example that the character already moves when the orientation was changed for only ten degrees. This prevents that the user rotates the device too much, which would trigger the native display orientation change.

4.8 Network Connection Failures and Offline States

A common problem of mobile devices is the loss of network connection. The problem in this scenario is that it is not possible to inform the server about the drop of the network. Even though it is possible to detect the offline state locally, it is not possible to send that information to the server since the connection is closed already before. This can then influence the whole gameplay since offline players will be still in the game. One solution would be the use of a constant signal from the clients to the server in specific time steps and if a signal does not get received for a longer time the character gets automatically removed from the game.

4.9 Audio and Vibration

Audio can be quite useful when it comes to the unique scenario of public space Games because of the possibility to give the users personal feedback. However, some mobile browser prevent playing audio without any user interaction to improve the user experience and limit unnecessary bandwidth or resource usage. Furthermore, it is not possible to play multiple audio sources at the same time [26]. Another approach of giving the user individual feedback would be the use of vibrations. Unfortunately, again not all mobile browsers support this functionality yet, therefore, to keep some kind of constancy for all devices, it is not recommended to use these haptic feedbacks yet, but they should still be kept in mind for future projects.

As this chapter shows there are several constraints in the development of applications for mobile browsers, due to their rather small size, operating systems and more. Besides the implementation of the input interface it is also very important to find a satisfying technology for the connection between server and clients in terms of latency, scalability and a robust network. For this reason, the most commonly used technologies will be discussed in the following chapter.

Chapter 5

Web-Based Server-Client Connection Technologies

Many Public Space Games are played in real time, which means that the games have to respond accordingly to the players input within a specific amount of time to offer a good user experience. To accomplish this, a network technology, which provides low latency for the whole system has to be chosen. Furthermore, the technology has to work performant in terms of scalability. That means that the systems performance has to be independent of the amount of players in the game as the number of players is theoretically unlimited in Public Space Games. And lastly, the network has to be robust, which means that connections and disconnections have to be handled in a good way as interruptions, such as incoming calls, and unintentional disconnections of the system, like people just walking away, can appear. This chapter describes the most commonly used connection technologies between two endpoints, such as server and client, in general and discusses the benefits and drawbacks of each of them.

5.1 Polling

The *Hypertext Transfer Protocol*[1] (HTTP) is a request/response protocol which is used for the communication between server and clients. In the standard HTTP model only the clients can initiate a connection and send requests to the server and not the other way around. Normally, web browsers send requests to the web servers, which will acknowledge the request and then send a response back. To receive the most up-to-date information from the server, the clients have to send periodical requests. This technique is called *Polling*[2], which is rather inefficient as it can happen that many requests are made while there is not even new content available and so negative responses get sent back and the clients close the connection. Furthermore, the data is queued until the server receives the next poll request from the client which reduces the responsiveness of the system. It is a good solution if the point in time of new information is known or predictable, as the client requests can be synchronized with it, but that is often not the case [25, 28].

[1] https://www.w3.org/Protocols/

[2] https://tools.ietf.org/html/rfc6202

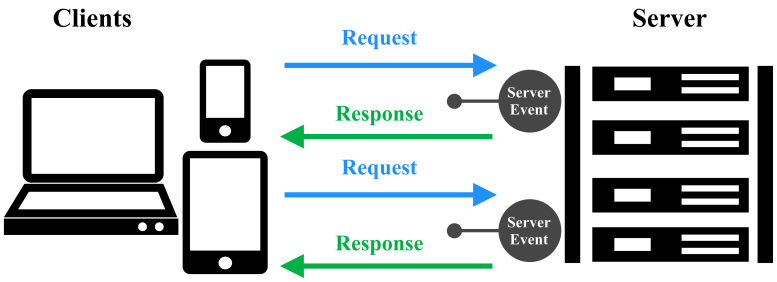


Figure 5.1: Concept of *Long Polling*. Image based on [34].

5.2 Long Polling

An improvement of this approach is the so called *Long Polling*[3]. The basic concept of that technique is similar to the previous one, but instead of sending an immediate response back to the client, the server waits and holds the request open until new information is available or the end of designated timeout is reached as shown in Figure 5.1. After the client receives the response a new long poll request is sent right away. Although this technique is an improvement overall, it still has several issues. Firstly, each request and response is a complete HTTP message and so they contain a full set of HTTP headers which can be, especially for small messages, significantly larger than the real payload carried by HTTP. Secondly, the maximal latency can be an issue. The average latency of *Long Polling* is close to only one network transit as the server waits for new information and send it then back to the client, but since the server has to wait for a new request after the response was sent, the maximal latency is increasing. And as HTTP is carried over the *Transmission Control Protocol*[4] (TCP) it can increase even more, since lost packages will be retransmitted. Lastly, *Long Polling* still opens and closes TCP/IP connections like *Polling* and so if the amount of messages between server and client is high, the performance improvements are barely significant [25, 28].

5.3 Streaming

Another similar approach is *Streaming*[3]. The difference to the previously mentioned techniques is the constantly opened request, even after a response was sent. The server and clients do not have to open and close the connections every time when a request and a subsequent response were sent, which in turn leads to a significant reduction of the networks latency. Loreto [25] describes the basic life cycle of an application using HTTP streaming as follows:

1. The client makes an initial request and then waits for a response.
2. The server defers the response to a poll request until an update is available, or until a particular status or timeout has occurred.

[3] https://tools.ietf.org/html/rfc6202
[4] https://tools.ietf.org/html/rfc793

3. Whenever an update is available, the server sends it back to the client as a part of the response.
4. The data sent by the server does not terminate the request or the connection. The server returns to step 3.

Nevertheless, *Streaming* has unwanted issues. Intermediates, such as proxies and firewalls, can buffer the response of a server which increases the latency of all message deliveries. Furthermore, the maximal latency can again increase since browser techniques used to terminate HTTP streaming connections are often associated with JavaScript and/or DOM (Document Object Model) elements that will grow in size for every message received. To prevent a growing memory in the client, the stream has to be terminated from time to time and a new request has to be sent to establish the stream again. In this scenario the sequence is similar to *Long Polling* and furthermore the connection is again based on TCP/IP which can lead to retransmission of lost packages. Both of these aspects increase the maximal latency [25, 28].

5.4 WebSocket

Both of the previously mentioned techniques are based on the HTTP which was designed for document sharing and not for rich interactive web applications like public space games. HTTP is a stateless protocol, meaning that each request is treated unique and independent, which has the advantage that no additional data about the session has to be saved on the server but, in turn, leads to the major drawback of unnecessary overhead data as each message includes redundant header information. Furthermore, HTTP is half duplex, which is rather inefficient as each information, from the server to the client and the other way around, flows in one direction at a time [28].

All this can be improved by the use of a *WebSocket*, which is a full-duplex, bidirectional, single-socket connection. This means that both, server and client, can use the same connection for messaging and further it is also possible to send data from the server to the client without receiving a request first, which reduces the latency overall as shown in Figure 5.2.

Additionally, *WebSockets* also work with firewalls and proxies, which were problematic for the approaches mentioned in Section 5.3. As soon as *WebSockets* detect a proxy server, they establish a tunnel by requesting the proxy to open a TCP/IP connection on a specific host and port. Thereby, an unimpeded communication between server and client is possible.

Furthermore, there is an API available that enables web pages to use the *WebSockets* protocol. The protocol is based on HTTP which makes it fully backward compatible as it was designed to work well with the existing Web infrastructure. To switch the protocols, a so called *WebSocket* handshake is attempted. The browser sends an upgrade request via HTTP to the server and if the server supports *WebSockets* the protocols will be switched. Once the upgraded connection is established, the server and client can communicate in full-duplex mode. The data gets framed with only two bytes, which removes all the previous overhead. To show the amount of how much the overhead is reduced, Peter Lubbers and Frank Greco made a comparison between *Polling* and *WebSockets* for a typical use case scenario [36].

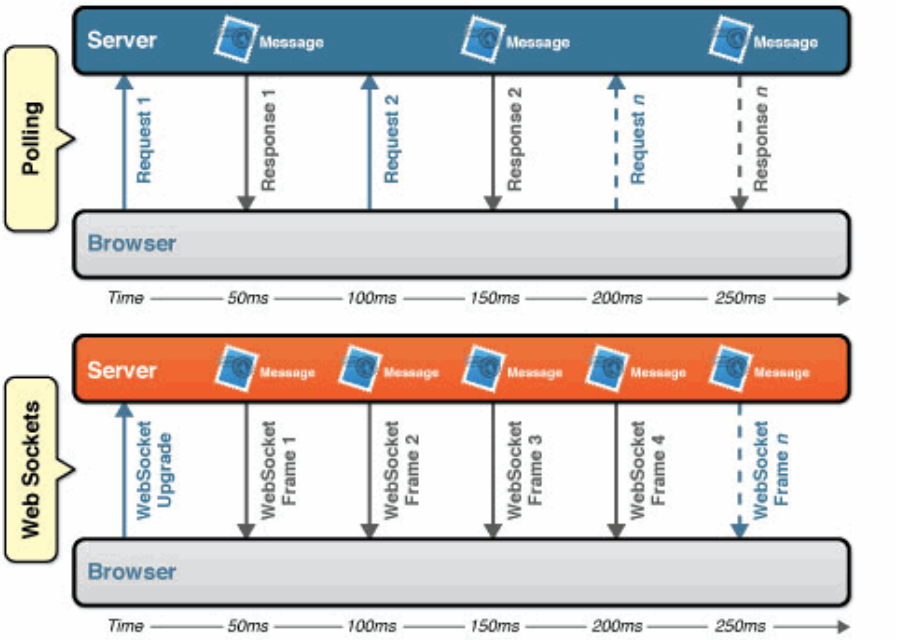


Figure 5.2: Latency comparison between *Polling* and *WebSocket* applications [36].

The basic idea of the experiment was to request real-time stock data from a message broker in time intervals of one second. This approach worked generally fine but the amount of overhead in all messages was huge in comparison with the actual data. Program 5.1 and Program 5.2 show the request and response header of a single message without any data. Both headers together contain 871 bytes while the data for a typical stock topic message is only about twenty characters long. It is noteworthy that headers with less than 871 bytes exist, but, on the other, hand there are also headers with more than 2000 bytes, so the amount of bytes in this example is around average. Still, it can already be seen that the amount of unnecessary header information is much more than the actual data.

To show the difference in a scenario with a large number of users, they set up three different use cases where only the network throughput of the request and response headers is considered:

- **Use Case A:** $1,000$ clients polling every second: Network throughput is $(871 \times 1,000) = 871,000$ bytes $= 6,968,000$ bits per second (6.6 Mbps).
- **Use Case B:** $10,000$ clients polling every second: Network throughput is $(871 \times 10,000) = 8,710,000$ bytes $= 8,710,000$ bytes $= 69,680,000$ bits per second (66 Mbps).
- **Use Case C:** $100,000$ clients polling every second: Network throughput is $(871 \times 100,000) = 87,100,000$ bytes $= 696,800,000$ bits per second (665 Mbps).

It is already visible that the amount of unnecessary network throughput is enormous. To make the comparison, the whole setup was also implemented with *WebSockets*, where the amount of overhead is just two bytes instead of 871.

```
1 GET /PollingStock//PollingStock HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.5)
4 Gecko/20091102 Firefox/3.5.5
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-us
7 Accept-Encoding: gzip,deflate
8 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
9 Keep-Alive: 300
10 Connection: keep-alive
11 Referer: http://www.example.com/PollingStock/
12 Cookie: showInheritedConstant=false; showInheritedProtectedConstant=false;
13 showInheritedProperty=false; showInheritedProtectedProperty=false;
14 showInheritedMethod=false; showInheritedProtectedMethod=false;
15 showInheritedEvent=false; showInheritedStyle=false; showInheritedEffect=false
```

Program 5.1: HTTP request header of the example scenario [36].

```
1 HTTP/1.x 200 OK
2 X-Powered-By: Servlet/2.5
3 Server: Sun Java System Application Server 9.1_02
4 Content-Type: text/html;charset=UTF-8
5 Content-Length: 21
6 Date: Sat, 07 Nov 2009 00:32:46 GMT
```

Program 5.2: HTTP response header of the example scenario [36].

- **Use Case A:** $1,000$ clients receive one message per second: Network throughput is $(2 \times 1,000) = 2,000$ bytes $= 16,000$ bits per second (0.015 Mbps).
- **Use Case B:** $10,000$ clients receive one message per second: Network throughput is $(2 \times 10,000) = 20,000$ bytes $= 160,000$ bits per second (0.153 Mbps).
- **Use Case C:** $100,000$ clients receive one message per second: Network throughput is $(2 \times 100,000) = 200,000$ bytes $= 1,600,000$ bits per second (1.526 Mbps).

As Figure 5.3 shows, the amount of overhead data is highly reduced with the use of *WebSockets*. Furthermore, less latency is introduced since the server can send messages without waiting for client requests. These two advantages make *WebSockets* highly recommendable for building real time web applications such as public space games.

5.5 WebRTC

Normally, the browser communicates with the webserver either via HTTP or *WebSockets* which are both based on TCP as mentioned before. The basic work flow is that the client sends a request to the server and the server sends a response with the requested content, like webpages or images, back.

Another approach to connect and exchange data with the server is the use of the *Real*

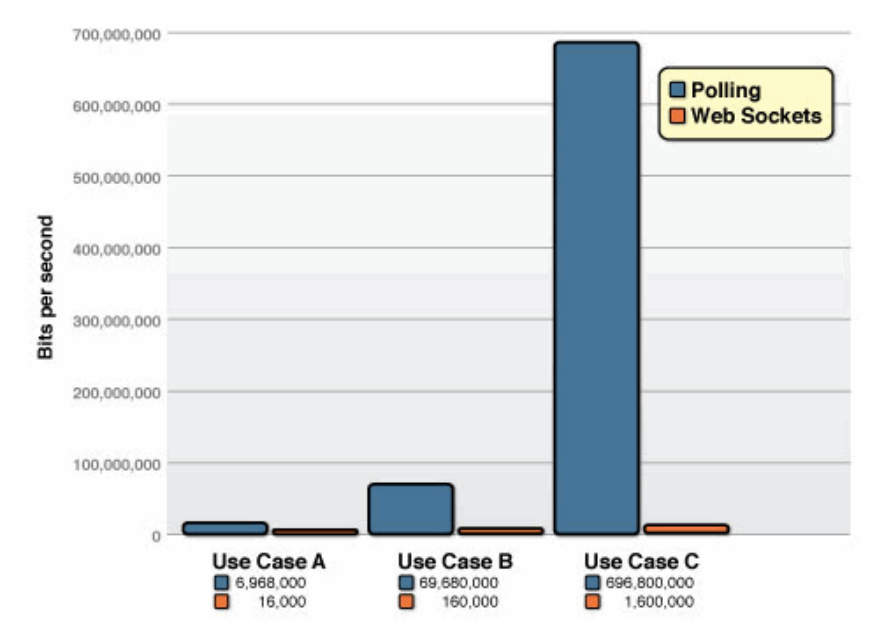


Figure 5.3: Overhead comparison between *Polling* and *WebSocket* applications [36].

Time Communication[5] (RTC) function in the browser, which is based on the on-the-wire protocol that can run over TCP and also over other protocols like the *User Datagram Protocol*[6] (UDP) or the *Stream Control Transmission Protocol*[7] (SCTP). Furthermore, *WebRTC* establishes a Peer Connection, which means that browsers can communicate with each other in a direct way. To understand the general work flow and benefits of *WebRTC* it is important to know more about other standard network standards and technologies first. The following Sections give an overview about the most important facts of these technologies in relation to *WebRTC*.

5.5.1 Transport Layer Protocols

Networking, as it is known today, is based on the *Open System Interconnect*[8] (OSI) model, which divides the tasks of network communication into seven different groups of functions, also called layers [27]. One of these layers is the so called *Transport Layer* which controls the reliable and timely transmission of data between two network nodes through flow control, segmentation and desegmentation, and error control [7]. For this, various protocols are used, often UDP and TCP, but also others like SCTP. Each of them has its own benefits and drawbacks.

[5] https://webrtc.org/
[6] https://tools.ietf.org/html/rfc768
[7] https://tools.ietf.org/html/rfc4960
[8] https://searchnetworking.techtarget.com/definition/OSI

User Datagram Protocol

UDP is a connectionless protocol, which means that there is no predetermined setup procedure for two communicating UPD processes before sending messages, which makes it faster than other protocols since there is less overhead [15, 19]. However, it is also more unreliable since there is no delivery and duplicate protection guaranteed [23]. Therefore, UPD should only be used for time critical multimedia applications where the loss of some data is acceptable like in video calls with *WebRTC*. For applications in which the stream of data should be delivered more reliably, TCP is recommended.

Transmission Control Protocol

TCP guarantees, in contrast to UDP, that the data stream will arrive intact, without any information lost, duplicated, or out of order [24]. To achieve reliability, each TCP packet is given a sequence number with which the correct order and missing or duplicated data can be detected. Furthermore, TCP has to recover from data that is damaged, lost, duplicated, or delivered out of order by the Internet communication system. Therefore, the receiving TCP must send back a positive acknowledge (ACK) in a specific amount of time, otherwise the data is sent again. Damaged data will be discarded and can be detected by a checksum [22].

All these error detection mechanisms increase the latency, which is a major drawback of using TCP for real time applications such as public space games. However, TCP is still used for most network related tasks, since it is important that the data is received in the originally intended and complete form.

Stream Control Transmission Protocol

SCTP can be seen as a hybrid of the two previously mentioned protocols as it is connection oriented, similar to TCP, and message oriented, similar to UDP. Furthermore, the reliability and ordering mechanism are optional. It is up to the final implementation if one of these mechanisms is necessary [29]. For example, in real time applications, such as public space games, an unreliable solution is preferable since the error detection functionalities increase the latency. The main reason of mentioning this rather unusually used protocol, is its use for the data channel of *WebRTC*.

5.5.2 Network Address Translation

The initial reason of implementing the *Network Address Translation*[9] (NAT) was the prevention of *Internet Protocol*[10] (IP) address exhaustion, since the NAT basically translates unique public IP addresses to private reusable ones. Furthermore, this is also useful for companies when they change their Internet Service Provider (ISP) as only the external IP address changes and the internal structure remains the same [5, 9]. There are different types of NAT in terms of how the mapping between public and private IP addresses is managed. The most noteworthy type in relation to the overall topic of *WebRTC* is the symmetric NAT, as it causes the most problems.

[9] https://tools.ietf.org/html/rfc2663
[10] https://tools.ietf.org/html/rfc791

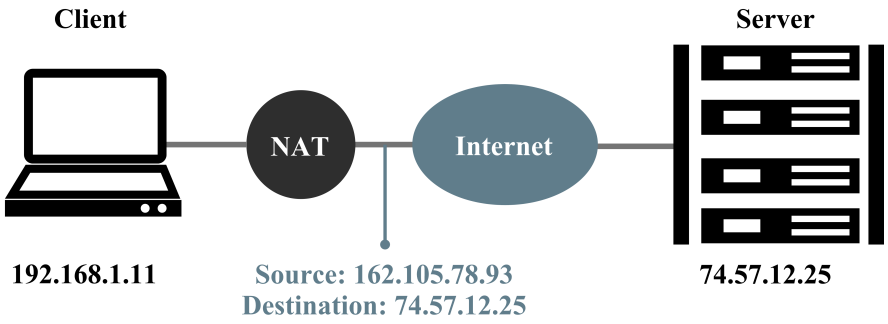


Figure 5.4: Basic concept of NAT.

For a better understanding a typical workflow of the NAT, as shown in Figure 5.4, will be explained first. The general scenario describes a laptop in a local network, which sends a request for a website to a accordingly responding server. First, the client sends the request with the private address and port. NAT uses this information, saves it in a corresponding table, and translates it into the public IP address with a specific port. Then, the server can use the source information of the received request as the destination of the response. The NAT then receives the response, looks up the corresponding information in the table, and translates the address from the public to the private one.

This is especially problematic when it comes to peer-to-peer connections. When both clients want to communicate directly with each other, they have to know the IP address where they can reach each other. The problem is that each of them only knows their local address, which is unreachable from external networks and so they have to find out the public IP and Port, which is generated by the NAT. This can be done with a STUN server which is discussed in the next Section.

Another problem appears with the use of the previously mentioned symmetric NAT. Normally the NAT uses the same mapping for all connections of a client, but the symmetric NAT creates a unique mapping for each connection. Thus, even with the use of a STUN server, it is not possible to detect the reachable address, as the connection to the *Session Traversal Utilities for NAT*[11] (STUN) server gets mapped differently than the connection between two clients. This makes a peer-to-peer connection impossible and so a *Traversal Using Relays around NAT*[12] (TURN) server is necessary [30].

5.5.3 Workflow of WebRTC

The general setup and workflow of *WebRTC* is more complex than the previously described communication technologies. The overall system contains more necessary endpoints and due to problematic mechanisms, like the NAT, fall back solutions have to be provided. The following describes each necessary step to exchange data between two clients, which problems can appear and how they can be solved.

[11] https://tools.ietf.org/html/rfc5389
[12] https://tools.ietf.org/html/rfc5766

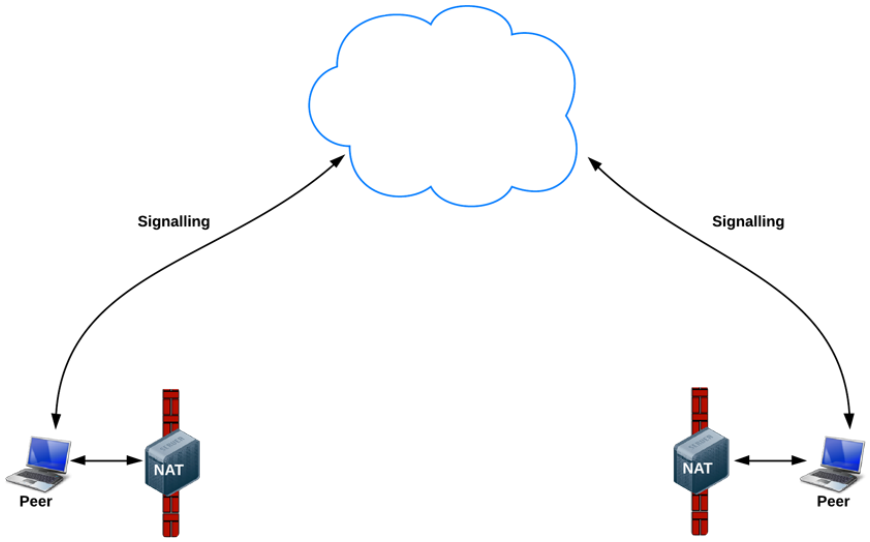


Figure 5.5: Two peers connecting and exchanging information via a signaling server [35].

Connect Users

First of all, the users have to be connected in some way. A simple solution is both visiting a specific website, which provides them a way to connect to a shared signaling server with a unique session.

An example scenario would be that the first person, who visits the webpage creates an unique token. This token can then be added to a URL and sent to the second user. As soon as both opened the unique link, the users are connected as shown in Figure 5.5.

Start Signals

The next step is that both users exchange information about their *WebRTC* setup. This can be done by the previously mentioned signaling server via *WebSockets* or other similar communication systems. How the exchange is done eventually is not specified by the *WebRTC* standards, which is a benefit on the one hand as it leaves possibilities open for innovation and evolution, but, on the other hand, it often confuses people who are new to learn about *webRTC* development.

Find Candidates

Then, both peers have to exchange information about how they can reach each other, so they have to find out their own IP address and port. Often clients use a router, including the NAT, to connect to the Internet. This can cause problems since each client obtains a externally unreachable private IP address. To find out the public IP address, which is important for the other client to establish the connection, the use of a STUN server, as shown in Figure 5.6, can be helpful. The basic functionality of the STUN server is rather simple as it just lets the browser know if it is behind a NAT and returns the mapped address and port.

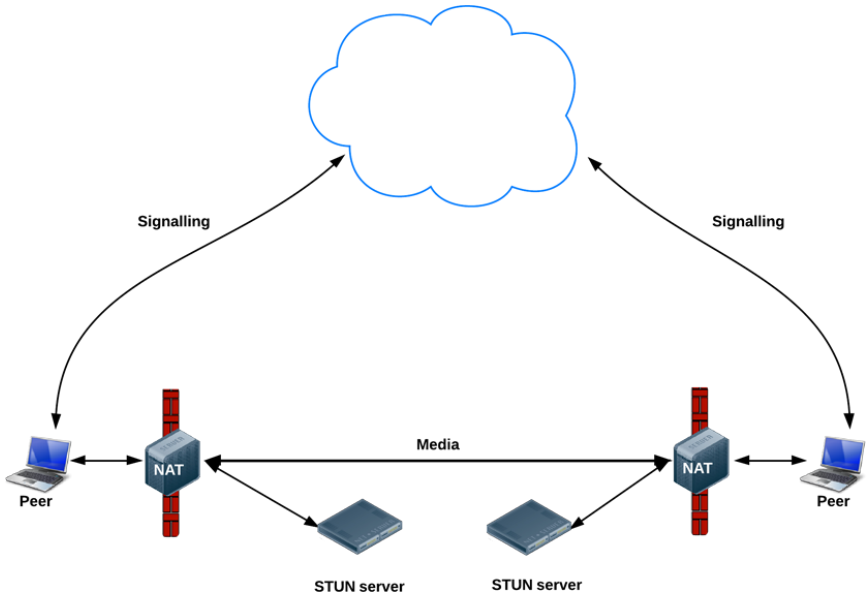


Figure 5.6: Two peers using STUN servers to identify their public address and port [35].

This can still be problematic if symmetric NAT is used as described in Section 5.5.2 since the port of the connection between the browser and the STUN server can differ from the resulting connection between the two browsers. A solution to this problem provides a TURN server as shown in Figure 5.7. If the STUN server cannot find the correct mapped address and port, the media gets relayed via the TURN server, which, on the one hand, leads to a non-peer-to-peer connection, but, on the other hand, makes at least a data exchange possible [14, 17].

A framework which handles exactly this process of connecting two peers is called *Interactive Connectivity Establishment*[13] (ICE). ICE tries to establish a direct connection between the browser via UDP. If STUN fails, it will try to connect via TCP, first with HTTP and then with HTTPS. And lastly, if the direct connection fails overall, it will establish the connection via the TURN server [31].

Negotiate Media Sessions

Before the browsers can start to exchange data, it is important that they agree on the type and format of each media like codec, resolution, bitrate and more. For the exchange of this information the so called Session Description Protocol (SDP) is used.

Start RTCPeerConnection Streams

When the negotiation about how the media will be presented is done, they can finally exchange data. To control the communication they can then either use the already used signaling server solution or use the data channel of *WebRTC* itself [14, 17].

[13] https://tools.ietf.org/html/rfc5245

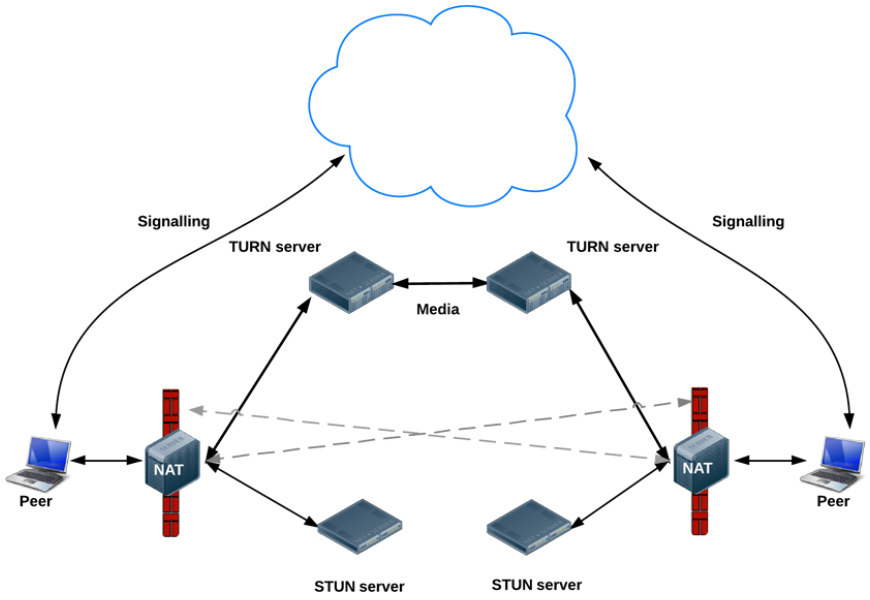


Figure 5.7: Two peers using TURN servers to relay media as fall back solution [35].

After all the research about the technologies which can be used for the communication between client and server have been done, a decision about the most fitting approach for the final project had to be made. *Polling* and *Long Polling* were not considered to be tested directly from the beginning of the implementation as both approaches were not efficient for the use case of an interactive system in the author's opinion, due to the high amount of messages between server and client because of the request and response mechanism, which gets even worse due to the theoretically unlimited number of players. The initial idea was to first use *WebSockets* and then further compare them with an implementation of *WebRTC*. Nevertheless, due to the need of a whole server/client structure including a playable game prototype, there was not enough time left to also use *WebRTC*. Furthermore, early play tests with the *WebSocket* approach showed satisfying results in terms of playability, even with multiple players. Therefore the author decided to focus on *WebSockets* and to evaluate them in terms of latency, scalability and network robustness. How they have been used in the final prototype and how they have been evaluated will be shown in the following chapters. It is still noteworthy that the author recommends *WebRTC* for future projects in relation to Public Space Games since first of all developers could focus more on other input and output modalities like camera/display and microphone/speakers and further the overall latency would be reduced, due to UDP, which could be especially interesting for streaming sensor data such as the gyroscope data.

Chapter 6

Web-Based Public Space Game

The use of smart devices as second screens and extensions of interactive modalities enhances game setups in various ways, however it also entails completely new aspects which have to be considered as shown in the previous chapter. Existing projects already benefit from this technique, but still they are either made for private spaces or require the installation of additional software like applications, whereby especially the second aspect is problematic since it is an additional barrier for people joining the game.

This makes the approach of a web-based public space game in combination with smart devices rather unique, but therefore also contains all the design challenges and requirements of smart devices, web technologies and public space games. The following sections will show how such an unique setup can be implemented and, furthermore, the main requirements of this example implementation will be described.

6.1 Characteristics

The basic concept of the overall setup was defined right from the beginning: Implementing a game prototype for an urban area which can be played with multiple players by connecting a personal smart device with the system via a web technology whereby the final gameplay and used web technology can be chosen by the author. Still, it was necessary to define specific requirements to provide a satisfying user experience.

6.1.1 Requirements

Due to the unique setup it was important to first think about the necessary requirements to create an enjoyable user experience. Since the overall setup consists of several elements, each of them had to be considered and, as the integration of smart devices into public space games is rather new, a special focus was given to the mobile controllers and their connection.

Instant Drop-In and Drop-Out

There are several barriers for joining public space games as shown in Chapter 3. Therefore, the mechanism to connect and to disconnect from the game should be simplified as much as possible. Instead of the need of additional software, like the installation of

an to the user unknown application, a web-based approach was preferred, since most recent smart devices already have a mobile browser preinstalled and so the connection can be simply done by entering a provided URL. To make this even easier it would be preferable to provide a QR-Code to the user, as people who already used this technology before know immediately what they have to do and, furthermore, the scanning is more comfortable than entering a complete link. Disconnections should be handled in a similar simplified manner and should not be dependent on an explicit action of the user as smart devices can be easily interrupted by incoming calls or network loss and therefore be disconnected from the system.

Efficient Network Performance

As mentioned in Chapter 5 there exist several techniques to connect a server and a client. It was important to choose one which performs efficiently in various aspects. Firstly, the system should be scalable which means that it is as independent as possible from the number of players. Secondly, the network has to be robust and therefore connections and disconnections have to be handled correctly. So if a new player joins the game a new character has to be created in the game. If the player then gets interrupted, for example, by an incoming call or network loss, or disconnects in an unintended way, like simply walking to far away from the system, the character has to be either completely removed from the game or disabled until the player reconnects.The former approach is preferable since the player will not loose his ingame progress, like points for the highscore. When the interruption is over or the user wants to join the game again, the reconnection should be as simple as possible and the game should identify the user as a previous player. Lastly, and most importantly, the latency of each message between the client and server should be as low as possible since the playability of the overall system relies on this aspect.

Several Input Modalities

To detect the constraints of mobile web browsers in a useful manner, several different input modalities were defined in the beginning of the project. First of all, buttons should be supported as they are often used in game controllers for simple interactions, like letting the character jump. Furthermore, swipe gestures should be possible as they are rather often used on smart devices. They could be used for an virtual joystick or to swipe in-game elements from the controller into the game. And lastly, the use of any kind of sensor data, like gyroscope or accelerometer, should be implemented.

To provide a larger potential player base it is important that the core game mechanics, like the movement of a character for example, do not only rely on explicit input modalities which require specific hardware features as they are may not supported by all devices. Therefore, special kinds of interactions should be only provided as additional functionality.

User Identification

New players in public space games are basically anonymous to the system. On the one hand, it is a benefit as people can not directly be associated with their virtual character

by the audience, which prevents social inhibition in case of wrong interactions of the user. On the other hand, this is a drawback as it prevents personal information and settings. Therefore, it is necessary to identify the user in some way to provide more gaming features like a competitive gameplay via a highscore for example. Furthermore, this also facilitates the identification of the own virtual character and to simplify it even more, the personal information should also be shown on the mobile smart devices.

6.2 System Details

To set up a web-based public space game, three different main components have to be implemented. First of all, a server has to be implemented to make a communication between multiple endpoints possible. Next, a game prototype has to be developed to provide some kind of interactive media to people. And lastly, the extension via smart devices has to be made and therefore mobile clients have to be implemented to let the users interact with the system. This section describes the overall setup in general and each of the components in detail.

6.2.1 Tools and Design Decisions

There exist several tools, technologies and programming languages when it comes to the implementation of the unique setup of a web-based public space game. Still, it is necessary to choose the correct ones which fit the best to the defined system requirements.

Server

For the implementation of the server *Node.js*[1] was used, which is a server-side framework, useful for building highly scalable and fast applications. Furthermore, it uses a nonblocking event-driven architecture and hence, most of the operations are handled in an asynchronous way, which means that events are called and in turn trigger a callback function. However, the main reason why this framework was used is the fact that it is built on Google Chrome's V8 JavaScript Engine and therefore uses JavaScript as programming language, which was initially planned to be used for each of the three components mentioned above [12].

Mobile Client

The web content of the smart devices is implemented in Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript as they are the foundational programming languages to create websites.

HTML describes the basic structure of a web page and is built out of elements with included content, which are wrapped in tags, a definition of elements for the browser, therefore it is a markup language.

CSS describes how the elements have to be displayed in the browser. It can control the visibility, color, font, size, position, layout and even animations of HTML elements.

[1] https://nodejs.org/en/

Styling rules can be applied in three different ways and therefore also have a prioritization. They can be defined directly inline within an element's attribute, in an internal style sheet and in an external style sheet whereby they are listed from highest to lowest prioritization.

JavaScript provides interaction for web pages like clicking on a button for example. It has the major benefit compared to the previous mentioned languages above that it can control the HTML and CSS of the website. Elements can be added or removed and, furthermore, also be changed in the style, however it is required [18].

Network Connection

As described in Chapter 5 there are several technologies to establish a network connection between server and clients. *Polling* has high latency due to the constant request and response model and often sends empty packages back as there is no new information available. *Long Polling* is an improvement of this technique, but still does not work efficient enough for real time applications. Furthermore, both of these technologies have a high amount of unwanted header data. *WebSockets* have the benefit of removing this unnecessary overhead and additionally establish a connection which does not get closed after a sent message and also allows the server to send messages without a required client-side request before. Still they are based on TCP, a reliable protocol, and therefore packages may have to be sent again which introduces higher latency. This can be improved by the use of *WebRTC*, which offers different channels. A channel for web based communication via video and audio which is based on UDP and therefore faster as package loss will not be corrected and a data channel which is based on SCTP where the reliability can be chosen in the implementation depending on the final use case. In the end, *WebSockets* were chosen, as the initial idea was to implement *WebSockets* and then compare them with *WebRTC*, but that was unfortunately impossible due to the amount of the overall system implementation and the remaining time. However, *WebSockets* still showed satisfying results for real time applications in early play tests and therefore are perhaps a fitting technology for the prototype of this thesis.

Game

Since the focus of the project was not the implementation of the game, the decision was made to use a game engine, which is suited for the creation of a prototype. Unity is a cross-platform game engine, which means it offers an editor for creating 2D, 3D, VR, and AR games which can be deployed on various platforms like PCs, consoles, the web, smart devices, home entertainment systems, embedded systems, or head-mounted displays [39]. Unity was initially chosen due to its support of JavaScript, but the latest version unfortunately removed the functionality to create JavaScript files since they started a deprecation process due to the small amount of the overall use of their so called *UnityScripts* [32] and so the game was implemented in C# in the end.

6.2.2 Gameplay

To test the used web technology, a small game had to be implemented to provide users some kind of interaction modality. The prototype is a 2D platformer where people can

Figure 6.1: Final Unity game prototype.

join the game by either entering the IP address at the top right corner in the game into their mobile browser or by scanning the QR code below as shown in Figure 6.1. The users will then be redirected to the HTML file where they can choose their preferred controller interface and mechanism. As soon as at least two players are connected, the in-game timer starts. The game can either be built as standalone to play directly on a computer or in *WebGL* where it can also be loaded by external PCs.

Character

Each character has a random color and a specific number or a chosen name at the top of his head so that players can identify themselves in the game. The name can be changed at any time by putting the smartphone into portrait mode and entering the new name. As soon as the submit button or the return button of the keyboard is pressed, the name will be changed in the game. When nothing is entered a predefined default ascending number will be used. Furthermore, the name or number is also shown on the controller so that it is even easier to identify the own character.

Movement

The movement of the characters was kept quite simple, as shown in Figure 6.2, so that it is easy for the players to press the correct button without looking on the controller while playing the game. Each player can move to the left and right. This can be done in three different ways. One controller contains only buttons, so as long as the left or right button is pressed, the character moves in the chosen direction. As soon as the button is released again, the character stops. The next modality is to use a virtual joystick as shown in Figure 6.2. To move the character, the user just has to put a finger on the

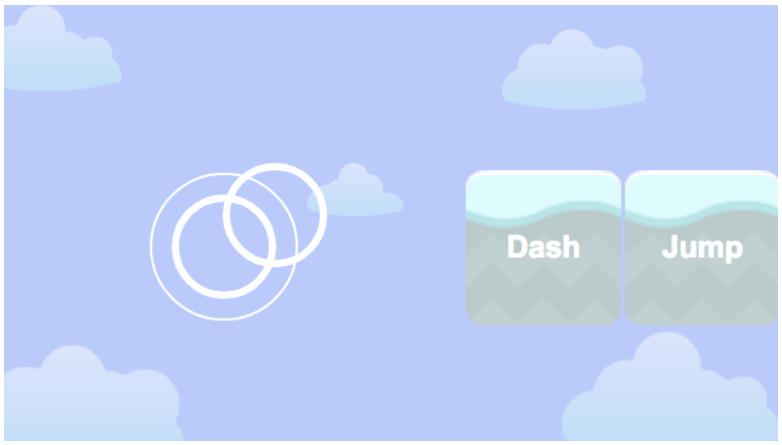


Figure 6.2: User interface of the mobile controller with a virtual joystick.

left half of the display, move it in the desired direction and keep the finger pressed. To change the direction, the finger just has to be moved without releasing the joystick first. As soon as the player takes the finger off the display, the virtual character stops moving. The last option is to use the gyroscope of the smart device for the movement. When the user tilts the smart devices further than a specific angle around the x-axis (pitch/beta at Figure 4.2), the character starts to move. To stop the virtual avatar, the user has to hold the device in a horizontal position to keep the angular degree between the specified positive and negative angle.

Furthermore, it is possible to jump or double jump by pressing the specific button twice. And lastly, there is also a dash button to kick other players of the platforms. Every action button can also be used in combination with one of the movements. When a player dashes into another one, a particle system and a sound will be triggered to get a visual and auditory feedback.

Goal

The goal of the game is to score as many points as possible during each game round. If a player dashes into another one, who in turn falls off all platforms, the hitting player receives 50 points. On the other hand if a character falls down on his own fault he loses 25 points. At the top left corner in the game the names of the top five players inclusively their current points are always visible.

Debugging

Specific keys of the desktop keyboard have been provided with functionality to simplify the debugging of the game. With the arrow keys all characters in the game can be moved. Pressing the *S* key spawns new players on the platforms. The *D* key can be used to let the characters dash forward and with *Space* the virtual avatars jump. Furthermore, it is possible to reset the game round by either pressing *R* to spawn all the players at different spawn points or *Z* to spawn them at 0,0,0.

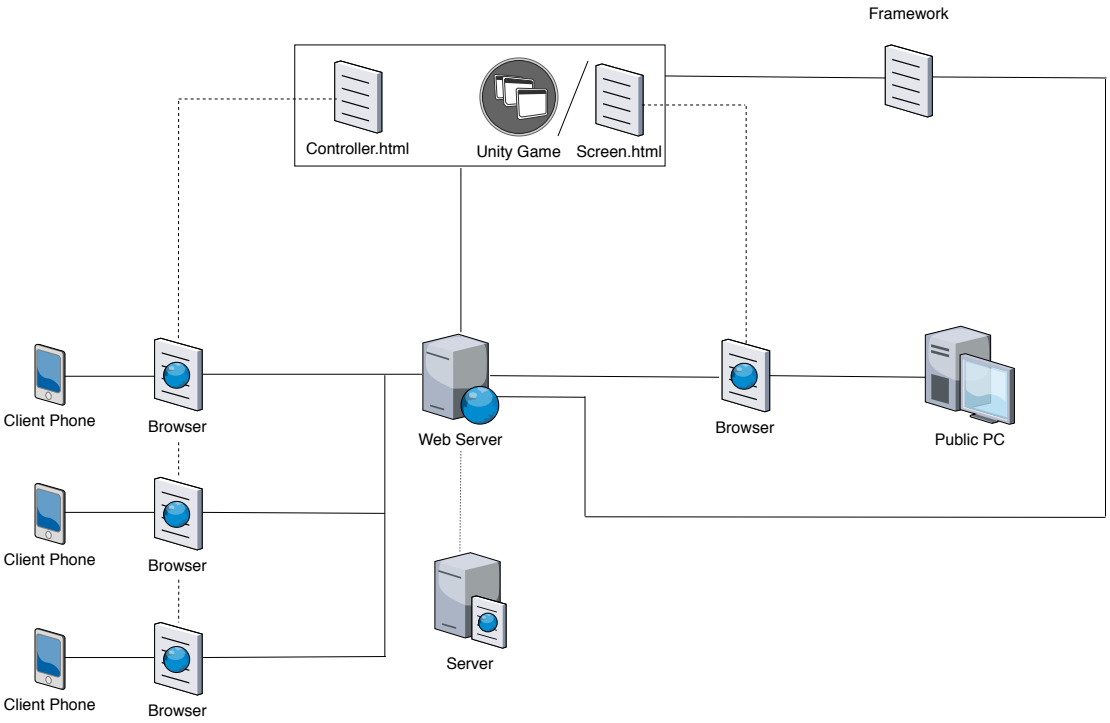


Figure 6.3: The network architecture of the whole project.

6.2.3 Network Architecture

This Section describes the basic network structure of the overall system as shown in Figure 6.3. Each component will be described in detail and the connection to each other will be shown.

The web server in the center of the figure connects all the mobile clients with the final game prototype and is therefore the core element of the overall system. Also the server at the bottom center of the picture is noteworthy, which can be used to make the web server externally accessible. For example a SFTP server with a forwarded port to the previous uploaded web server can be used. This has the major benefit that joining the game is even more simplified for the user since both do not have to be in the same network to establish a connection. However, it also has the drawback that a higher latency will be introduced as the messages have to be delivered via the Internet instead of the local network.

To enable interactivity for the overall system, three different kind of files are mandatory on the web server. Firstly, controller files have to be provided. The mobile clients can connect to the web server via a specific IP address and port and then get redirected to the chosen controller files mentioned before. Secondly, a game prototype has to be available, either as a standalone build which can be played directly on the personal computer or an externally reachable web file, like a *WebGL*[2] build or a *JavaScript*[3]

[2] https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
[3] https://techterms.com/definition/javascript

game. Lastly, the communication between the server and the clients has to be handled and therefore a framework was implemented, which will be described in detail in the next section as it was the initial goal of this thesis.

6.3 Implementation

To set up the system the implementation of a server, the clients, desktop and mobile, and most importantly a framework for the communication between them was necessary. This section will explain how each of these components have been implemented, describes them in detail and shows the aspects, which have to be considered to connect and start the overall system.

6.3.1 Server

As mentioned before, the server is implemented in *Node.js* and uses the *WebSocket* library *ws*[4]. Therefore, both of them have to be installed first. Then the web server has to be started.

In the beginning, a HTTP server starts as the *WebSocket* Protocol is perhaps not supported by the used browser. Then, an upgrade to a *WebSocket* server will be done if possible. Next, the server listens on a specific IP address and port and the clients can start to connect. Each connection will be saved on the server to directly redirect messages from one endpoint to the other instead of always sending broadcasts to all available clients as this produces unnecessary and unwanted network traffic.

6.3.2 Clients

The clients have to be differentiated in two categories: the mobile clients, as the controllers, and the desktop client, as the game prototype. The former type can again be subcategorized between buttons, virtual joystick and gyroscope controller whereby all of them have the same file structure.

The HTML file describes the basic structure, the CSS file the style and the JavaScript file the functionality of the controller. Noteworthy is the additional use of the libraries *virtualjoystick.js*[5] and *gyronorm.js*[6]. The design of the controller is based on the style of the game prototype and the user interface was implemented in a simple manner to give the user the possibility to play the game without looking at the display of the smart device. For this reason the buttons have also been extended in their actual size without changing the size of the image, so players do not have to be that precise and can also touch the screen above or beneath the represented button to execute the desired action. Furthermore, a text input field was implemented in the portrait mode of the controller to let the user choose a unique and personal name. The maximum length was set to ten characters to avoid an overlapping of the name and other elements in the game. If the player enters an empty name, an automatically generated ascending number will be used. The chosen name or player number is then represented on the controller, to simplify the

[4] https://www.npmjs.com/package/ws
[5] https://github.com/jeromeetienne/virtualjoystick.js
[6] https://github.com/dorukeker/gyronorm.js

identification of the personal character in the game. Additionally, the default operation of all elements, except of the input field, is prevented to avoid unwanted scrolling and zooms via double clicks.

The second type is implemented in *Unity* and therefore uses various C# scripts for the game mechanics. It is also important to mention the use of a package from the *Unity* asset store and the use of a library. The *Simple Web Sockets for Unity WebGL* package [38] is used for creating a single *WebSocket* in the game, to connect to the server and listen to the redirected incoming messages. Noteworthy is that the package is missing string receive handling, which caused problems with the *WebGL* build as no messages were received. To fix this problem, the *WebSocket.jslib* had to be extended with the functionality to receive strings. The *ZXing*[7] library was used to generate a QR-Code out of the IP address and port of the server, which can easily be scanned by the users to join the game.

Beside the implementations, again there are several aspects, which have to be considered to establish the connection to the server and start the game. It is important that the clients are on the same network as the web server, if the web server is used in the local network and not in combination with an externally reachable server. The IP address and port for the mobile clients can be changed in the *controller.js* files. The smartphones can then be connected to the server by entering the specific IP address in the browser. To connect the game prototype with the server, the IP address has to be set correctly in the *Global* script of the Unity project. The QR code changes automatically since it is generated out of the mentioned IP address.

6.3.3 Framework

The framework is implemented in JavaScript and can be used to handle the connection between multiple mobile clients and a specific endpoint (server/client/game). It is independent of the rest of the project, to make it usable for any other kind of game. The framework establishes a connection to a server via a *WebSocket* and has methods and listeners to cover the following aspects:

- messaging,
- directions,
- jump,
- dash,
- username,
- identification,
- vector2D,
- vector3D.

Unfortunately, the prevention of scrolling and zooming via double tap can not be prevented in the framework as this has to be done directly in the JavaScript file of the actual website. Still, it is noteworthy that the method `event.preventDefault()` has to be called in the *touchmove* and *touchend* event listeners of the actual document to prevent

[7] https://github.com/zxing/zxing

these default operations. Furthermore, it is very important to set the parameter option *passive* to false for both listeners.

To use the framework, first a `Connector` object has to be created with the IP address and port of the server:

```
var connector = new Connector("10.59.0.74", "3000");
```

This automatically generates an ID which can be used on the server to differentiate between the clients. Due to time constraints, randomly generated numbers, which are saved in the cookies are used in the framework as an ID. This solution could be improved by using a database for example. Then a *WebSocket* gets started in the background, which establishes a connection to the server. If the controller looses the connection at any time, the framework tries to reconnect to the server every 1500 milliseconds to provide a robust network for the user.

Methods

The methods offer a way to send messages to the other endpoints. They cover multiple aspects, which are mentioned above. The following code snippet shows an example of how to start and stop the movement of a character when a button is pressed and released:

```
document.getElementById("leftButton").addEventListener('touchstart', function (event
    ) {
   connector.moveLeft();
});

document.getElementById("leftButton").addEventListener('touchend', function (event)
    {
   connector.stopLeft();
});
```

It is also possible to send unique commands by either easily extending the framework or by using the following method:

```
connector.sendCommand({
   message: "doStuff",
   playerID: connector.getId()
 })
```

Listeners

The listeners can be used to handle incoming commands. The following code shows some methods which can be used to listen to the *WebSocket* events:

```
connector.onOpen = function (event) {
  // ...
}

connector.onMessage = function (event) {
  console.log("Received: " + event.data);
}
```

The `onMessage` method is called for every incoming message to also cover unique commands. To stick to the code snippet of the methods, the following example shows how to react to the incoming movement commands:

```
connector.onMoveLeft = function(event) {
  // start character movement
}

connector.onStopLeft = function (event) {
  // stop character movement
}
```

A covering of the messages without the use of the framework and its listeners by using for example an other programming language like C# could look like this:

```
if (jsonResponse.message == "stopLeft") {
    // stop character movement
}
```

Testing

The author of this thesis recommends the *Google Chrome Developer Tools* for testing the framework and the overall system since connections with different devices and operating systems can be used without the need of the real physical device.

This chapter has shown the necessary components for a web-based public space game in detail and also described the requirements which have been defined especially for this thesis. Furthermore it gave a deeper insight about the implementation of the components and how the requirements have been fulfilled. Still it is uncertain if *WebSockets* perform efficient enough for a real time application with multiple users. Therefore the next section will evaluate the chosen web technology and additionally the unique overall setup in general.

Chapter 7

Evaluation

The basic concept of enhancing a public space game with smart devices via the network is rather new. Furthermore, there exist several technologies for establishing the connection between the mobile devices and the final game prototype. Although the chosen technology of *WebSockets* is suitable for real time applications in general, there is still no proof if they are suitable for public space games in terms of latency, scalability and network robustness. Therefore, an evaluation about the performance of *WebSocket* and furthermore about the game itself, to detect its positive and negative aspects, was made. It is also worth mentioning that the evaluation was made in two different setups. First, an evaluation was made by using *WebSockets* in different networks via the Internet, but due to the high variety of latencies, a second evaluation was made in a single network where the connection to the Internet was blocked and therefore only the local network was used, which made the evaluation more dependent of the used technology instead of the actual Internet connection.

7.1 Game Experience Survey

The final game prototype of this thesis is a competitive platformer. It is important to keep this in mind since this only represents one of many different game genres. Therefore, it was necessary to create a game survey about the prototype itself to differentiate between the overall user experience of the game and of the used network technology.

7.1.1 Method

As mentioned before, the evaluation was split up in two different setups, but still used methods that shared some of their properties. Before the users started to play the game or even connected to the system, they were encouraged to listen to a detailed explanation of the game and its mechanics itself. The explanation was written down beforehand, to keep it the same for everyone and reads as follows:

> The game is a prototype and a multiplayer platformer. To connect to the game it is necessary to be in the same WiFi as the server and then either the IP address on the right top corner of the game has to be entered in any mobile browser or the QR-Code in the game has to be scanned. After the

> connection with the game it is possible to choose between three different controllers. The character can be controlled with buttons, a virtual joystick or with the gyroscope of the smart device. The character can be moved to the left and right, can make a double jump, which gets reset when the character lands on any platform again, and can dash into other players. A game round takes one minute. As soon as two players are connected, the game round starts. The goal of the game is to get as many points as possible. When a player falls of the platforms, he loses twenty-five points. If a player dashes into another player and the other player falls in turn off the platforms, the hitting player gets fifty points.

After the explanation the participants of the study joined the network of the server and started to play the game. Every time a new player joined, a few test rounds were carried out to give the user the opportunity to learn the mechanics of the game. As soon as all players understood how to control their character the real game round started. Each session was at least about two to three game rounds long, whereas each of the game rounds took one minute. Afterwards, the players were asked to fill out a game experience survey, which was created by the author himself, with the help of *Google Forms*[1]. Besides general information, like age and gender, it covers the most important aspects of this evaluation. Except of specific answers about problems, critics, opinions and the general information, all the question could be answered on a six-point Lickert scale ranging from "Disagree" to "Agree". The players had to answer the following statements and questions, whereby the last three questions were only valid for people where the amount of players changed during the game rounds:

- The game felt always responsive to my input.
- I could always orientate myself in the game.
- I could easily identify my character in the game.
- I did not see/feel any kind of latency during the game.
- The amount of input modalities was sufficient.
- I had problems to establish the connection.
- I would try out such a game in a public setup.
- The game is playable in my opinion in terms of latency, high amount of players and satisfying handling of connections and disconnections.
- Did you have any disconnections during the game?
- Did you have any interruptions during the game? (Lock button pressed, incoming call, home button pressed, ...)
- Was there a difference between three and five players in terms of latency?
- Was there a difference between five and eight players in terms of latency?
- Was there a difference between three and eight players in terms of latency?

It is important to note that the people were not informed to explicitly pay attention to the change of latency in terms of the increasing amount of players beforehand. Therefore, the results of the last three questions can be questionable.

[1] https://docs.google.com/forms/

7.1.2 Results

As mentioned before, the setup varied during the evaluation. First, playtests were made in different networks with connection to the Internet and later a second test was made in a single offline local network. If any major differences appeared due to the different setups, it will be mentioned directly in the description of the specific result, but still this has to be considered for the results of the game survey. Furthermore, it has to be noted that for most of the diagrams the results of the first playtests will be used, as it was tested with a higher amount of users. However, people could write down detailed opinions, critics and suggestions at the end of the survey, which represent the positive and negative aspects of the overall setup.

Enjoyable Gameplay and Easy Access

Almost every person who tested the game enjoyed playing it and liked the overall idea of the concept. People immediately started to think about exciting use case scenario. For example, one person had the idea to use such a system for interactive advertisements of companies in the cinema before the beginning of the actual movie, whereby all other participants of this test session agreed that this would be a very good use case scenario. Elements and graphics could just be replaced by more brand related content and people could then easily join the game and actively play until the movie starts instead of just watching the advertisements.

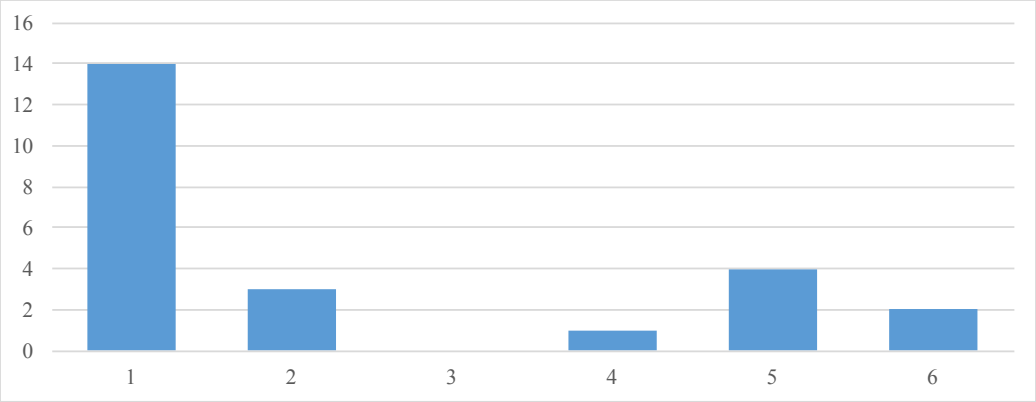


Figure 7.1: Results of the first playtests show that the majority of people had no problems to establish the connection to the game with the total range from “Disagree” (1) to “Agree” (6).

An aspect that was mentioned often, was the easy access to the game. As soon as people wanted to join the game, they intuitively asked, which application or software they have to download and install as they are used to this procedure from other products. Therefore, people were often surprised by hearing that there is no need for any additional software. Furthermore, it is worth noting that people were not annoyed by the fact that they have to connect same network of the server first to join the game since the overall access was easier than they expected. This is interesting because it could be seen as a barrier to connect to the game and could be avoided by using an externally reachable

server, which would in turn lead to higher latency as all the network traffic would be sent via the Internet instead of the local network. Furthermore, the positive responses about the easy access are very important since the simplified joining process of the game by entering a specific IP address in the browser or scanning a QR-Code was a requirement of the thesis project and therefore has been approved as a good concept to use as shown in Figure 7.1. Only some of the participants had problems with the connection due to the behavior of installed QR-Code readers as they opened the URL in a smaller in-app browser instead of opening it in the native browser of the smart device.

Varying Latency

One of the most negative results of the game experience survey was the high latency in some of the used networks. For the first playtests this was caused by the general Internet performance of the used network as all the messages between the server and the clients were sent via the Internet although private IP addresses were used. For the second approach the Internet was blocked by using entering a non-existing proxy server in the network configurations of the used laptop for the web server. This led to an improvement of the network performance between the clients and the server and thus people rather perceived any kind of latency with an increasing amount of players. Still people mentioned different reasons for the cause of the latency, which makes the evaluation of the used web technology with these results rather questionable and therefore a more precise technical evaluation of *WebSockets* will be done in the next section.

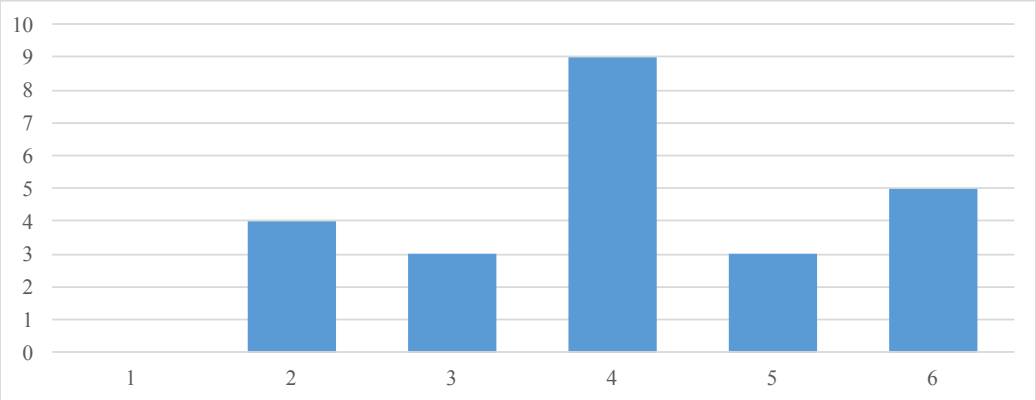


Figure 7.2: Results of the first playtests show that the game is playable in terms of latency, a high amount of players and the handling of connections and disconnections in most peoples' opinion with the total range from "Disagree" (1) to "Agree" (6).

Nevertheless, the results of the evaluation and the behavior of the people in relation to latency are still interesting. The users who tested the game were annoyed very quickly and frustrated whenever any kind of latency appeared. The players felt immediately treated in an unfair manner as their actions on the controller did not lead to their desired behavior in the game. Therefore, this information is very important to consider in the beginnings of the development of such a web-based public space game. Even though the game is still playable in terms of latency, a high amount of players and the handling of connections and disconnections, according to the majority of the participants

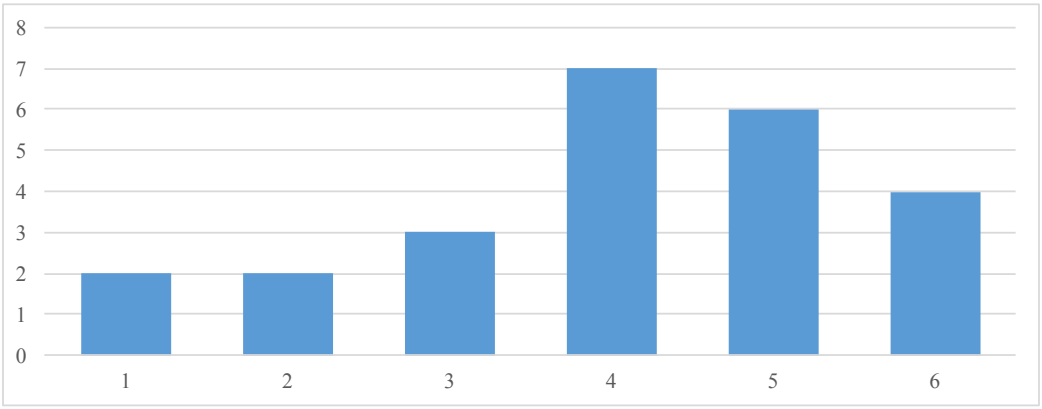


Figure 7.3: Results of the first playtests show that not all players could easily identify their character in the game with the total range from "Disagree" (1) to "Agree" (6).

as shown in Figure 7.2, it is crucial to choose performant technologies and design a fitting gameplay to prevent such frustrations right from the beginning.

Difficult Character Identification

Every character in the game has a random unique color and at least an ascending number or a chosen name, represented above the digital avatar and additionally on the screen of the smart device. This was the initial idea of identifying the own character easily in the game. Unfortunately it was still difficult for some players to identify themselves as described in the detailed critics, even though the majority of the players agreed on the easy identification of the character as shown in Figure 7.3. As the color selection is random, often the a similar one was chosen and therefore not useful for the identification as this color existed twice in the game. This caused confusion by the players and hence the own character was hard to find. Furthermore, an increasing number of participants naturally led to an increasing number of virtual in-game avatars. This was problematic since people either could not see their own character as it was hidden by overlapping another player or were overwhelmed by the multiple characters in general.

Some of the participants suggested to let the players choose their color on the smart device on their own before the actual game round starts. Another idea was to use more contrasting colors instead of the random generated ones. And lastly, the use of a short highlighting of a spawning character, like a lighting circle around the avatar, was proposed. All of these ideas could be used together, nevertheless, the identification of the character is a very important aspect for designing a web-based public space game, as these kind of games should theoretically support a high amount of players and therefore the users have to be able to identify their own avatar at any time as otherwise frustration and annoyance will be caused by the players.

Controller Characteristics

The testers of the game basically liked the simple control mechanics and the various input modalities of buttons, a virtual joystick and the gyroscope were sufficient for them

as shown in Figure 7.4. On the one hand, people mentioned that they prefer simple controllers as they think any additional features would make playing the game more difficult while, on the other hand, people recommended further features like choosing your own color, as mentioned before, adding vibration as feedback for the player or even taking a picture of the user for the face of the virtual character.

However, there were also aspects mentioned, which caused problems to play the game in general. Firstly, people suggested to rearrange the layout of the user interface, to place the jump and dash buttons vertically next to each other instead of horizontally.

Secondly, some participants of the first playtests had problems with zooming and scrolling in the browser. Even though this was disabled for many devices, it was still possible with some of them. This was very problematic since it destroyed the whole user experience as it was impossible to use the controller in the correct manner. For example, when people kept one button pressed for moving the character, additionally pressed the action button for jumping or dashing and slightly moved one of their fingers on the display, the device interpreted this gesture as pinch to zoom. Then, dependent of the finger movement direction, either only a small part of the whole controller was visible for the user or the layout changed to the tab view of the browser and therefore it was impossible to use the controller correctly. Furthermore, when scrolling was enabled, it was mainly inconvenient, due to the fact that the user interface moved constantly and therefore it was more difficult to press the correct buttons and secondly, the virtual joystick was hard to use as the swipe gestures were attenuated by the moving interface in the background. This problem was fixed for the second playtests since it was a major problem for the general gameplay.

Lastly, some users complained about the appearance of the action on the display as they often accidentally triggered actions unintendedly, like pressing the back button of the browser or opening the keyboard due a touch on the address bar. This could only be prevented in some browsers by using the full screen of the display for the content, but unfortunately not all browsers support this functionality.

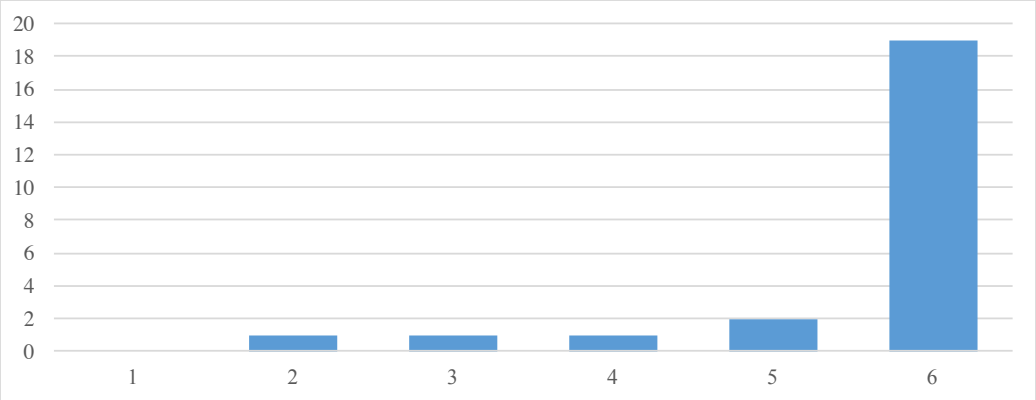


Figure 7.4: Results of the first playtests show that the amount of input modalities was sufficient for almost every person with the total range from “Disagree” (1) to “Agree” (6).

Miscellaneous

Beside the already mentioned results, players suggested further ideas for the overall setup, which are not exactly suitable for any of the previous mentioned topics, but still worth noting. One suggestion was to fix of the dash function as it was still buggy and therefore behaved differently. Another idea was to add items to make the game more interesting, which could also solve the negative impression of another participant that there is not enough incentive for moving the character in the game. Lastly, the duplication of the game screen on the smart devices display with an almost transparent user interface, like in *Playerunknown's Battlegrounds*[2] for smartphones, was proposed, which is a good idea, but unsuitable for anything else than first-person games due to the small display.

7.2 WebSocket Evaluation

One of the defined requirements for the project of this thesis was the use of a web technology for the communication between the server and clients with a suitable network performance in terms of latency, scalability and network robustness. Among other technologies, the decision was made to use *WebSockets* as they are already used for other real time applications and further showed satisfying results in early playtests. Yet, it is questionable if their performance is sufficient enough for the unique use in a web-based public space game with multiple players and therefore a fitting evaluating had to be made.

7.2.1 Method

The three most important, previously mentioned aspects for the used web technology were evaluated in different ways. The performance in terms of latency and scalability was evaluated in combination with each other since the latency is dependent of the amount of players. The correct handling of connections and disconnections during the game was evaluated in the previous mentioned game experience survey and has been additionally confirmed by its results. A simplified connection was provided by an URL and an additional QR-Code. The disconnections were handled in two ways. First, the gameplay of the prototype was rather independent of the players connection as the game rounds were rather short and disconnected character just stood still in the game world, which could be negatively used to easily get points, but did not directly influence the players user experience. Therefore, it was not problematic if a player disconnected from the game. When players got disconnected by interruptions like incoming calls, they were automatically reconnected as soon as the was phone call was finished. This has been achieved by continuously trying to establish a *WebSocket* connection as soon as the previous one got closed.

However, the latency in relation to the amount of players had yet to be evaluated and therefore the following method was used. As soon as a player pressed the jump button, a message with the corresponding command and an additional timestamp of the mobile client was sent to the server and in turn to the game. Afterwards, when the

[2] https://playbattlegrounds.com/main.pu

message was received, the game immediately sent back a response containing the client's and furthermore the games' timestamp itself. As soon as the smart device receives the package, an approximation of the latency will be calculated. This is done by subtracting the timestamp of the initially sent message from the current time of the client and dividing the result by two to get the latency of one way from the client to the game respectively vice versa. The timestamp of the game was then further used to calculate the difference between the client's and the game's time as they are both on different devices and therefore not synchronized. Even though the difference was included in the overall process of evaluating the latency and an approximation of the synchronized game time was calculated, it was not directly used as the calculation of the latency's approximation was independent of it. Then again a message was sent to the game. Afterwards the included information was separated, enhanced with game related data and then exported into a CSV (Comma-Separated Values) file that was created beforehand. An example of a final single entry of the evaluation list can be seen in Table 7.1.

Game ID	275
Date	08.06.2018
Player ID	792640402
Player Name	Toby
Sync Game Time	14:59:51,161
Actual Game Time	14:59:51,994
Client Timestamp Sent	14:59:52,658
Game Timestamp	14:59:51,928
Client Timestamp Received	14:59:52,733
Latency	37,5
Operating System	Android
Operating System Version	6.0.1
Browser	Chrome
Browser Version	53.0.2785.146
User Agent	Mozilla
App Version	5.0
Platform	Linux armv8l
Vendor	Google Inc.
Number of players	3

Table 7.1: Single example entry of the created list for the evaluation of *WebSockets*.

The first value of Table 7.1 is an ongoing number, which increases whenever the game is started in *Unity* to identify the individual play test sessions. Next, the actual date of the current day is entered, followed by the ID of the player who pressed the jump button. In addition, the name of the player is saved to simplify the identification of the

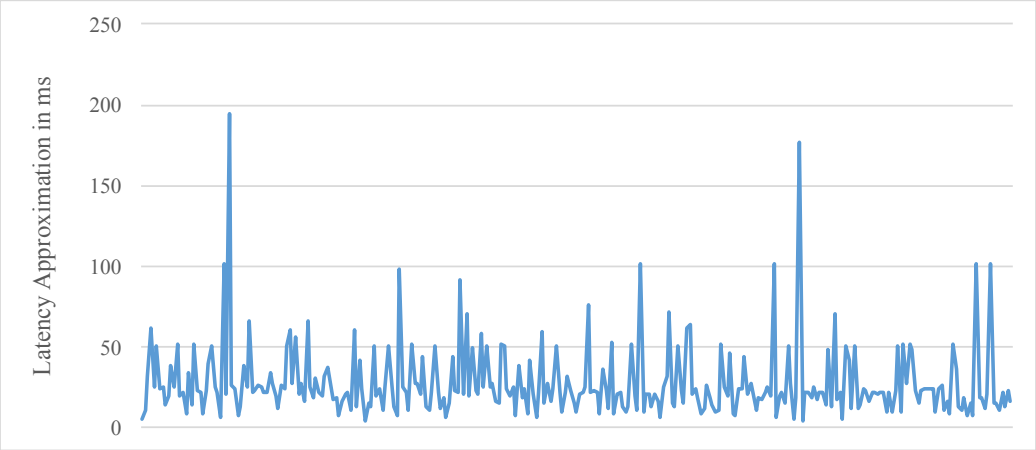


Figure 7.5: Results of the second playtests show that *WebSockets* perform efficient in terms of latency with a small amount of players.

used device if any unintended behaviors are detected. Then, various timestamps are included. Firstly, the unused synchronized game time. Secondly, the actual game time, directly accessed from the game's device. And lastly, the client and game times of the previously mentioned calculations. Afterwards, the most important value, containing the approximation of the latency, was saved, followed by device related information, which actually was not used for the evaluation itself, but still saved in case of unusual behaviors. At the end, the number of players was saved to be able to see the relation between the latency and the number of participants.

7.2.2 Results

The method mentioned above was used for every single playtest with various amount of players and different users. As the setup of the first playtests was dependent of the Internet connection, only the results of the second playtests were used for the technical evaluation since they are more related to the used web technology itself. The relevant data was acquired by letting multiple people play the game prototype, whereby the number of players was increased several times after some game rounds. This resulted in a high amount of data sets. Since the initial idea was to measure the latency of three, five and eight or more players, the results will be split up in these three categories for a better visualization with the only difference of using the results of ten simultaneous players instead of the initially planned eight, since there were more people available than expected. Furthermore, an amount of 300 consecutive entries of each category will be used for a better comparison of the single results.

As shown in Figure 7.5, *WebSocket* perform efficiently when only three players are connected with the system and play the game. Beside single higher latencies the arithmetic average is rather small with 27,80 milliseconds. This was also approved by the players. They could easily perform every action they desired, which were executed in time in the game.

After several game rounds the number of players was increased to five to see if

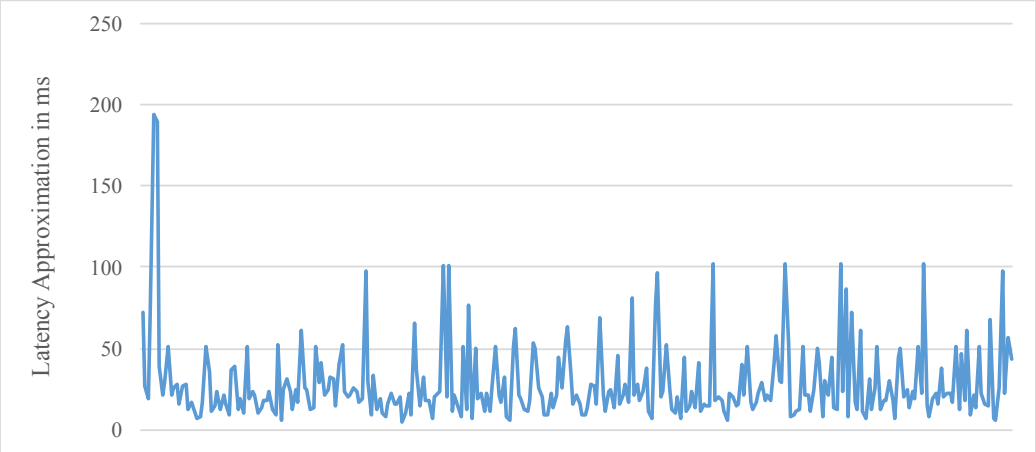


Figure 7.6: After increasing the amount of players slightly up to five, the results of the second playtests show that the performance of *WebSockets* hardly changed.

the latency changes. Even though the amount of users were higher before, the results of the latency were still similar to the previous ones as shown in Figure 7.6. Only the arithmetic average slightly increased up to 28,46 milliseconds. Yet, no outstanding latencies appeared except for some outliers. Also the players did not complain about any disturbing network problems and just continued playing against each other for multiple game rounds.

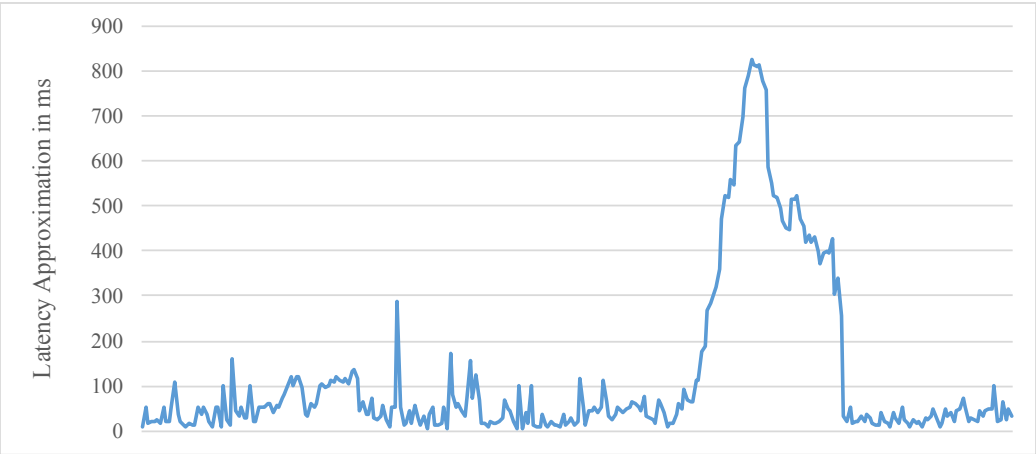


Figure 7.7: Results of the second playtests show that the latency increases with a higher amount of players by using *WebSockets* for the network communication.

As soon as eight participants played the game simultaneously, users noticed delays between pressing their desired action buttons and the actual executions in the game for the first time. Figure 7.7 shows that with ten players already rather high latencies appeared, which can interrupt the flow of the game. Also the arithmetic average was 119,88 milliseconds, four times higher than before. Even though this is an enormous

difference, it is still overall a rather low latency. Therefore, the gameplay was basically possible, but just interrupted from time to time.

This evaluation shows that *WebSockets* have a high potential for the use in public space games. Even though the last results showed a rather low network performance for higher amount of players, *WebSockets* should still be considered to be used for multiple participants as the high latency can also be also caused by other aspects like the network itself or the use of a single *WebSocket* in the game prototype instead of parallel ones for each character. A more detailed analysis of the results and a conclusion about the thesis in general will be given in the following chapter.

Chapter 8

Concluding Debate

8.1 Result Analysis

The evaluation of this thesis delivered valuable information for setting up a web-based public space game in combination with smart devices. Future projects with this unique setup can use the results for an improvement of their implementations and therefore, the most important outcome will be analyzed in detail.

Simplified Opt-In

In general, the connection via the web has been shown as a very good solution to overcome the barrier of people not joining the game because of inconvenient connection mechanisms like installing an application. Most of the people who tested the game prototype had no problems with establishing a connection to the system and were further even surprised about the simple way to join the setup. As this aspect does not directly rely on *WebSockets*, any kind of web communication can benefit from this information. The additional use of a QR-Code can be questionable. On the one hand, sometimes the scanning led to additional problems due to external applications, but, on the other hand, many people preferred this approach over entering a whole URL and therefore, QR-Codes should be either considered as a modality to join the game itself or at least as a second possibility like in the tested prototype.

Identification Improvements

The identification of the own character has shown as a very important topic. Simply showing the users name above the virtual avatar and additionally on the display of the smart device was not sufficient for some participants. Furthermore, a specific level of uniqueness should be provided for each character as users were often confused by similar colors of different figures. People who had problems to identify themselves in the game could not play the game as it was intended and were often just kicked from the platforms by other players since they just either just jumped or did not move at all while looking for their character.

Robust Network Connection

In addition to the easy access, *WebSockets* also proved to provide a very robust network connection. People did not have any kind of disconnections by the used technology itself and in any case of interruptions, like closing and reopening the browser, the users immediately reconnected to the game without even noticing the disconnection itself. This could have been done by fitting provided callback functions as soon as the connection gets closed.

Sufficient Network Performance

WebSockets showed satisfying results for smaller amounts of participants where people could play the game in an undisturbed manner. Even though the latency increased with a higher amount of players, it has to be mentioned that the used game prototype is based on a highly interactive competitive gameplay. Therefore, it is possible that *WebSockets* also provide a sufficient performance for higher amount of players by using a slower gameplay like in round-based games.

8.2 Further Prospects

The implemented setup of a web-based public space game extended with smart device turned out to be a valuable game concept. Yet, there are various possible improvements for the overall system. Firstly, individual player feedback does not exist in the current implementation. Only the name of the own character is shown on the display, but since not all browsers support a fitting use of audio and also haptic, like vibration, it is not yet possible to provide this kind of feedback, which can be very helpful for the players and therefore, should be definitely kept in mind for future projects. Secondly, the performance results of *WebSockets* could potentially have been influenced by various aspects, like the gameplay itself or the usage of a single socket in the game instead of multiple parallel ones. Therefore, further investigations have to be made about the performance of this particular web technology in public space games. Still the evaluation of this thesis already provides a first useful outcome. Thirdly, beside *WebSockets* there also exist other web technologies to provide a connection between smart devices and a game in urban areas. *WebRTC* is recommended by the author as, on the one hand, it provides very low latency due to the underlying transport protocol and the direct connection between two endpoints and, on the other hand, the media channels can be used to implement completely features, like displaying a video of the player directly in the game. Furthermore, the focus on interesting components, which have not been used in the project of the thesis at all, like the camera and microphone, would increase. Fourthly, in addition to a better identification of the character itself, a better identification for the clients in general should be used. The saving of a random generated number in the cookies is a rather simple and inefficient solution as cookies for example are only saved in a single browser and therefore, if a player changes to another one, he will be identified as a new participant even though he was already connected before. A preferable solution would be the use of an external database where for example complete user profiles could be created. Another benefit of this approach would also be the persistent storage of other game related information like the highscore of the players.

Fifthly, the use of an externally reachable server should be tested. Even though a local network provides lower latency and the participants who tested the game did not complain about connecting to the same WiFi of the server, this extension should still be considered as it is a further barrier of joining the game and further peoples' opinion about connecting to an unknown network perhaps changes in a real public setup. Lastly, and less importantly, but yet worth to note, is the change of game related aspects. As the platformer is a prototype, there still exist several problems, which have to be fixed, like the often similar colors of the character, the differently behaving dashing and the non-existing removal of disconnected characters. Furthermore, the gameplay could be made more interesting by additional levels or in-game features like items or could be completely changed to another game genre.

8.3 Conclusion

Various topics have been covered in this thesis. Public space games in general, functionalities and constraints of smart devices and lastly web technologies for the communication between two endpoints. Each of them has its own benefits and challenges. Public space games are an interesting idea, as they can be used, on the hand, for the entertainment of people in urban areas, but, on the other hand, theoretically also for the advertisement of companies by using brand specific media as mentioned by the people who tested the game prototype. However, they also have several challenges, which have to be accomplished, often by the use of additional hard- and software. A solution with great potential to overcome these challenges is the additional use of smart devices due to their accessibility, since many people have at least one with them most of the time, and further because they provide many different functionalities. Several input modalities are offered by touch gestures, device sensors, the microphone and the camera. Furthermore, it is also possible to give the players individual feedback by various output components like the speaker or the display itself. Another very important functionality is the availability of an interface to any network. Therefore, any web technology can be used to connect the user with the final game, whereby it is necessary to find one with a sufficient performance in terms of latency, scalability and network robustness. First results have shown that *WebSockets* work performant under these conditions, although there are still further investigations necessary.

However, even though this setup has many requirements and challenges, people showed great interest in it and therefore, the extension of public space games with smart devices via the web has been approved as a good concept and should be used for future similar projects.

Appendix A

CD-ROM/DVD Contents

A.1 Project

- /src: Project Source Code
- /images: Images of the game and controller

A.2 Thesis

- /Thesis_Temper.pdf: Thesis (this document)
- /Thesis_Temper.zip: LaTeXproject of the thesis

A.3 Evaluation

- /Game Experience Survey/Results_1.csv: Results of the game experience survey of the first playtests
- /Game Experience Survey/Results_2_charts.xlsx: Charts of the game experience survey of the second playtests
- /Game Experience Survey/Results_2.csv: Results of the game experience survey of the second playtests
- /Game Experience Survey/Survey_1.pdf: Game experience survey of the first playtests
- /Game Experience Survey/Survey_2.pdf: Game experience survey of the second playtests
- /WebSocket Evaluation/Results.csv: Original results of the *WebSocket* performance of the second playtests
- /WebSocket Evaluation/Results_charts.xlsx: Results inclusively charts of the *WebSocket* performance of the second playtests

References

Literature

[1] Muhammad Anshari and Yabit Alas. "Smartphones habits, necessities, and big data challenges". *The Journal of High Technology Management Research* 26.2 (2015), pp. 177–185 (cit. on p. 7).

[2] Rafael Ballagas et al. "The Smart Phone: A Ubiquitous Input Device". *IEEE Pervasive Computing* 5.1 (2006), pp. 70–77 (cit. on p. 13).

[3] Regina Bernhaupt. *Evaluating User Experience in Games. Concepts and Methods*. London: Springer Publishing Company, 2010 (cit. on p. 15).

[4] Harry Brignull and Yvonne Rogers. "Enticing people to interact with large public displays in public spaces". In: *In Proceedings of the IFIP International Conference on Human-Computer Interaction*. (Zurich). Amsterdam: IOS Press, 2003, pp. 17–24 (cit. on p. 12).

[5] Christopher Carthern et al. *Cisco Networks. Engineers' Handbook of Routing, Switching, and Security with IOS, NX-OS, and ASA*. New York: Apress, 2015 (cit. on p. 27).

[6] Darryl Charles et al. "Player-Centred Game Design: Player Modelling and Adaptive Digital Games". In: *Proceedings of DiGRA 2005 Conference: Changing Views - Worlds in Play*. 2005 (cit. on p. 15).

[7] Deji Chen, Mark Nixon, and Aloysius Mok. *WirelessHART. Real-Time Mesh Network for Industrial Automation*. New York: Springer US, 2010 (cit. on p. 26).

[8] Victor Cheung et al. "Overcoming Interaction Barriers in Large Public Displays Using Personal Devices". In: *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. (Dresden). New York: ACM, 2014, pp. 375–380 (cit. on pp. 12–14).

[9] Kjeld B. Egevang and Pyda Srisuresh. *Traditional IP Network Address Translator (Traditional NAT)*. Tech. rep. 3022. Fremont: Association Management Solutions, Jan. 2001. URL: https://www.rfc-editor.org/rfc/rfc3022.txt (cit. on p. 27).

[10] Katharina Emmerich, Stefan Liszio, and Maic Masuch. "Defining Second Screen Gaming: Exploration of New Design Patterns". In: *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*. (Funchal). New York: ACM, 2014, 7:1–7:8 (cit. on p. 15).

[11] Andreas Friedl. “Integration of Mobile Devices into a Floor-Based Game to Increase Player Dynamics”. MA thesis. Hagenberg: University of Applied Sciences Upper Austria, Sept. 2015 (cit. on pp. 3–9, 12–14).

[12] Cory Gackenheimer. *Node.Js Recipes. A Problem-Solution Approach*. New York: Apress, 2013 (cit. on p. 34).

[13] Saul Greenberg et al. “Dark Patterns in Proxemic Interactions: A Critical Perspective”. In: *Proceedings of the 2014 Conference on Designing Interactive Systems*. (Vancouver). New York: ACM, 2014, pp. 523–532 (cit. on p. 14).

[14] Alan B. Johnston and Daniel C. Burnett. *WebRTC. APIs and RTCWEB Protocols of the HTML5 Real-time Web*. St. Louis: Digital Codex LLC, 2013 (cit. on p. 30).

[15] John Kristoff. *The Trouble with UDP Scanning*. Mar. 2002. URL: https://condor.depaul.edu/jkristof/papers/udpscanning.ps (cit. on p. 27).

[16] David Machaj, Christopher Andrews, and Chris North. “Co-located Many-Player Gaming on Large High-Resolution Displays”. In: *2009 International Conference on Computational Science and Engineering*. (Vancouver). Vancouver: IEEE, 2009, pp. 697–704 (cit. on pp. 3, 5).

[17] Rob Manson. *Getting Started with WebRTC. Explore WebRTC for Real-time Peer-to-peer Communication*. Birmingham: Packt Publishing, Limited, 2013 (cit. on p. 30).

[18] Sarah Martin. *The Definitive Guide to Squarespace. Learn to Deliver Custom, Professional Web Experiences for Yourself and Your Clients*. New York: Apress, 2017 (cit. on p. 35).

[19] Christian Nagel et al. *Pro .NET 1.1 Network Programming*. New York: Apress, 2004 (cit. on p. 27).

[20] Den Odell. *Pro JavaScript Development. Coding, Capabilities, and Tooling*. New York: Apress, 2014 (cit. on p. 16).

[21] Kenton O’Hara, Maxine Glancy, and Simon Robertshaw. “Understanding Collective Play in an Urban Screen Game”. In: *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*. (San Diego). New York: ACM, 2008, pp. 67–76 (cit. on pp. 3–5, 12, 14).

[22] Jon Postel. *Transmission Control Protocol*. Tech. rep. 7. Fremont: Association Management Solutions, Sept. 1981. URL: http://www.rfc-editor.org/rfc/rfc793.txt (cit. on p. 27).

[23] Jon Postel. *User Datagram Protocol*. Tech. rep. 6. RFC Editor, Aug. 1980. URL: http://www.rfc-editor.org/rfc/rfc768.txt (cit. on p. 27).

[24] Brandon Rhodes and John Goerzen. *Foundations of Python Network Programming*. 3rd. New York: Apress, 2014 (cit. on p. 27).

[25] Peter Saint-Andre et al. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. Tech. rep. 6202. Fremont: Association Management Solutions, Apr. 2011. URL: http://www.rfc-editor.org/rfc/rfc6202.txt (cit. on pp. 21–23).

[26] Aditya R. Shankar. *Pro HTML5 Games. Learn to Build your Own Games using HTML5 and JavaScript*. New York: Apress, 2017 (cit. on p. 20).

[27] Phani R. Tadimety. *OSPF. A Network Routing Protocol*. New York: Apress, 2015 (cit. on p. 26).

[28] Vanessa Wang, Frank Salim, and Peter Moskovits. *The Definitive Guide to HTML5 WebSocket*. New York: Apress, 2013 (cit. on pp. 21–23).

Online sources

[29] Bloggeek.me. *SCTP*. 2014. URL: https://webrtcglossary.com/sctp/ (cit. on p. 27).

[30] D.A.R.Y.L. *Symmetric NAT and Its Problems*. 2011. URL: http://www.think-like-a-computer.com/2011/09/19/symmetric-nat/ (cit. on p. 28).

[31] Sam Dutton. *Getting Started with WebRTC*. 2012. URL: https://www.html5rocks.com/en/tutorials/webrtc/basics/ (cit. on p. 30).

[32] Richard Fine. *UnityScript's long ride off into the sunset*. 2017. URL: https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/ (cit. on p. 35).

[33] Google. *Super Sync Sports*. 2013. URL: https://experiments.withgoogle.com/collection/chrome (cit. on p. 10).

[34] Joe Hanson. *What is HTTP Long Polling?* 2014. URL: https://www.pubnub.com/blog/2014-12-01-http-long-polling/ (cit. on p. 22).

[35] Sam Dutton Justin Uberti. *WebRTC - Plugin-free realtime communication*. 2013. URL: http://io13webrtc.appspot.com/ (cit. on pp. 29–31).

[36] Peter Lubbers and Kaazing Corporation Frank Greco. *WebSocket*. 2018. URL: https://www.websocket.org/quantum.html (cit. on pp. 23–26).

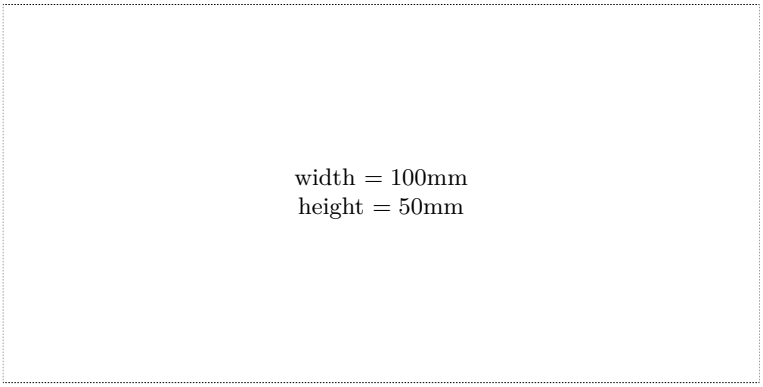
[37] Sony. *That's You!* 2017. URL: https://www.playstation.com/en-us/games/thats-you-ps4/ (cit. on p. 12).

[38] Unity Technologies. *Simple Web Sockets for Unity WebGL*. 2015. URL: https://assetstore.unity.com/packages/essentials/tutorial-projects/simple-web-sockets-for-unity-webgl-38367 (cit. on p. 40).

[39] Unity Technologies. *Unity*. 2018. URL: https://unity3d.com/public-relations (cit. on p. 35).

Check Final Print Size

— Check final print size! —



— Remove this page after printing! —