Universitatea Tehnică a Moldovei

Real-Time Communication and Interaction in Modern Web

Comunicarea și interacțiunea în timp real în contextul tehnologiilor web

Student: **Istratii Andrei**

Conducător: **Zarea Ivan**

Chişinău 2016

Ministerul Educației al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea de Calculatoare, Informatică și Microelectronică
Filiera Anglofonă „Computer Science”

Admis la susținere
Prof. univ., dr. hab. Viorel Bostan,
Director Filieră Anglofonă

_____

„__”_____ 2016

Real-Time Communication and Interaction in Modern Web

Proiect de licență

Student: A. Istratii (_____)
Conducător: I. Zarea (_____)
Consultanţi: G. Covdii (_____)
M. Balan (_____)
E. Gogoi (_____)

Chişinău 2016

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Filiera Anglofonă „Computer Science”

Aprob.
Prof. univ., dr. hab. Viorel Bostan,
Director Filieră Anglofonă

_____

„__”_____ 2015

CAIET DE SARCINI

pentru proiectul de licență al studentului

Istratii Andrei

(numele și prenumele studentului)

1. Tema proiectului de licență: *Comunicarea şi interacţiunea în timp real în contextul tehnologiilor web*

confirmată prin hotărârea Consiliului facultății de la „ *21* ” *octombrie 2015*

2. Termenul limită de prezentare a proiectului *31.05.2016*

3. Date inițiale pentru elaborarea proiectului *Sarcina pentru elaborarea proiectului de diplomă.*

4. Conținutul memoriului explicativ

Introducere

1. *Descrierea și analiza domeniului de studiu*
2. *Analiza aplicației din punct de vedere arhitectural*
3. *Tehnologiile disponibile și implemetarea sistemului*
4. *Analiza economică*

Concluzii

5. Conținutul părții grafice a proiectului

Diagrama use-case generală a sistemului, Interfața principală a programului.

6. Lista consultanţilor:

Consultant	Capitol	Confirmarea realizării activităţii	
		Semnătura consultantului (data)	Semnătura studentului (data)
G.Covdii	Argumentarea economică		
E. Gogoi	Controlul calităţii		
M. Balan	Controlul calităţii		
V. Bostan	Standarde tehnologice		

7. Data înmânării caietului de sarcini *01.09.2015*

Conducător _____

semnătura

Sarcina a fost luată pentru a fi executată

de către studentul _____ *01.09.2015*

semnătura, data

PLAN CALENDARISTIC

Nr. crt.	Denumirea etapelor de proiectare	Termenul de realizare a etapelor	Nota
1	*Elaborarea sarcinii, primirea datelor pentru sarcină*	*01.09.15– 30.09.15*	*10%*
2	*Studierea literaturii de domeniu*	*01.10.15– 30.11.15*	*20%*
3	*Alegerea şi pregătirea de lucru a softului*	*01.12.15 – 25.12.15*	*20%*
4	*Realizarea programului*	*16.01.16 – 30.04.16*	*25%*
5	*Descrierea programului, diagramele UML*	*01.05.16 – 15.05.16*	*10%*
6	*Testarea aplicaţiei*	*16.05.16– 28.05.16*	*10%*
7	*Finisarea proiectului*	*29.05.16– 31.05.16*	*5%*

Student *Istratii Andrei ( )*

Conducător de proiect *Zarea Ivan ( )*

Abstract

This thesis represents an attempt to use modern web technology in the field of real-time communication in order to enhance and encourage communication and interaction between humans. It is to be done by the development of a web-based multiplayer game that uses mobile phones as wireless controllers in order to emulate the experience of a gaming console.

The text of the thesis consists of four chapters that encompass individual steps of the project's development process. Chapter 1 performs an in-depth analysis of the domain, provides a brief overview of how did human interaction with computers evolve, and identifies the motivation for the project. In the same chapter are depicted the potential technologies (WebSockets and WebRTC) that can be used for solving the problem in question. The last sections of the first chapter contain an analysis of applications that are currently present on the market and follow similar ideas.

Chapter 2 is focused on the architectural aspects of the system. It includes the results of the modeling process in form of UML diagrams with descriptions of the system and individual parts of it, from different perspectives. Design decisions are also specified in this chapter, along with the motivations that stood behind them.

The third chapter provides a detailed insight of the project implementation. The three major subsections describe respective parts of the system, specifically: the main game, the controller component and the code that realizes the communications between these two. Code excerpts with commentaries try to emphasize the techniques and tools that were used during development.

The last chapter performs an economic overview of the project with the attempt to estimate its market potential. It includes calculations of various types of expenses that result in the total product cost. This is used to compute the economic indicators of the project and the retail price of the individual unit of the product if it had to be sold on the market.

At the end of the thesis, the conclusion presents the outcomes of the project and the general results gain during the work with the technologies used in the development process.

Rezumat

Această teză reprezintă o încercare de a utiliza tehnologiile web moderne și anume acelea din domeniul comunicării în timp real, cu scopul de a consolida și de a încuraja comunicarea și interacțiunea între oameni. Scopul va fi realizat prin dezvoltarea unui joc multiplayer bazat pe web ce utilizează telefoanele mobile în calitate de dispozitiv de control, emulînd experiența unei console de jocuri.

Textul tezei este format din patru capitole, care cuprind etapele individuale a procesului de dezvoltare a proiectului. Capitolul 1 efectuează o analiză detaliată a domeniului, oferă un scurt istoric a modului în care a evoluat interacțiunea omului cu calculatorul și identifică motivația pentru proiectul tezei. În același capitol sunt prezentate potențialele tehnologii (WebSockets și WebRTC) ce pot fi utilizate pentru rezolvarea problemei în cauză. Ultimele secțiuni ale primului capitol conțin o analiză a aplicațiilor care sunt în prezent pe piață.

Capitolul 2 este axat pe aspectele arhitecturale ale sistemului. El include rezultatele procesului de modelare în formă de diagrame UML, însoțite de descrierea sistemului și a părților individuale a acestuia din diferite perspective. La fel sunt specificate deciziile de proiectare, împreună cu motivațiile care au stat în spatele lor.

Al treilea capitol oferă o perspectivă detaliată asupra implementării proiectului. Trei subsecțiuni majore descriu părțile respective ale sistemului, și anume: jocul, componenta dispozitivului de joc și codul care realizează comunicarea între acestea. La fel sunt incluse fragmente de cod cu comentarii ce accentuează tehnicile și instrumentele care au fost folosite în timpul dezvoltării.

Ultimul capitol realizează o trecere în revistă a proiectului din punct de vedere economic cu încercarea de a estima potențialul său economic. Acesta include calcule de diferite tipuri de cheltuieli care în ansamblu constituie costul total al produsului. Acesta este folosit pentru a calcula indicatorii economici ai proiectului și prețul de vânzare a unei unități de produs în cazul în care acesta ar fi introdus pe piață.

La finalul tezei sunt prezentate rezultatele generale a proiectului care au fost obținute în timpul lucrului cu tehnologiile utilizate în proces de dezvoltare.

Contents

List of Figures

Listings

Introduction

Today, people cannot imagine their lives without computers by their side, although, only seventy years ago, such a world could be the setting of a bestselling science fiction novel. Back then, it was a research field that had just planted its roots through the creation of the first ever electronic computer.

Throughout these years, computers have evolved as much as the ways of interacting with them. People transitioned from using various switches and knobs to ergonomic keyboards and touch screens as their input devices. Computers have also become more connected as with the appearance of the Internet a person could communicate and interact with a device that is located on the other side of the planet. So, the present thesis sets its focus specifically on these technologies viewed inside of a modern context.

Communication and interaction are crucial aspects of human life and technology can be used to greatly enhance them. Take, for example, team-building events in a company, that are supposed to bring the employees together for a common activity like sport or some kind of ice-breaking game. In a situation when the former might be complicated to organize and the later becomes rather silly when people get to know each other, computer games come to the rescue. The game development industry has flourished ever since computers became largely available, moreover, specialized consoles have been developed to serve the very purpose of gaming. Game consoles can be a wonderful alternative for team- building activities, but they require considerable investments and often limit the number of players that can be engaged in a game at one moment in time.

The thesis proposes a solution to the logistic problems described above. In the context of recent developments in web technology, and especially the field of real-time communication (RTC), it is possible to create a web-game based multiplayer game and web-based controllers that would communicate and emulate the experience of a game console, with the benefit of being able to run everywhere where there is a browser installed.

Real-time communication and interaction is a long-running concern of game developers around the world. As Internet speeds have increased lately, multiplayer online gaming gained unimaginable levels of popularity and developers had to come up with solutions and techniques that would make gameplay as smooth and responsive as possible in order to create the impression that all players are located in the same room and interact with a single computer through dedicated input devices. Sadly, this interest in RTC was, until recent, noticeable only in stand-alone applications and lacked development in the web context. As the available technologies are very fresh and many specification documents are still in draft states, this thesis will perform an attempt to gather and analyze the tools that are currently present on the market and will show a poof of concept game that is able to cope with the tasks outlined above.

The project proposed in the thesis is a simple web-based multiplayer arcade that employs real-time communication technologies like WebSockets and WebRTC to connect and use players' mobile phones as gaming input devices. It aims to bring an engaging and interactive experience for a group of people located in the same room without any setup procedures and without a need for specialized gaming gear like gamepads or joysticks. Various patterns and techniques are expected to be identified as a result, in order to be used in future projects of this kind.

The four chapters that follow, encompass the individual steps of the project's development process. The first chapter performs an analysis of the domain and of potential technologies that can be used for solving the problem in question. Several existing applications are presented and analyzed in the last sections.

The second chapter describes the architectural decisions and the motivations behind them. It includes the results of the modeling process in form of UML diagrams with descriptions of the system and individual parts of it from different perspectives.

The chapter three provides a detailed insight of the project implementation. The three major subsections describe respective parts of the system, specifically: the main game, the controller component and the code that realizes the communications between these two. Code excerpts with commentaries emphasize the techniques and tools that were used during development.

The last chapter performs an economic overview of the project with the attempt to estimate its market potential. It includes calculations of various types of expenses that result in the total product cost. This is used to compute the economic indicators of the project and the retail price of the individual unit of the product if it had to be sold on the market.

1 Domain Analysis

1.1 User Interfaces and Input Devices

At the very beginning of the computer era, the devices that we now know as PCs, laptops and tablets were immense pieces of machinery that occupied entire laboratories and only highly qualified individuals had access to them. These machines where developed in order to perform various numerical problems by executing digital computations. As the saying goes, a machine can only do what a man tells it to do, so engineers had to come up with different means to set tasks for these electronic beasts, that is, a need for *user interfaces* arouse. Operators used large stacks of punched cards to feed instructions and data sets to computers. These punched cards in turn where created using specialized devices that also required some knowledge in the field. Over all it was a complex process that couldn't be easily grasped by an ordinary persons. At that time, however, the majority of computer user had PhDs and were trained to perform these very tasks so the difficulty of communicating with machines wasn't really a great problem.

In time, the interest towards computers grew bigger among enthusiasts while the devices themselves where becoming smaller and smaller, eventually giving birth to the term of 'microcomputer'. Among the first microcomputers to get widespread popularity was the critically acclaimed Altair 8080 which represented a box with lights and switches on the front panel that where used to feed data and instructions into the computer and read the results back from it.

As more hobbyists took a hold of such devices people discovered many uses for them beyond that of performing various mathematical operations. They learned how to connect teletypes to computers and this way the later got an interface that is familiar to all of us today, a keyboard. For some time people interacted with computers using command lines, but with the development of computer graphics and the creation of graphical user interfaces, a need for a new kind of controller appeared, specifically the need for a pointing device. Since then, a standard personal computer included a mouse and a keyboard among it's peripheral devices.

1.1.1 Input Devices for Gaming

On the other hand, the development of input devices never stopped with the creation of the mouse and keyboard. A wealth of specialized devices was created to perform tasks that were specific to a particular use of the computer. One of the most notable fields that was moving the development of specialized peripherals was and still is the *game development industry*. During the years, various types of controllers were developed, some of them are listed below:

Gamepad is a general-purpose gaming device that is used as a controller for a wide range of game genres, from arcades and fighting games to role-playing titles and action third-person shooters;

Joystick is a specialized controller that is often used in flight simulation games, although its use can be extended far beyond gaming, for such purposes as remote control of a robot arm in a warehouse;

Steering Wheel is used to provide almost authentic driving experience in racing video games.

(a) Gamepad (b) Steering Wheel Controller (c) Joystick

Figure 1.1 – Examples of Input Devices for Gaming

1.2 Mobile User Interfaces

Very soon, technological progress made it possible to stick microcomputers virtually in any device, be it a fridge, microwave oven or a car. The most benefit out of this, however, gained the industry of mobile phones, eventually giving birth to the concept of smart-phones. Today, almost every person living in a more or less civilized region of the world has in his/her pocket a little 'brick' with the computer power thousands of times greater than the one that was used to send people to the moon. This breakthrough in portable computing facilitated the development of software for such devices and the majority of companies started to support mobile applications where it made sense.

In the context of user interface design and input handling techniques, developers found themselves in a completely different world. Nobody would carry a mouse and a keyboard in order to interact with the phone, instead, people would use the number pad plus a couple of additional buttons present on the phone. In time, another revolution took place, it was the moment when touch screens became reality. Once again, developers had to think their way through these changes and create user interfaces that would suit fingers poking into the screen rather than button clicks. With the difficulties of user interface redesign for touch devices came also a wide range of new possibilities. Now that the whole screen constituted an input device, the users could perform gestures like taps, swipes, pan motions, pinch-zoom actions. These gestures made the process of using an application a great deal more intuitive when compared to the times when every action was performed through the click of a certain button.

Besides touch screens, mobile phone vendors began to include in their devices such sensors like accelerometers and gyroscopes. Accelerometers are used to catch the moment when the device is moved and to calculate the direction and acceleration of such movement. On the software side this could be used in various ways, the simplest being to switch to the next song in the music player by jerking the phone to the right. Meanwhile, gyroscopes where designed to capture the orientation of the device at a given moment in time. This functionality is employed every time the phone is tilted over 90 degrees and triggers the switch between landscape and portrait view modes.

When it came to the industry of mobile gaming, touch screens and sensors that could capture the motion and orientation of the device were a really big deal, because they opened the possibilities for developing complex user interfaces that would simulate specialized gaming devices. The accelerometer and gyroscope would be used as the steering wheel in racing games while the touch screen would capture taps in various regions and interpret them as clicks of different buttons of a gamepad, whose simplified image would be rendered on the same screen.

1.3 Mobile Devices as Controllers for Desktop Computers

When comparing the features of a specialized gaming device like *Nintendo Wii*[1] Controller and the capabilities of a modern smart-phone, an interesting pattern of similarities can be observed. Both are able to capture device motion and position. Both respond to clicks of buttons in case of a gamepad and taps in designated areas in case of a phone. Touch screens can be used to simulate even the motion of a joystick, by capturing swipe or pan gestures. One feature that a specialized gaming device may have, that a phone falls short of, is ergonomics, but to a certain extent this can be neglected. With such a list of similarities, a logical question appears: Why not use mobile phones as gaming controllers for desktop games?

1.3.1 Usability Limitations

As it turns out, now that smart-phones run full-fledged operating systems, connecting one to a computer is not that hard of a task. Mobile operating systems like Android, Apple iOS and Windows Mobile, all support socket programming, thus a programmer can setup a communication channel over the network between a mobile phone and a laptop connected to the same Wi-Fi hotspot. They can even be located on different sides of the planet, it is sufficient to have an Internet connection to be able to exchange data between devices.

Overall it looks like a way to go approach. On the other hand, there are some problems with it that are not to be observed at first sight. In the first place, every time that a company wants to create a game with support for mobile devices as controllers, the developers will have to write custom mobile applications in addition to the main game, they will also have to devise specialized communication protocols. This is both, time and money consuming and is not economically convenient. Secondly, there is a problem on the user side which is best illustrated with a situation. Let's suppose a party or a team-building event where the host tries to entertain his guests with a multiplayer computer game. Not everyone has at home a gaming platform like Sony Playstation or Microsoft Xbox, neither does anyone have more than 2-3 PC gamepads. In this situation, the ability to use smart-phones as controllers would be a great benefit. In case the connection is implemented in the way described above, a gaming party would transform into a setup party, where guests would spend a lot of time installing mobile applications that are probably needed only for one-time use and don't hold any value by themselves without the main game. Fortunately both problems can be solved relatively simple through the same solution.

1.3.2 Web-based Workaround

One thing that all modern smart-phones have in common is a decent web browser. Today, browsers are more than just viewers of HTML documents, they are entire ecosystems and programming environments that are almost independent of the underlying operating system. The fact that every mobile phone has such an environment makes it possible to write cross-platform application served on the web that would otherwise be installed manually as a native app. Modern browsers support APIs that can interact with various parts of a mobile device like accelerometer, gyroscope and touch screen, exactly the things that are necessary to simulate a fully functional gaming device.

Having said this, one thing that could solve the problems stated in the section above may be a framework or toolkit that unifies in a single library all the APIs that are necessary to connect a mobile phone to a computer as a remote gaming controller. It would make it easier and faster to develop games with such a feature. This toolkit may also include user interface building blocks which can be combined to create custom controllers.

The purpose of this thesis is to create a simple web-based multiplayer isometric arcade called 'Snowfight'. The main game is started by accessing its web page. The first thing the players see is a lobby and a connection URL. Players join the game by navigating to the provided URL using the browsers installed on their phones. When enough players have connected and the participants decide to start the game, a button can be clicked on the main game screen. The game play concept is rather simple, players can move their characters around the game space and throw snowballs in each other using the controls rendered by the web application that works on their phones. Every player has a certain amount of hit points (HP) and if a player is hit, his HP amount decreases. When a player's HP amount reaches zero, he is eliminated from the game. The goal of the game is to eliminate all players from the adversary team.

In order to implement this project it is necessary to explore the topic of real-time communication in the web, as to be able to chose the right technology that would provide a smooth and responsive gaming experience.

1.4 Real-Time Communication over the Web

One of the main uses of the Internet is hosting and accessing web pages. When a user writes a URL in the browser's search bar and presses 'enter', the browser preforms an HTTP request to the server specified by the URL and fetches a web page. At this point in time, for most cases, the communication between the server and the client (browser) is ceased until the user clicks on another link that would restart the process.

On the other hand, modern web has greatly developed during the past few years. Internet connection speed has increased dramatically, and many websites have stopped being just static web pages and have moved towards a more dynamic model. They are not even called websites, today, all around the world there are web applications.

With the requirements for a more dynamic web came also the technical challenges of making it so. For instance, how would a user's browser receive a notification about a message that was sent to this user from another part of the globe? When such questions just appeared, developers tried

different techniques using the old tools they had at hand at that time. The concepts of HTTP Long Polling and HTTP Streaming appeared in this period. Soon, however, they were found to be causing quite a bit of issues[2] and a need for custom protocols arose.

As the web is mainly composed of clients and servers, the logical outcome was a protocol that would connect them and would permit bidirectional communication between the server and its clients, thus WebSockets were proposed. At the same time, browsers grew and their uses extended. Soon, real-time communication out- sized the context of client-server architecture and when video chat applications and multiplayer games started to conquer the web realm, engineers developed WebRTC, which solved the issue of communication in peer-to-peer setups. A brief description of both protocols as per their RFCs is presented in the sections that follow.

1.4.1 WebSockets for Client-Server Communication

Historically, creating web applications that need bidirectional communication between a client and a server (e.g., instant messaging and gaming applications) has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

As RFC6455[3] points out, this results in a variety of problems:

- The server is forced to use a number of different underlying TCP connections for each client: one for sending information to the client and a new one for each incoming message;
- The wire protocol has a high overhead, with each client-to-server message having an HTTP header;
- The client-side script is forced to maintain a mapping from the outgoing connections to the incoming connection to track replies.

A simpler solution would be to use a single TCP connection for traffic in both directions. This is what the WebSocket Protocol provides. Combined with the WebSocket API, it provides an alternative to HTTP polling for two-way communication from a web page to a remote server.

The same technique can be used for a variety of web applications: games, stock tickers, multiuser applications with simultaneous editing, user interfaces exposing server-side services in real time, etc.

The WebSocket Protocol is designed to supersede existing bidirectional communication technologies that use HTTP as a transport layer to benefit from existing infrastructure (proxies, filtering, authentication). Such technologies were implemented as trade-offs between efficiency and reliability because HTTP was not initially meant to be used for bidirectional communication. The WebSocket Protocol attempts to address the goals of existing bidirectional HTTP technologies in the context of the existing HTTP infrastructure; as such, it is designed to work over HTTP ports 80 and 443 as well as to support HTTP proxies and intermediaries, even if this implies some complexity specific to the current environment. However, the design does not limit WebSocket to HTTP, and future implementations could use a simpler handshake over a dedicated port without reinventing the entire protocol. This last point is important because the traffic patterns of interactive messaging do not closely match standard HTTP traffic and can induce unusual loads on some components.

1.4.2 WebRTC for Peer to Peer Communication

Real-time communication has been on the market for a while, however, it was implemented mainly in corporate environments as in-house solutions. It was quite expensive in terms of effort to include such technology in existing projects especially in the context of the web. In recent years this started to change with the appearance of various frameworks and APIs.

WebRTC[4] is a free, open project that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities via simple APIs. The WebRTC components have been optimized to best serve this purpose. Due to its properties, WebRTC is of particular interest for the development of the project for the present thesis.

As Sam Dutton points out in his article[5], WebRTC has now implemented open standards for real-time, plugin-free video, audio and data communication. This solves a range of problems relevant to real-time communications:

- Many web services already use RTC, but need downloads, native apps or plugins. These includes Skype, Facebook and Google Hangouts;
- Downloading, installing and updating plug-ins can be complex, error prone and annoying;
- Plugins can be difficult to deploy, debug, troubleshoot, test and maintain, and may require licensing and integration with complex, expensive technology. It's often difficult to persuade people to install plugins in the first place.

The guiding principles of the WebRTC project are that its APIs should be open source, free, standardized, built into web browsers and more efficient than existing technologies. Presently it is capable of enabling efficient transmission of audio, video and arbitrary data in a peer-to-peer fashion. At the same time WebRTC still needs servers because of the following reasons[6]:

- For clients to exchange metadata to coordinate communication: this is called signaling;
- To cope with network address translators (NATs) and firewalls.

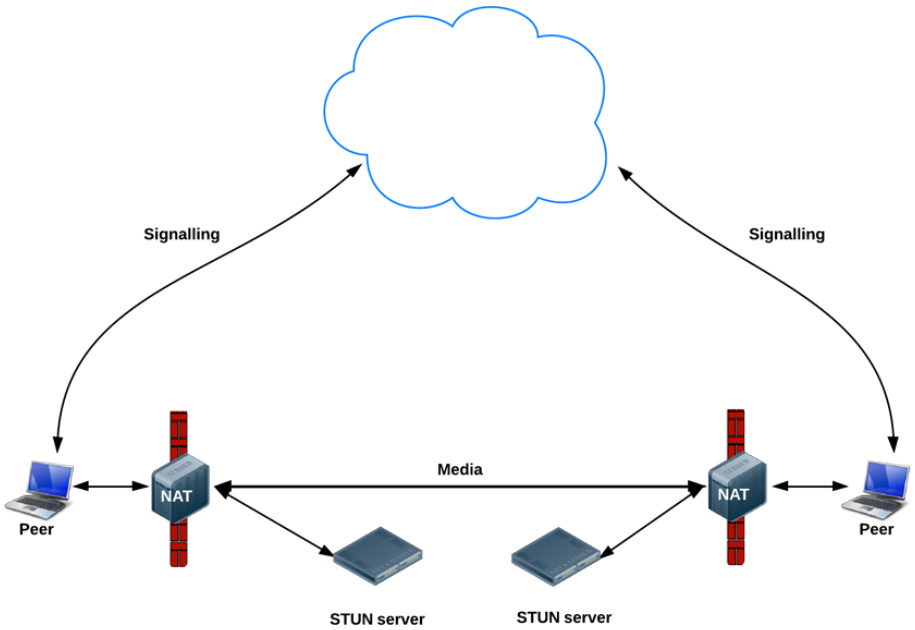


Figure 1.2 – Session Traversal Utilities for NAT (STUN)

Signaling is the process of coordinating communication. In order for a WebRTC application to set up a 'call', its clients need to exchange information like session control messages, error messages, as well as metadata and settings. This signaling process needs a way for clients to pass messages back and forth. To avoid redundancy and to maximize compatibility with established technologies, signaling methods and protocols are not specified by WebRTC standards. This approach is outlined by the JavaScript Session Establishment Protocol[7].

In order to find the best route from peer to peer, WebRTC makes use of ICE (Interactive Connectivity Establishment) technologies. Usually two kinds of methods are used, Session Traversal Utilities for NAT (STUN) and Traversal Using Relays around NAT (TURN). WebRTC uses a STUN server to find a direct connection between peers in order to achieve the smallest latency. If no such connection can be established, the process falls back to using a TURN server that works as relay for all further communications. Figures 1.2 and 1.3 showcase both situation in a visual manner:

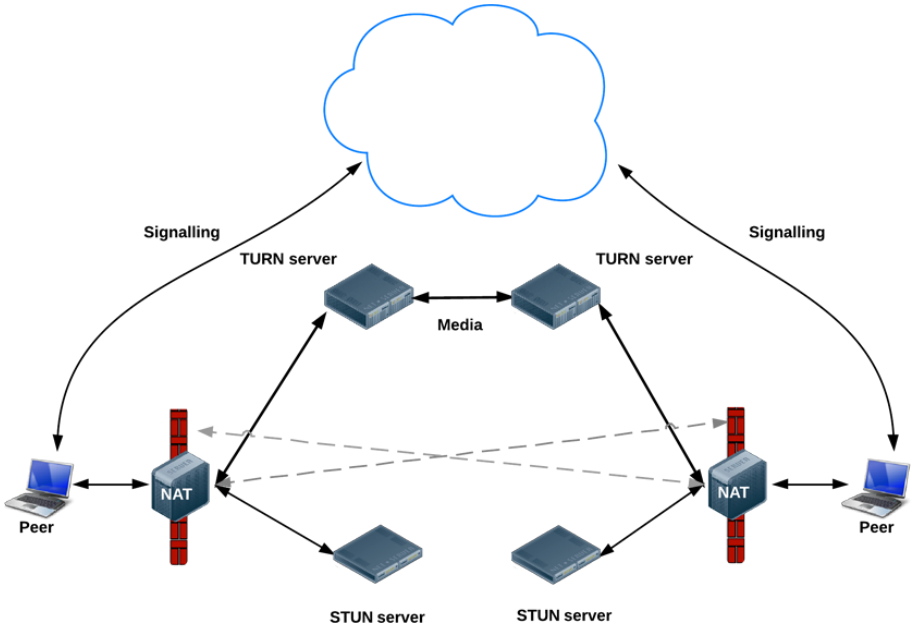


Figure 1.3 – Traversal Using Relays around NAT (TURN)

WebRTC is the most appealing technical solution for the project proposed in this thesis. The protocol will make sure that the players will be connected to the game using the shortest possible network route therefore having the smallest latency. In most cases players will be located behind the same NAT, and after establishing a connection, no data will travel further than to the router.

1.5 Existing Solutions on the Market

Even though WebRTC is a quite fresh technology, there are already plenty of projects using it. Some of these projects are created by developers as part of the Chrome Experiments[8] project, in order to demonstrate the technology and its capabilities. Two projects in particular are of interest to this thesis as they represent the concept of a mobile phone being used as a gaming controller. The following sections briefly describe the games and point out things that may be applied to this thesis.

1.5.1 Lightsaber Escape

The Lightsaber Escape[9] is a small single-player arcade game created by Lucasfilm Ltd in collaboration with Google in order to promote the seventh film in the Star Wars trilogy. It requires a personal computer a supported mobile device and an Internet connection. To play the game, the user has to access the main page of the game (`https://lightsaber.withgoogle.com/`) in a browser window on the computer (figure 1.4).

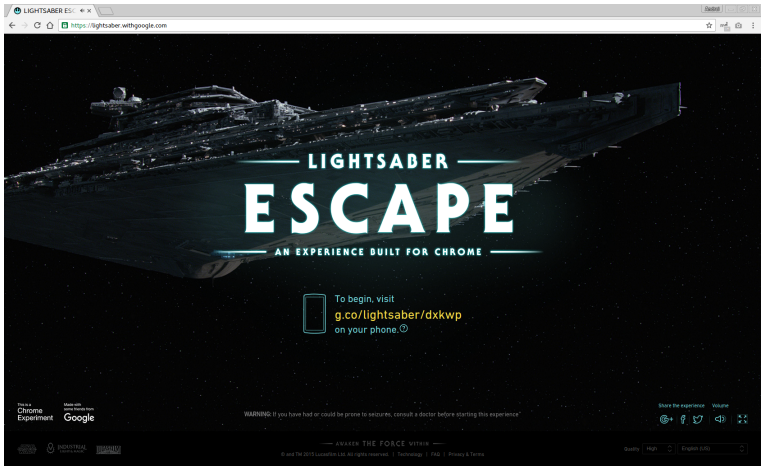
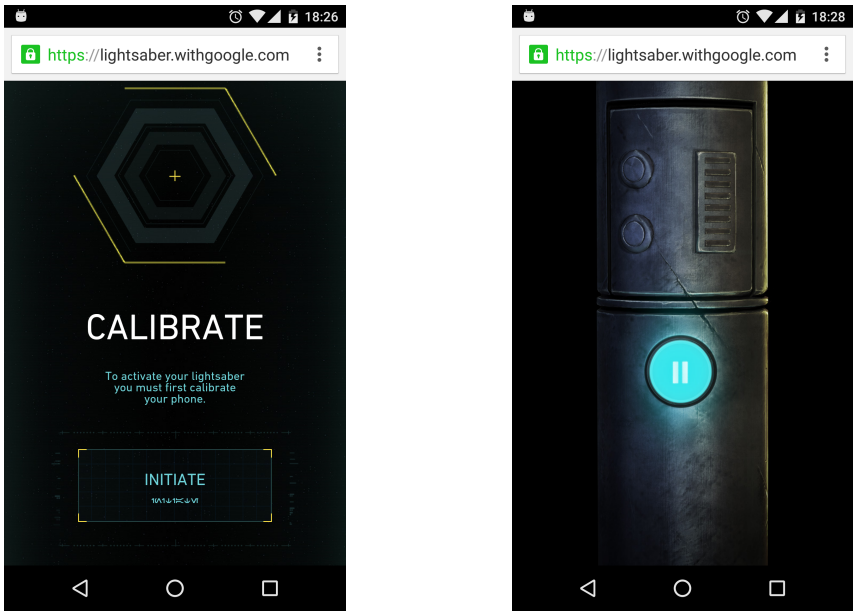


Figure 1.4 – Home Page of Lightsaber Escape

The user is presented with a short link that has to be entered in the navigation bar of the browser on the mobile device. This brings the user to a page with instructions to calibrate the device (figure 1.5 (a)). During the calibration process, a connection is established between the phone and the computer and the phone's gyroscope and accelerometer are calibrated. After this procedure the lightsaber becomes active (figure 1.5 (b)):



(a) Calibration Screen (b) Active Lightsaber

Figure 1.5 – Lightsaber Escape as Seen on a Mobile Phone

With an active lightsaber, the player can start the gameplay, which consists of dodging enemy laser blasts by controlling the lightsaber through movements of the mobile phone. The player passes a couple of corridors in such a manner, until he/she reaches a hand-to-hand combat encounter that resolves to the game's finale.

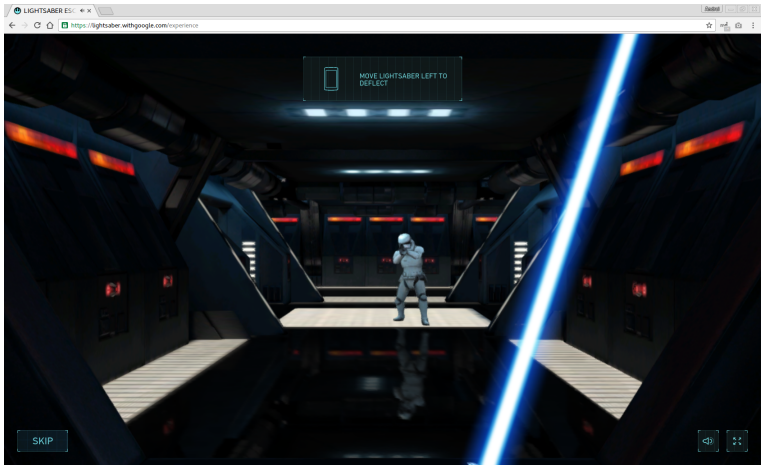


Figure 1.6 – Lightsaber Escape Gameplay

There are a couple of important things to note about this game. The clever way in which are leveraged the motion-detection capabilities of a phone makes for a really engaging experience, even though the whole gameplay lasts only about 5 minutes. If played on a powerful computer with a decent GPU, it could be noticed that control mechanism is very smooth and responsive, this is thanks to an efficient use of network traffic in the communication between devices. Overall this game is a good way to promote a product using innovative technology and it gives some hints on how to develop the project of this thesis.

1.5.2 Super Sync Sports

Super Sync Sports[10] is another Chrome Experiment with the same concept of connecting a mobile phone to a computer in order to achieve an interactive experience. This game has support for multiplayer gameplay for each of the three game modes. Cycling mode is shown in figure 1.7.

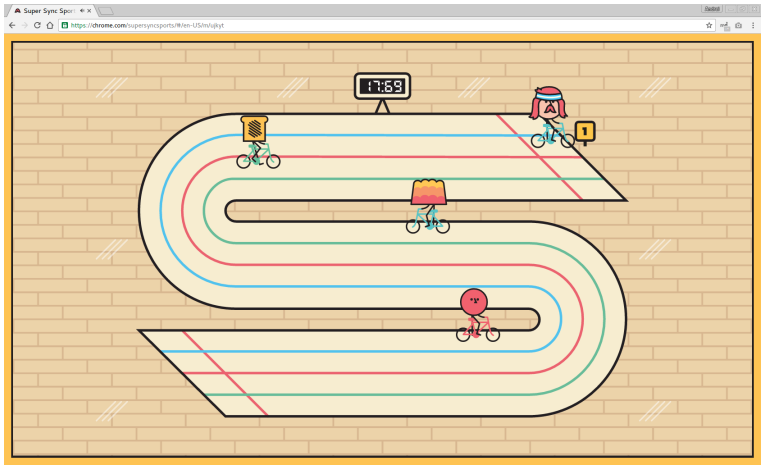
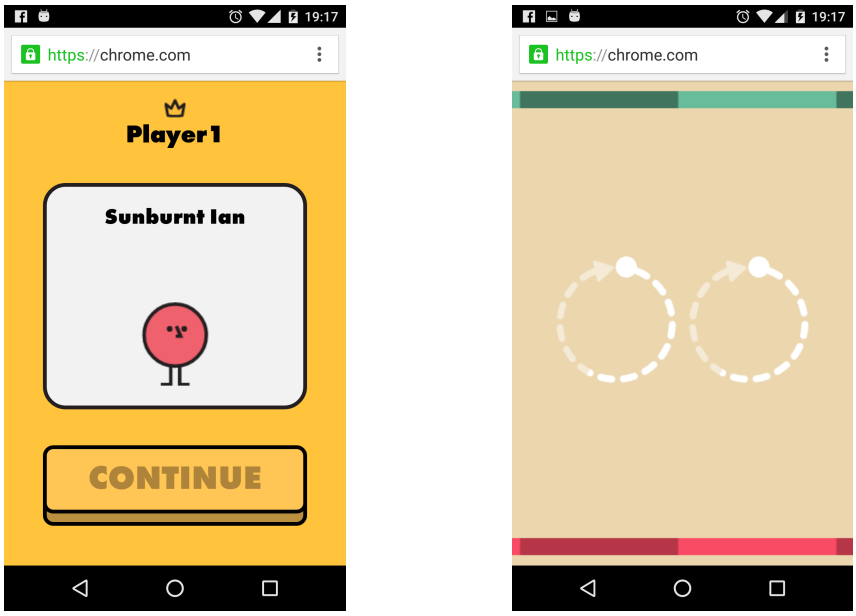


Figure 1.7 – Super Sync Sports Cycling Challenge

Once connected, players can choose a character (figure 1.8 (a)) and use the controls on the phone (figure 1.8 (b)) to race with each other on a track. In contrast to Lightsaber Escape, does not use motion-detection sensors of the phone, instead, the player activity is controlled by sophisticated touch screen gestures.



(a) Character Selection Screen

(b) Player Controls for Cycling Challenge

Figure 1.8 – Super Sync Sports on a Mobile Phone

From the technical point of view, Super Sync Sports and Lightsaber Escape share a lot of similarities like the connection procedures and the efficient use of WebSockets technology. It is also interesting how Super Sync Sports delegates things like character choice to the users phone rather than shows it on the common game screen. This greatly increases the game configuration speed and allows players to dive into the game right away.

1.6 The 'Snowfight' – Project Description

Both of the games described above are great examples of interactive experience achieved through the use of a mobile phone as a gaming controller. Under the hood, they employ one of the two communication technologies be it WebSockets or WebRTC.

The goal of this thesis is to replicate to some extent the functionality found in these games using the available developer tools and to identify patterns and techniques that can be used in similar projects. To achieve this, a multiplayer game will be created on top of aforementioned technologies. The concept is rather simple: players use their phones to connect to the game, they are placed in an isometric environment stylized as a winter playground. The gameplay represents a kind of 'death-match', in which players throw snowballs at each other with the attempt to eliminate all enemies from the game before being eliminated themselves.

The controller component of the game should consist of a gamepad-like system rendered on the phone's browser with two main control elements: a trackball for controlling player direction and

velocity, as well as a button for throwing snowballs. The communication between the controller and the game should be efficient enough to allow up to 6 player to experience a smooth and responsive gameplay, provided they are all connected to the same network.

Among other requirements, it should be possible to adjust the settings of the player's avatar using the phone. Things that can be adjusted are the player's on-screen name and the color of the avatar. Overall the game should represent a functional prototype that would illustrate use the technology and show potential of further development.

1.7 Domain Analysis Conclusions

As with any project, domain analysis is an important development step that is able to prevent a great part of poorly made decisions. It can also boost the creativity process as similar projects are analyzed and user opinion is considered.

This chapter tried to identify a way to merge the domains of gaming and real- time communication in an attempt to advance the concept of human interaction with computers. A brief history of user interfaces and gaming devices evolved into a discussion of how mobile phones could change the way people play games.

After that, two technologies where proposed to be used in order to achieved the defined goal, mainly WebSockets and WebRTC, and a description of the thesis project was provided in the context of two existing games that leverage the same tools and technologies. As a result it is evident that for the purpose laid out in the thesis, WebRTC is the best match, and should be used in the implementation of the project.

2 System Modeling

This chapter contains an attempt to model the application by use of UML diagrams so that the further development of the system is easier and has well defined boundaries. As a rule, the process of modeling an information system usually starts off by laying out a general picture and then deepens into more detailed views of the inner workings of the system.

2.1 System Use Cases

Use case diagrams are used to provide a good understanding of what a user can do with the application in question. In a nutshell, a game is out there to be played and the 'Snowfight' is no exception. The gameplay has a rather simple concept behind it and includes several things:

- Moving around the field;
- Throwing snowballs at adversaries;
- Being eliminated from a game by being hit with snowballs;
- Using the phone as a controller device.

Besides playing the game itself, the players are able to view game results after a match has finished and they can also modify the character's appearance and name before the game starts.

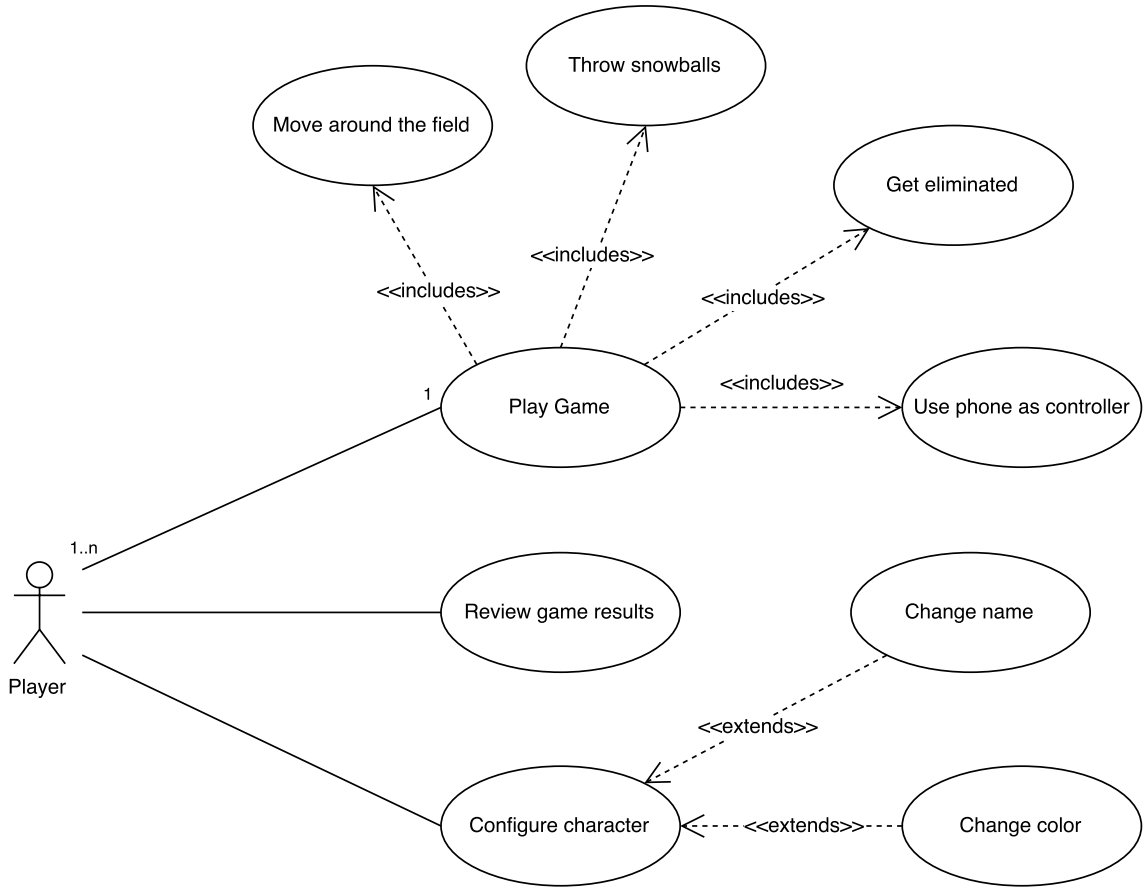


Figure 2.1 – Game Use Cases from the Player's Perspective

2.2 User Activity Graph

When it comes to the sequence of activities that may take place as part of the use of an application, the activity diagram does an excellent job at illustrating the transitions between different activities and various conditions that might alter the activity flow.

A typical set of activities that take place as part of playing 'Snowfight' is represented in the graph below. The players start off by accessing the index page of the game in a web browser. This brings them in a game lobby that contains a shareable URL used to connect controllers. Players navigate to this URL in the browser on their smart-phones, personalize some character configurations and connect to the game by pressing the start button on their 'controllers'. When all players are connected, the game is ready to start, which is done by hitting the respective button in the game lobby. Now the players enter the game loop, that is, they play a match, review the results and are given the opportunity to restart the match and when all players make their decisions the game is restarting leaving out those players who chose to quit.

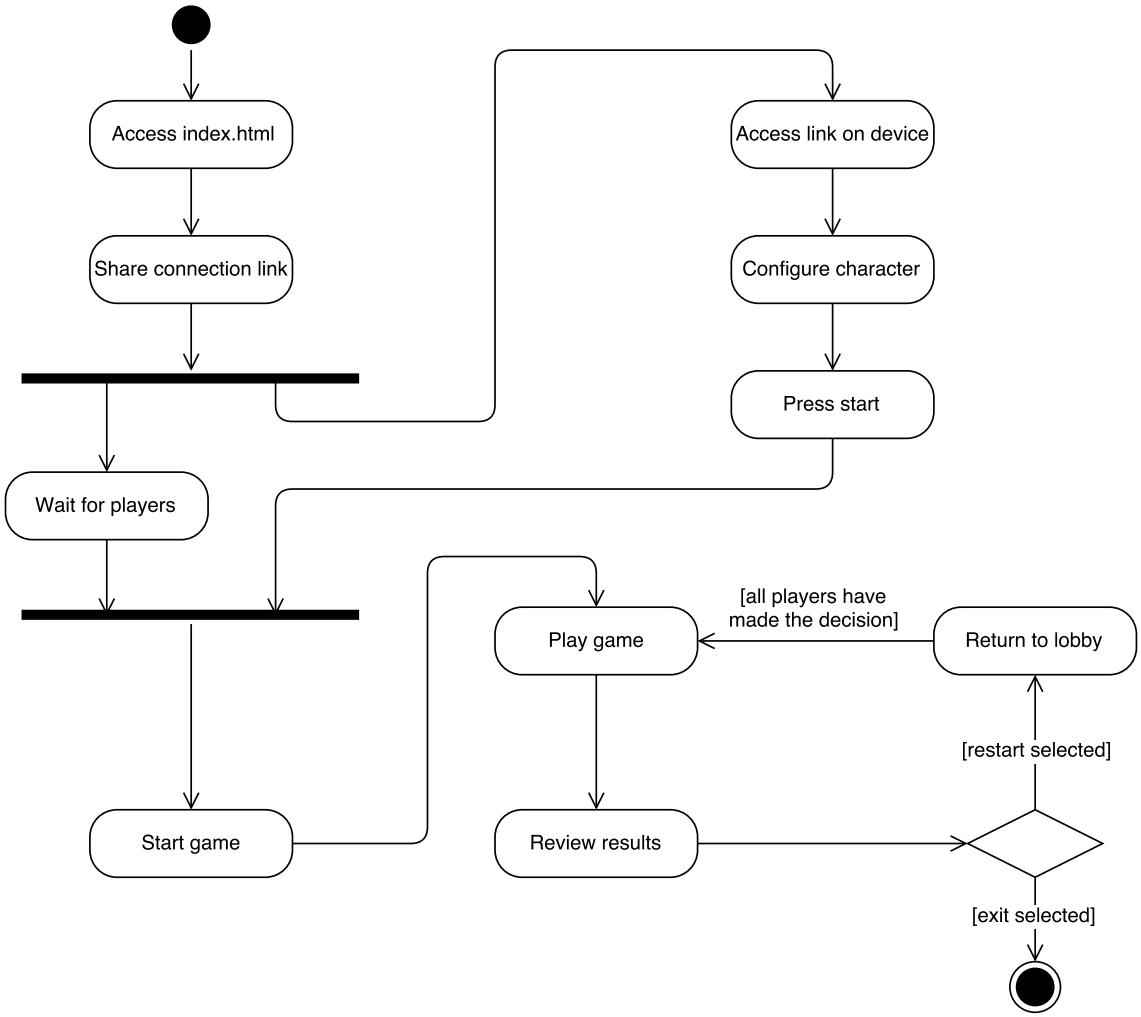


Figure 2.2 – Top-level Activities of the User

2.3 High-level Game States

When compared to simple web applications, games have a very substantial distinction in the sense that they are real-time system. A web page can be considered pretty much static once it has rendered, and all that happens afterwards is the result of various event handlers modifying the obtained document. Games, on the other hand, are being rendered to the screen every fraction of a second and keep changing as a living system. All this takes place in a, so called, game loop, and it is quite difficult to implement the various stages the game passes through in a single chain of 'if's. This is why even at the modeling stage, the game is split in different states through which it can pass and which have different things to render and process.

The game in question has four main states:

Lobby - works like a simple web page and is used to gather players before game;

Asset pre-loading - is the period after the 'start' button was hit, but before the game simulation begins, and it is responsible for preparing all the assets before they are used;

Active game - the state in which the main gameplay takes place that performs all physics computations and rendering, it also includes the game logic;

Results overview - is like an interlude between matches, when the game simulation is stopped and only information about the match is rendered to the screen.

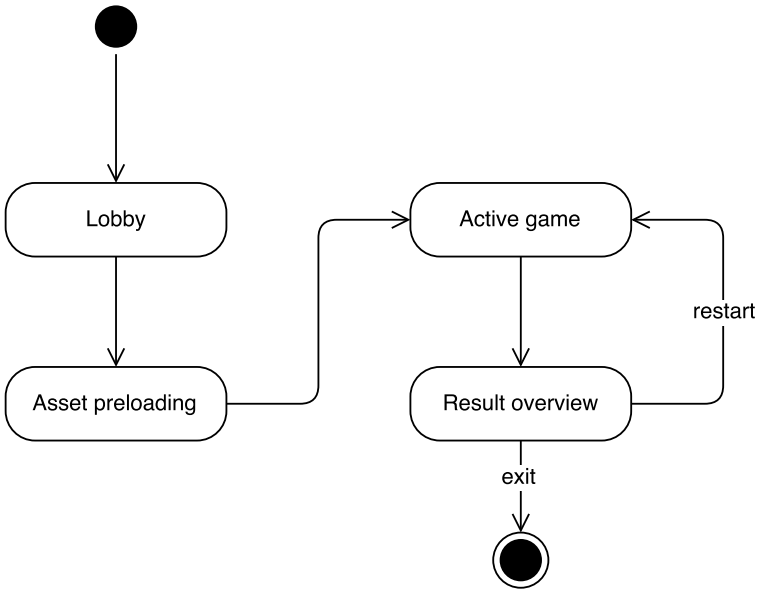


Figure 2.3 – Game States

Such a stateful approach encourages the separation of logic that concerns different parts of the system and uses an elegant way to interchange behavior when certain events happen. In addition, when game states are isolated from each other it is possible to use resources more efficiently as it is not necessary to process all game graphics and physics when the player is in the main menu.

2.4 Player States

At a lower level the game logic is controlled by the states of individual game objects and the main game object of the 'Snowfight' game is the player. It quite convenient to model the player as a state machine because the way it is render, the way it is updated and the manner in which it reacts to input heavily depend on its current state. When put in real context, it becomes clear that it is not possible to use the same animation or texture for a player that moves as well as for a player that is disabled. In these two states the player also has different kind of response to user input. When in default state, the player can move around the field as a reaction to the motion controller, as opposed to the situation of a disabled player when any kind of input has a null result.

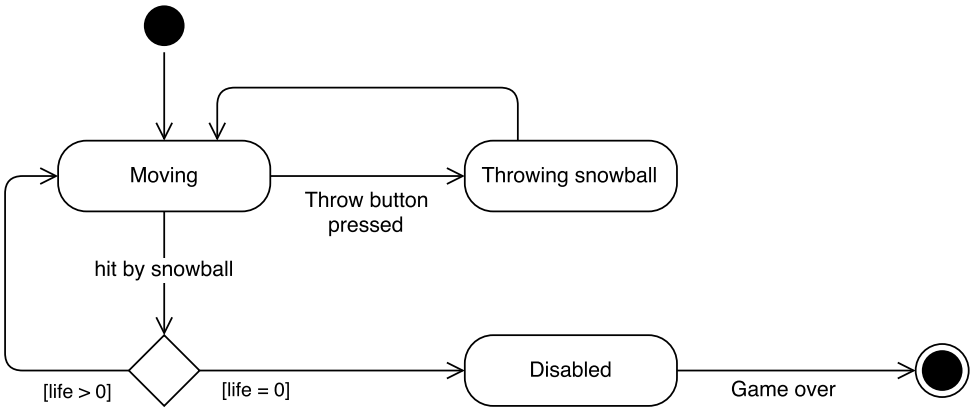


Figure 2.4 – Player States

As it can be observed in the diagram above, the *moving* state is the default state as well as the first one in which the player starts the game. The transition to another state might happen it two cases, and specifically, if the user hits the button for throwing snowballs, the player transitions to the respective state, the appropriate animation is played and the logic of trowing a snowball is triggered. As soon as the *throwing* process gets to its end, the player transitions back to the *moving* state automatically. In the second case, the state change is set off by a game event rather than through direct user input, that is, when a player is hit by a snowball of an adversary, the state machine reaches a decision point. The amount of 'health points' of the player in question is assessed and recalculated and if it is equal to zero, the player enters the *disabled* state and is considered eliminated from the game till the end of the match, otherwise, an automatic transition back to the *moving* state takes place.

Modeling entities as finite state machines can be very rewarding because they can untangle large chains of conditional statements and group relevant behavior of an object in isolated chunks of code. This approach is not a silver bullet, however, and also has some drawbacks. For example, if an object might need to know the previous state that it was in, but with the current setup it is oblivious. Therefore, more advanced systems usually make use of the concept known as push-down automata, this can be viewed as a combination between state machines and stack data structures. With push-down automata it is possible to return to the previous state of the object. For this game, however, it is perfectly enough to have a simple system with a small number of states.

2.5 Class Hierarchy of Game Objects

In the process of modeling a software system, an important role goes to laying out the relationships between different classes of objects. Class diagrams are recognized as one of the most effective tools that is able to convey a comprehensive view of the structure of an information system. When it comes to the game depicted in this thesis, the player is once again in the center of attention as it is acts as a container for various game components and represents the main logical unit of the game.

The diagram below illustrates the main classes defined in the project and the relations between them. It can be observed that the Player class responds to some standard methods that can be found in almost every game out there: *render*, *update* and *handleInput*. These methods are the connection points of this object to the rest of the game and are invoked in the respective sections of the game loop. Besides standard methods, a Player contains helper methods like *startMoving* and *throwSnowball* that would normally be invoked in the *update* method, however, the state machine architecture imposes a slightly different pattern. Instead of performing all the logic in the methods mentioned above, the Player delegates the decisions of 'what to do' to a state object that executes necessary actions depending on the specific state of the Player. The states are managed by an object of PlayerStateManager class, whose reference is stored in a field of a Player.

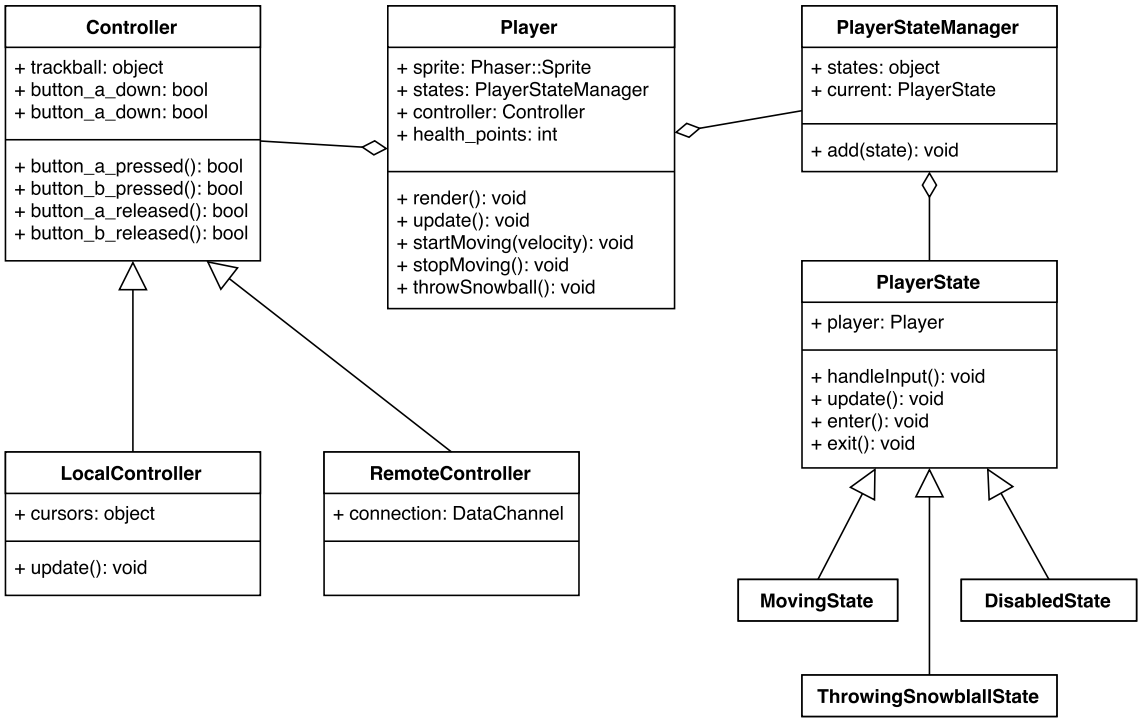


Figure 2.5 – Player Class and Its Components

Another aspect of the architecture worth mentioning is the unified interface of a controlling unit. The Player holds a reference to a Controller object, which, in turn, can be of different types. This is motivated by the fact that the game can be later modified to support artificial intelligence bots as adversaries at the same time maintaining the same controller interface. Thus, a Player can gather input information in the same way, regardless of the input source, whether it is a local keyboard, a remote smart-phone or just a bot script.

2.6 Components and Libraries

At the very beginning of the software development industry, programs had to be written almost entirely from scratch, which took quite a bit of time and effort, and usually it had to be redone with every following project. Today, however, the vast majority of recurring problems have been solved and their programming solutions can be found in various libraries that can be included and used in independent projects. This way, software development has become biased from writing all the code from zero, towards a more modular approach where applications would share pieces of reusable software and the programmers would only write code to glue all of it together.

Below is a component diagram that represents the dependency graph of the 'Snowfight' game. It points out what are the main components of the application and which libraries are used to obtain the desirable functionality.

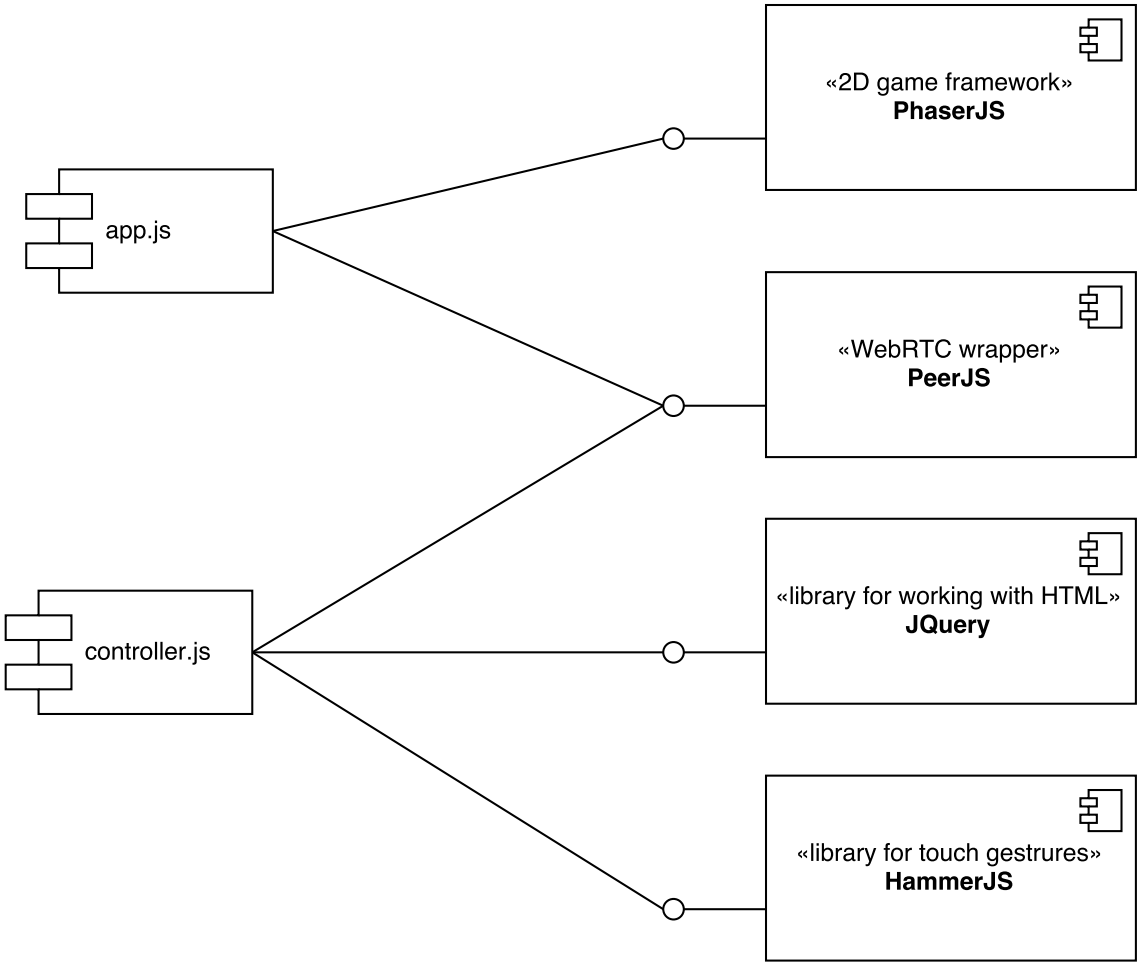


Figure 2.6 – Component Diagram

The two components of the application are the main game module and the module of the controller code. These modules share a dependency which is the 'PeerJS' library. It is used to establish peer two peer connections between the game and controllers, thus, it should be present in both places. The main game makes extensive use of the PhaserJS game development library, while the controller code exclusively includes a library for processing touch gestures called HammerJS.

2.7 Deployment Setup

When the application is moved to production it is important to understand the deployment configuration of the system in order to be able to manage it efficiently. In case of the game in question, it is not enough to consider only a web server that would host the application files. The specifics of peer to peer communication protocol involved in the project, require a dedicated service for brokering the connections. In setups where raw WebRTC is used it is necessary to deploy a STUN or TURN server. On the other hand, this project includes a wrapper library that features a pre-configured server application. This server application can be deployed in one of the three variants:

- Hosted on the same machine as the web server;
- Deployed to a separate dedicated server;
- A third-party SaaS (Software as a Service) solution.

The diagram that follows, illustrates the second type of deployment when the web server and the 'PeerJS' server are hosted on two different dedicated machines. The web server delivers the appropriate scripts to the client's devices. This communication is performed using HTTP or HTTPS protocols. Client devices (a computer and a phone), in turn, establish a peer-to-peer WebRTC connection, as a result of communications with the broker server which are executed over a combination of HTTP and WebSocket protocols.

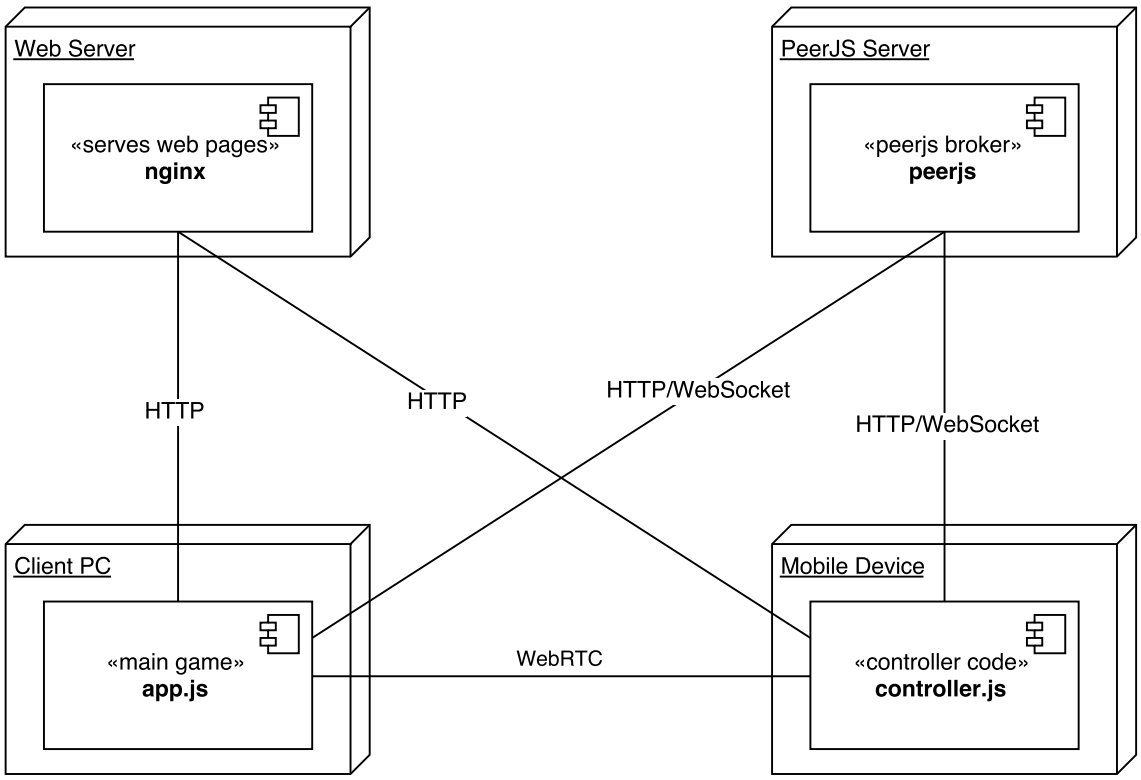


Figure 2.7 – Deployment Diagram

2.8 System Design Conclusions

This chapter provided an overview of the system design process of the 'Snowfight' game. By means of different types of UML diagrams it was possible to obtain several perspectives of the most important parts of the application model from different points of view.

It can be observed that the application design follows a couple of architectural patterns found mostly in the game development industry, for example, the stateful approach to modeling game objects. On the other hand, a lot of particularities still can be traced back to the fact that the application is not a native one, but rather web-based.

The value of system design can be hardly seen in such small projects as the one depicted in this thesis, however, large systems cannot go without a good round of design and analysis. Moreover, developer teams with a big number of employees can vastly benefit from having diagrams that describe various parts of a system. These provide documentation and make it easier for people who are new in the project to pick up the material and continue further development.

3 System Implementation Details

In this chapter is exposed a detailed description of the implementation details of the thesis project. In the following sections are presented script excerpts from both, the game and the controller parts of the system as well as the communication code that links them together.

3.1 Main Game

The development of computer games is one of the most complex branches of the software industry. Games encompass about a dozen different fields of mathematics and computer sciences, to name a few, there is three-dimensional calculus, differential calculus for physics, computer graphics, artificial intelligence, game theory, etc.

Due to its complexity, game programming is among fields where the concept of libraries and frameworks is more relevant than ever. Along the years, developers around the globe have stumbled on the same problems in a recurring manner and as a result, a lot of experience was gained which today is materialized as game engines and development kits for all kind of platforms and types of games. In this context it is much easier to prototype a game than it was ten years ago.

This thesis uses an HTML5 game framework called *Phaser*[11] which features a rich set of tools that are required to build a 2D video game. This includes rendering to an HTML Canvas or WebGL context, a physics engine, particle system, asset management framework, sound engine, input handling, tiles, sprites and much more. This framework was selected due to the ease and speed with which one can develop a working prototype game that looks nice enough to be presentable. On the other hand, 'Snowfight' is designed as an isometric game and doesn't meet the main purpose of the framework, but this problem is easily solved by making use of the framework's powerful plug-in system. The isometric plug-in extends the Phaser capabilities like physics into the three-dimensional world and perform an isometric projection on a two-dimensional canvas afterwards.

3.1.1 The Design of a PhaserJS Game

In order to use a tool efficiently it is important to understand how it works. The Phaser framework follows classic game programming patterns and has a standard game loop which consists of three main steps:

Input handling - the part where the user input is collected and transformed into actions that are applied to the world whether it is changing the direction and velocity of the player's movement or throwing a snowball;

Simulation update - represents the process of updating the parts of the system that do not directly depend on user input, like collision resolution, object position adjustment depending on its velocity and such things as choosing and applying the right frame of a sprite animation;

Rendering to screen - consists of drawing all elements and textures on an HTML5 Canvas or a WebGL context. When using Phaser, the programmer would seldom override this step as the framework already does most of the work, with of some rare cases when very specific adjustments and post-processing is needed.

In Phaser games, the loop itself is hidden under the hood of the framework, while a simple yet flexible interface is provided to the programmer to control and fine-tune the steps specified above. In order to bootstrap a game it is enough to instantiate a Phaser::Game object while providing the necessary callbacks as in the example below:

```
var game = new Phaser.Game(800, 600, Phaser.AUTO, '', { preload: preload, create:
    create, update: update });

function preload() {
}

function create() {
}

function update() {
}
```

Listing 3.1 – Minimal Setup for a Phaser Game

The three methods in the example are not the only ones that can be overridden and Phaser documentation list all of them and explains how they can be used to customize the various use cases. The next section not only presents these methods, it describes how different versions of them can be specified to be used in different context, thanks to the concept of game states.

3.1.2 Game State Management

Every game is like a living system and everything a user sees happens inside a game loop. Even if the screen shows the same static image, the game loop still runs and refreshes the screen about 60 times per second. At the same time, most games have a couple of situations when they behave completely different, the most prominent example being the case of a three-dimensional shooter or racing simulation game when the user is in the main menu compared to the time when he/she is engaged in the gameplay itself.

At the implementation level, it would be quite inconvenient to make the same decision dozens of times per second, specifically the decision of choosing whether to render the main menu or to compute the world physics and render the game. The decision tree and, respectively, the chain of flow-control structures grow with the number of these states that the game might be in.

Software development techniques already include a solution to this kind of situation in the body of a design pattern called the state pattern. *The Gang of Four*[12] describe a state object as one that encapsulates some state-dependent behavior. This maps exactly to what happens in games, various game states like main menu, active gameplay, cinematic cut-scenes, can be modeled as objects that have specific implementations of methods for rendering, performing updates and handling user input every frame.

The Phaser framework makes extensive use of the state pattern and gives developers the opportunity to model their games as a series of interchanging states with a set of predefined methods that are called by Phaser at specific times in the main game loop and can be overridden in order to

achieve certain behavior. Some of the most commonly used methods of the abstract State object are presented below with a small description of what are they usually used for:

init() – the very first function called when the State starts up;
preload() – normally used to load game assets;
create() – called when the State is ready to enter the game loop;
update() – is for programmer's own use in order to define main game logic;
preRender() – called after all Game Objects have been updated, but before any rendering;
render() – called after the game is rendered, for final post-processing and style effects;
shutdown() – will be called when the State is shutdown.

As the 'Snowfight' game is still quite small at this stage in development, it features two main game states. Listing 3.2 shows the code that defines the 'preload' stage of the game. It is responsible for loading the assets and bootstrapping all of the game systems like physics and plug-ins. For this purpose it defines the *preload* and *create* methods.

```
1  var Preload = function () {};
2
3  Preload.prototype.preload = function () {
4    this.load.spritesheet('knight_8frame', 'assets/img/sprites/knight/8
       frame_combined.png', 70, 70);
5
6    this.load.image('tile',      'assets/img/sprites/tile.png');
7    this.load.image('snowball', 'assets/img/sprites/snowball_small.png');
8  };
9
10 Preload.prototype.create = function () {
11   // Add and enable the plug-in.
12   this.game.plugins.add(new Phaser.Plugin.Isometric(this.game));
13
14   // Start the IsoArcade physics system.
15   this.physics.startSystem(Phaser.Plugin.Isometric.ISOARCADE);
16   this.physics.isoArcade.gravity.setTo(0, 0, -500);
17   this.game.iso.anchor.setTo(0.5, 0.5);
18
19   this.game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL
20
21   this.state.start('play');
22 }
23
24 module.exports = Preload;
```

Listing 3.2 – Preload State

The 'play' state, on the other hand, represents the description of the main behavior of the game. It overrides the *create*, *render* and *update* methods and contains all the logic necessary to control gameplay process.

3.1.3 Player's State Machine

Similarly to the architecture of the whole game, various subsystems can be also modeled after the state pattern. Specifically the Player's behavior is heavily dependent on the state that it is in at a given point in time. The need for a stateful design aggravated at the point of developing the input handling system as depending on the state of the player, user input had to be processed in very different ways, for example when a player is disabled, movement controls have no effect as opposed to the normal activity of the player.

The programming concept behind state handling of the player is similar to the one used in the game object. A player object holds a reference to a state manager that is responsible for keeping track of the current state as well as adding and storing other states. A state object, at the same time, holds the necessary logic to perform input handling or a player update. It also includes the definition of the actions that have to be executed when a player enters or exits a state. With this setup, when the game passes through the game loop and a player receives a call of the update method, for example, it delegates it to the state manage which in turn calls the method on the current state object.

Listing 3.3 presents the definition of a dummy state as well as the code of the state manager.

```
var PlayerState = function (player) {
  this.player = player;
}

PlayerState.prototype.handleInput = function (controller) { }

PlayerState.prototype.update = function () { }

PlayerState.prototype.enter = function () { }

PlayerState.prototype.exit = function () { }

var PlayerStateManager = function (player) {
  this.player = player;
  this.states = {};
  this.current = new PlayerState();
}

PlayerStateManager.prototype.add = function (name, state_class) {
  this.states[name] = state_class;
}

PlayerStateManager.prototype.start = function (name) {
  this.current.exit();
  this.current = new this.states[name](this.player);
  this.current.enter();
}
```

Listing 3.3 – Player States

In his book on game programming patterns[13], Robert Nystrom provides an excellent example of how games can leverage the science behind the automata theory and how a nested chain of *if* statements can be converted to an elegant finite state-machine. The Player class in 'Snowfight' tries to follow that example and model the object as a graph of states and transitions.

Diagram 2.4 from the chapter about system design shows the states that a player can have (*moving*, *throwing* a snowball and *disabled*) and the various transitions that might happen between them. In some cases a transition takes place as a result of an event or a condition that evaluates to truth, at the same time some states transition to the next state immediately after finishing their job. A good example of such behavior is the transition from the player's state of *throwing* a snowball back to *moving*, which happens right away without additional condition checks.

At the implementation level transitions are performed by the player state manager and one can be triggered by a single line of code:

```
MovingState.prototype.handleInput = function (controller) {
  // ...
  if (controller.button_a_pressed()) {
    this.player.states.start('throwing_snowball');
  }
  // ..
}
```

Listing 3.4 – Moment of Transition to the Throwing State

3.1.4 Physics and Rendering

An important part of a game is how it looks and feels. This mainly depends on the physics and rendering systems of the game. Luckily, the Phaser framework includes a decent physics engine that is capable of resolving rigid body collisions and applying physical forces to objects. Rendering is also mostly handled by the framework and the customization of the visual aspect of the game consists of picking or drawing the right textures for the sprites.

The 'Snowfight' is an isometric game and in order to turn Phaser into a pseudo- three-dimensional game engine, a respective plug-in is used. It extends the coordinate space to a three-dimensional space and allows to perform collision resolution in such conditions.

```
// Start the IsoArcade physics system.
this.game.physics.startSystem(Phaser.Plugin.Isometric.ISOARCADE);
```

Listing 3.5 – Isometric Plug-in Setup

Player animation is also an important aspect of the game. Phaser supports sprite animation. In case case an animated object represents just a rectangle whose texture is repeatedly changed in order to create the illusion of motion. It is a widely used technique in 2D games. In case of games with an isometric projection the sprites seem to gain an additional dimension. This is, however, just an illusion created by graphics artists and the definition of the sprite remains the same, it is a rectangular texture oriented towards the camera.

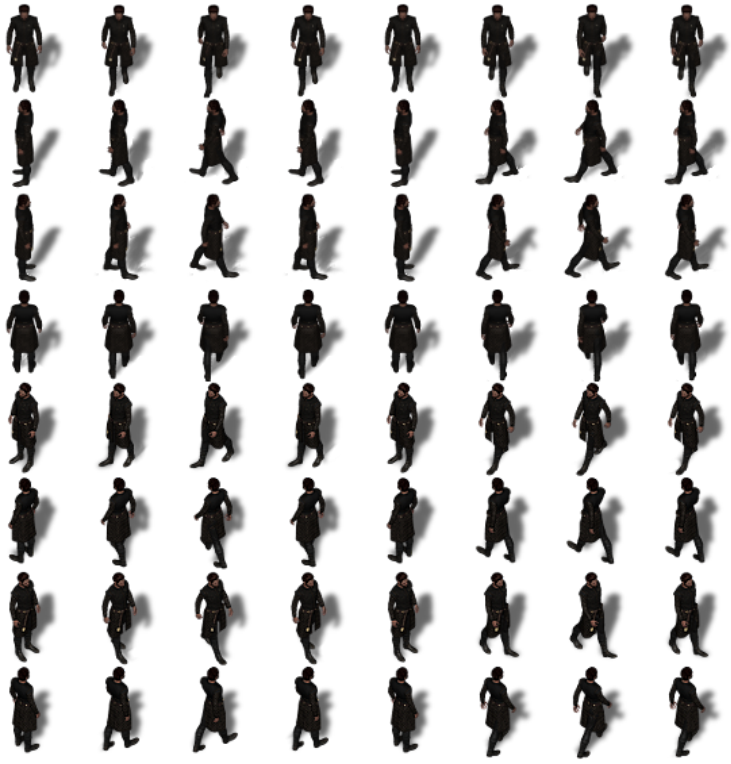


Figure 3.1 – Player Spritesheet

Figure 3.1 represents a spritesheet with 8 looped animations of character motion in 8 directions with 8 frames each. In order to load and use these animations in the game it is necessary to tell Phaser the frame indices of each animation and associate every batch with a name. The listing below demonstrates this procedure:

```
Play.prototype.makePlayerSprite = function (x, y) {
  var sprite = this.game.add.isoSprite(x, y, 100, 'knight_8frame',
                                       0, this.groups.players);
  sprite.anchor.set(0.5);
  this.physics.isoArcade.enable(sprite);
  sprite.body.setSize(18, 18, 60, 0, 0, -18);

  sprite.animations.add('south_east', [0, 1, 2, 3, 4, 5, 6, 7],  20, true);
  sprite.animations.add('south_west', [8, 9, 10,11,12,13,14,15], 20, true);
  sprite.animations.add('north_east', [16,17,18,19,20,21,22,23], 20, true);
  sprite.animations.add('north_west', [24,25,26,27,28,29,30,31], 20, true);

  sprite.animations.add('south', [32,33,34,35,36,37,38,39], 20, true);
  sprite.animations.add('west',  [40,41,42,43,44,45,46,47], 20, true);
  sprite.animations.add('east',  [48,49,50,51,52,53,54,55], 20, true);
  sprite.animations.add('north', [56,57,58,59,60,61,62,63], 20, true);

  return sprite;
}
```

Listing 3.6 – Sprite Animation Loading

3.1.5 Uniform Interface for Input Handling

The development process becomes increasingly more difficult as the number of moving parts in the system grows. It also becomes harder to debug and test individual subsystems in isolated environments and in the case of 'Snowfight' the development of the game was substantially slowed down by the fact that every time the page was refreshed, it was necessary to reconnect the controller. In addition, when some problems appeared it was not clear right away whether the problem was in the game logic, in the controller code or in the communications in-between.

One solution to the situation described above was to separate the game from the controllers and develop it separately by providing the necessary input from the local keyboard. This way, communication errors and bugs that concern controller rendering would not stagnate the evolution of the core game.

From the implementation point of view the task required a unified interface for all possible input sources, in this case only two of them. However, such an approach opened opportunities to connect an artificial intelligence bot to the abstract input device, which permits the addition to the game of non-player characters (NPC) that would be controlled by the computer.

The base class of the Controller sets up the necessary variables and a set of helper functions like the one in the listing below (3.7), that performs a check if a button was pressed just before the function call or it was down all along.

```
var Controller = function () {
  this.trackball = { x: 0, y: 0 };
  this.button_a_down = false;
  this.button_b_down = false;

  this.button_a_was_down = false;
  this.button_b_was_down = false;
}

Controller.prototype.button_a_pressed = function () {
  if (!this.button_a_was_down && this.button_a_down) {
    this.button_a_was_down = true;
    return true;
  } else {
    this.button_a_was_down = this.button_a_down;
    return false;
  }
}
```

Listing 3.7 – Controller Base Class Highlights

At the same time, listings 3.8 and 3.9 contain the specific bits of code that govern input collection from the local keyboard and a remote mobile device respectively. The main difference between the two is the fact that a local controller should be update every iteration of the game loop in order to represent the accurate state of the input device, while the remote controller is updated asynchronously by means of callbacks.

```
var LocalController = function (phaser_input) {
  Controller.call(this);

  this.cursors = phaser_input.keyboard.createCursorKeys();
  this.a_key = phaser_input.keyboard.addKey(Phaser.KeyCode.A);
  this.b_key = phaser_input.keyboard.addKey(Phaser.KeyCode.B);
}

LocalController.prototype.update = function () {
  var v = { x: 0, y: 0 };

  if (this.cursors.up.isDown) { v.y -= 50; }
  if (this.cursors.down.isDown) { v.y += 50; }
  if (this.cursors.left.isDown) { v.x -= 50; }
  if (this.cursors.right.isDown) { v.x += 50; }

  this.trackball = v;

  this.button_a_down = this.a_key.isDown
  this.button_b_down = this.b_key.isDown
}
```

Listing 3.8 – Local Controller

```
var RemoteController = function (connection) {
  this.connection = connection;

  Controller.call(this);

  connection.on('data', function (data) {
    switch (data.type) {
      case 'trackball':
        this.trackball.x = data.x;
        this.trackball.y = data.y;
        break;
      case 'button_pressed':
        this['button_' + data.name + '_down'] = true;
        break;
      case 'button_released':
        this['button_' + data.name + '_down'] = false;
        break;
      default:
        break;
    }
  }.bind(this))
}
```

Listing 3.9 – Remote Controller

3.2 Controller

The second major part of the system is the remote controller that is used to operate a character on the main game screen. Following the motivations described in the chapter on domain analysis, it represents a web page rather then a native mobile application. As a result, it requires no prior installation procedures because any modern mobile device is likely to have a decent web browser that supports the necessary technological specifications.

The primary goal of the controller component is to provide an experience similar to using a dedicated gaming device like a gamepad. The touch screen of a mobile phone can be used to model various control elements like buttons and joysticks. Even thought these fall short of three-dimensional tactile feedback, they can be customized and optimized for an engaging and efficient gaming experience. Moreover, modern browser APIs can trigger device vibrations of different patterns in order to provide additional feedback.

The design of the controller and the kind of control elements it contains depend on the type of actions they have to operate. In case of 'Snowfight', there are two main activities that are controlled by the user, these are *movement* and *throwing* snowballs. The mechanics of the game state that a snowball is thrown in the direction of the player's current orientation. This means that it is not necessary to have a control element like a joystick that would control the direction, instead, a simple button should be enough to perform this action. Movement, on the other hand, is a more complex activity and is controlled two parameters, mainly, direction and speed. An ergonomic solution to this task is a joystick-like control element. In the context of this system, the element is called a track ball as it resembles a ball when viewed on the screen.

Figure 3.2 represents a development version of the web-based 'gamepad' rendered on the screen of a mobile device in landscape mode. The left side contains the track ball element for movement and the right side has the button for throwing snowballs. The blue button at the top is a shortcut for switching to full-screen mode.

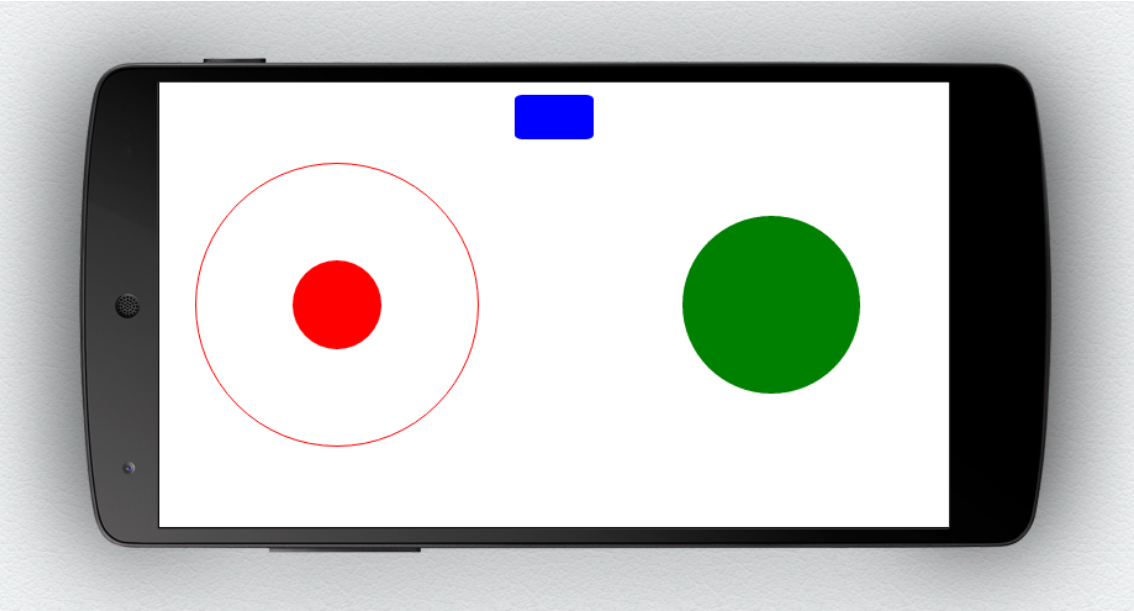


Figure 3.2 – Web-based Gamepad

As it is a web application, the controller and its elements are nothing but an HTML5 document with appropriate styles and some scripts attached. The controller layout is presented in the listing below. All elements are constructed of *div* tags that have specific class and id attributes in order to be later processable by style-sheets and scripts.

```
<div class="controller">
  <div class="left-side">
    <div id="trackball-one" class="trackball-container">
      <div class="circle"></div>
      <div class="trackball"></div>
    </div>
  </div>

  <div class="right-side">
    <div class="button-container">
      <div id="btn-a" class="btn"></div>
    </div>
  </div>

  <div id="fullscreen-toggle"></div>
</div>
```

Listing 3.10 – HTML Layout of the Controller

3.2.1 Track Ball Control Element

In order to control character movement, it is necessary to represent some how numerically this activity. In its simplest instance, movement on a plane has a direction and speed, this maps perfectly onto the characteristics of a two- dimensional vector, specifically its orientation and magnitude. Given a data model, the next step is to come up with a control element that is able to generate such data as a result of user input. On conventional gamepads this feature is attributed to joysticks. They can be tilted to a certain degree within a range in 2 directions, thus generating a set of 2D coordinates. Joysticks also include a spring that returns them to the initial position in case of no user input.

In the context of a touch screen device, it is not that simple to create a moving stick, however, it is possible to draw a kind of ball that would track the motion of a finger pressing on the screen and would reset to its initial position when the touch motion is terminated. In addition, the web browser interprets touch events in a different way from events generated by a mouse device. Moreover, complex gestures like *pan*, *pinch zoom*, *rotate* and *swipe* require considerable effort in writing code that detects these gestures, identifies the right type and fetches appropriate data sets. Luckily, these problems have been solved by other people who developed libraries specifically for such tasks. This thesis project makes use of the HammerJS[14] library that can recognize an handle gestures made by touch, mouse and pointerEvents. Gesture recognition is easily enabled by creating a *Hammer* object instance and specifying the HTML element that should trigger the events. Different options can be passed along in order to customize different parts of the recognition process.

```
this.hammer = new Hammer(this.ball[0]);
this.hammer.get('pan').set({ direction: Hammer.DIRECTION_ALL });

var that = this;

this.hammer.on('panstart', function (ev) {
  that.ball.css({transition: 'all 0.0s'});
  that.clampAndMove(ev);
  that.callbacks.onstart(that.normalizeCoords(that.ballPosition));
});

this.hammer.on('panmove', function (ev) {
  that.clampAndMove(ev);
  that.callbacks.onmove(that.normalizeCoords(that.ballPosition));
});

this.hammer.on('panend', function (ev) {
  that.clampAndMove(ev);
  that.callbacks.onend(that.normalizeCoords(that.ballPosition));
  that.resetBall();
});
```

Listing 3.11 – HammerJS Setup

Listing 3.11 demonstrates how the ball element is passed to a *Hammer* object. After that, an option is specified to track *panning* motion in all direction. The *pan* recognizer generates 3 kinds of events: *panstart*, *panmove*, *panend*. The listing above also shows how callbacks are attached to each of the events in order to make use of the information that is generated by them.

3.2.2 Button

The button part of the controller was considerably easier to implement as there where no gestures involved. However, there was one aspect that had to be clarified. When a button is pressed, its color should change to give some kind of feedback to the user. This can be done through CSS pseudo-classes in a trivial way, but it turns out such an approach has a drawback. There is a delay of approximately 300ms between clicking on the button and the color change. In case of usual websites it is almost unnoticeable, however, in case of games, immediate feedback is crucial and the lack of it impacts severely the user experience.

A logic workaround to this problem is to trigger the color change through JavaScript code, by adding or removing the respective CSS class attribute, as it is shown in listing 3.12

```
$('#btn-a').on('touchstart mousedown', function (event) {
  // ...
  $(this).addClass('pressed');
  // ...
});
```

Listing 3.12 – Button Color Switch

3.3 Peer to Peer Communication

Communication is the focal point of this thesis and makes all parts of the system work as a whole. The game and the controller are nothing but simple web applications and present no special value by themselves. In order to make them communicate and interact, it is possible to apply several techniques. Most things on the web communicate under a client-server architecture. This implies there is an entity (a server) that orchestrates the whole process, it performs all computations and provides services to the clients who always connect directly to it. This project, however, has a slightly different twist to it.

The game and the controller are both applications are executed on the client machine, moreover, there can be many simultaneous game sessions that must not overlap in terms of communication. If the system were to employ the client- server architecture, all the communications would have to be proxied through a main server which would maintain control over all games and controllers. Obviously, this model would be painfully slow and inefficient as large amounts of data would have to travel long distances between devices and the server even though the game computer and all the player are located in the same room and connected to the same network.

In this context, the best option is a peer-to-peer communication model that permits the exchange of information between parties directly, without the use of intermediaries. Modern browser have recently introduced support for the WebRTC technology that enables peer-to-peer communication with the use of a server only for establishing the connection.

3.3.1 Setting Up a Connection with PeerJS

WebRTC programming interfaces are quite low level and performing the basic tasks requires considerable effort from the developer. As with other software there are libraries that have gathered the boilerplate code that is often used in projects and hide it under simpler and higher level APIs. 'Snowfight' uses PeerJS[15] which wraps the browser's WebRTC implementation to provide a complete, configurable, and easy-to-use peer-to-peer connection API. Equipped with nothing but an ID, a peer can create a P2P data or media stream connection to a remote peer.

```
1  var satellite = new Peer({
2    host: Settings.peerjs_host,
3    port: Settings.peerjs_port,
4    secure: Settings.using_https,
5    debug: Settings.peerjs_debug_level
6  })
7
8  var hub_id = 'test';
9
10 var dataConnection = satellite.connect(
11   hub_id, { metadata: { player_name: player_name } }
12 );
```

Listing 3.13 – PeerJS Initialization

Listing 3.13 shows how easy it is to initialize a Peer object. It is enough to provide the address and port of a PeerJS server that will broker the connections. A data connection is established simply by invoking the *connect* method on the Peer object. At the same time it is possible to send arbitrary metadata that will be associated with the connection, in this case it is the player's name which can be entered by the user. Once a connection is open, it is trivial to send data over it:

```
dataConnection.send({
  type: 'button_pressed',
  name: 'a'
});
```

Listing 3.14 – Sending Data Using a 'DataConnection' Object

When a controller initiates a peer-to-peer connection, the game lobby has to catch this event and execute the necessary procedures in order to be able to receive data. As many other things in JavaScript this is done by the use of callbacks that are attached to specific events. Listing 3.15 shows that every incoming connection object, that is intercepted by the game while in lobby, is collected in a JavaScript array.

```
var hub = new Peer(hub_id, {
  host: Settings.peerjs_host,
  port: Settings.peerjs_port,
  secure: Settings.using_https,
  debug: Settings.peerjs_debug_level
});

var connections = [];

hub.on('connection', function(dataConnection) {
  connections.push(dataConnection);
});
```

Listing 3.15 – Connection Gathering

Later on, at the initialization step of the game, every connection is attached to a *RemoteController* object, which in turn, sets callbacks (listing 3.16) that process incoming data and update the internal state of the controller.

```
var RemoteController = function (connection) {
  this.connection = connection;

  Controller.call(this);

  connection.on('data', function (data) {
    // perform data processing
  }.bind(this))
}
```

Listing 3.16 – Controller Callbacks

3.3.2 Communication Protocol

Establishing a connection is definitely not enough for an efficient and fruitful communication. The chosen format of data that is to be exchanged between the parties can heavily influence the flexibility and the performance of the system. Devising a protocol is not a simple task because there are a lot of questions that a developer should be able to answer before proceeding to the implementation step.

Some of the things that should be clarified are listed below:

- what kind of data to transmit;
- how to order the data inside of a message;
- how to encode the transmission;
- how reliable should be the connection.

The fact that a communication library is being used, greatly simplifies some of the aspects of the protocol. For instance, PeerJS by default encodes all messages in binary be it a string or a JavaScript object. The gaming context also emphasizes performance over reliability, that is, if a message about the current position of the trackball is lost, nothing catastrophic will happen.

The question that remains, though, is what kind of data should be sent from the controller to the game and how should a message be formatted. At this point in development, the system needs only two types of messages, one about the trackball position changes and one about the changes in the state of a button. These to types can be easily differentiated by a *type* field in the top level object of the message. In addition, a small number of message types does not enforce a deeply nested structure. Listings 3.17 and 3.18 represent examples of messages for trackball position change and a button event respectively:

```
{
  "type": "trackball",
  "x": 45,
  "y": -82
}
```

Listing 3.17 – Trackball Message

```
{
  "type": "button_pressed",
  "name": "a"
}
```

Listing 3.18 – Button Message

This protocol is by far not the best and the most efficient, but it works fine for the current state of the system. At the same time, in case of further development, it is certain that the message format will change drastically.

3.4 Implementation Conclusions

This chapter of the thesis gave an overview of the tools and techniques used build all the parts of the system. It presented the Phaser framework and a typical structure of a game that uses this framework. The sections about the controller explained the motivation behind its components and control elements. The final sections on peer-to-peer communication described the procedures that where employed in order to establish a communication channel between the controller and the main game. It also provided a brief description of the protocol used to exchange information and control messages between the parties.

Overall, the technical side of the project is relatively straight forward and does not contain scientific breakthroughs nor great innovations. On the other hand, it represents an attempt to realize an already existing concept and identify various pattern that can be reused in other projects on a greater and more innovative scale.

4 Economic Analysis

4.1 Project Description

The focus point of this thesis is the development of a toolkit that would enable software engineers to create with ease web-based applications that feature real-time communication with mobile devices and leverage all the sensors and feedback elements that a modern smart-phone can deliver. In the economic context, however, it is not enough to develop a framework, it is also crucial to have a demonstration application that implements logic which uses this framework, so that it can gain public attention and obtain necessary financial investments to support its further development. The 'Snowfight' game is a wonderful way to illustrate the power of real-time peer-to-peer interaction between a desktop browser and a browser of a mobile device. It can also effectively showcase the majority of the use-cases of the framework.

4.2 Project Time Schedule

In order to be successful, a project needs to have an effective development methodology. Presently, *Agile Software Development* gains popularity and proves to be a flexible and robust way to create software. This methodology implies an iterative and adaptive process of development and this in turn influences the time table that has to be established.

4.2.1 Objective Determination

The main objective of the project can be derived from its motivation. The project should illustrate at its best the features of the set of technologies described in the thesis, mainly those of real-time communication. This is to be done through the development of a computer game called 'Snowfight'. The game represents an isometric multiplayer arcade in which participants can throw snowballs in each other and score points. The players should control their characters via their smartphones, specifically through a web page rendered in the browser of the phone which represents a some kind of game controller. The gameplay should feel very responsive as if the player uses a wired gamepad.

Besides showing off technical features of the framework, the game should appeal to potential players that would use it as an entertainment asset at parties or team-building events. It should be engaging and well balanced as to provide casual, yet challenging, and maybe even addictive, gameplay. The controller part of the system be supported by most types of mobile devices.

The game is to be analyzed from the economic perspective so that it is clear what amount of resources is necessary to develop, deploy and maintain such a project. All kinds of expenses should be taken into account, like expenses for different types of assets, indirect expenses and the amount of allocations for the salary fund. This information is to be used in order to determine the total product cost and its retail price if it was going to be sold.

4.2.2 SWOT Analysis

In order to get a better picture of the product and different paths of its development it is a great idea to analyze the project from the perspective of Strengths, Weaknesses, Opportunities and Threats, as it has proven to be an effective technique in project management.

Strengths

- Intuitive user interface;
- Multiplayer with up to 10 players;
- Doesn't require any special programs to be installed;
- Entertaining.

Weaknesses

- The technology is available only on mid to high-end devices;
- The game is somewhat bound by the limitations of the graphical framework;
- Only one game mode.

Opportunities

- Add several game modes;
- Improve graphics art;
- Use this project as a step-stone for another project.

Threats

- Lack of visibility on the Internet;
- Development of similar frameworks by other teams.

4.2.3 Time Schedule Establishment

Most of IT projects' lifetimes consist of 5 basic steps: planning, research, development, testing and deployment. These steps are in turn subdivided into smaller parts in order to make the whole process manageable in terms of tasks. Most of the time table will be allocated to the step of development and testing as these are the steps with the most workload. Planning and research by itself shouldn't take up too much time, as requirements usually evolve during development, also research is a thing that usually never stops. The development of the project can also be split in tasks that can be performed in parallel thus minimizing the overall time. Total duration of the project is computed using the equation (4.1).

$$D_T = D_F - D_S + T_R, \tag{4.1}$$

where D_T is the duration, D_F – the finish date, D_S – the start date and T_R – reserve time. Table 4.1 represents the first iteration of the project schedule. The following notations are used to improve readability: PM – project manager, D – developer, GD – graphics designer, SM – sales manager.

Table 4.1 – Time schedule

Nr	Activity Name	Duration (days)	People involved
1	Project concept definition	5	PM, GD, SM, D
2	Market analysis	10	PM, SM
3	Domain analysis	15	PM, D
4	Product requirements specification	5	PM, D
5	System modeling (UML)	10	D
6	Graphic design	10	GD
7	Framework development	20	D
8	Main game development	30	PM, GD, D
9	Validation of results	10	PM, GD, D, SM
10	Documentation	5	D
11	Deployment and testing	10	PM, D
12	Active marketing	15	SM
	Total time to finish the system	145	

4.3 Economic Motivation

This section aims to give a perspective of the project from the economic point of view. This includes the expenses and profits encountered as a result of the project's activity as well as the various strategies of commercializing. All the costs and prices are given in MDL (Moldavian lei) currency. Tangible and intangible assets, indirect expenses will also be taken into account. One important thing to mention is that the game is the fact that it is being developed as an open-source project and it's primary goal is to popularize the framework and toolkit underneath. Nevertheless, there are still opportunities to commercialize the game, especially on the rising wave of the term 'Gamification' which gained substantial popularity lately in various industries. Thus, the game in modified versions can be licensed to companies that would like to 'gamify' their internal processes. Another monetization model that might bring some profit and cover the development expenses is the so called 'freemium' pricing model. This implies that the game should be released for free with a limited set of features and the full package of features would be unlocked to people that purchase a subscription of some kind.

4.3.1 Asset Expense Evaluation

In order to compute the potential cost of the product it is necessary to evaluate the cost of various assets involved in the production of the project. There are tangible and intangible assets that impose costs for the project, the budgets for both types are exposed in tables 4.2 and 4.3.

Table 4.2 – Tangible asset expenses

Material	**Specification**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
Workstation	Apple Macbook Pro 13' Retina	33000	2	66000
Smartphone	Nexus 5 and iPhone 5s	9000	2	18000
WebHosting	Microsoft Azure VM	10000	1	10000
			Total	96000

Table 4.3 – Intangible asset expenses

Material	**Specification**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
License	Enterprise Architect Desktop Edition License	1900	1	1900
License	Sublime Text 3 License	1500	1	1500
License	Adobe Photoshop CC 2016	2000	1	2000
Domain Name	Domain name registration fee	500	1	500
			Total	4900

4.3.2 Expendable Materials Expenses

Besides the costs of assets, the development of the project implies some direct expenses like consumable materials for the office. Even though the amounts are not that big, is important to keep track such expenses as they tend to add up. These expenses are presented in the table 4.4.

Table 4.4 – Expendable materials expenses

Material	**Specification**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
Whiteboard	Universal Dry Erase Board	500	1	500
Paper	A4 - 500 sheet pack	60	2	120
Marker	Whiteboard marker	15	10	150
Pen	Blue pen	5	10	50
			Total	820

4.3.3 Salary Expenses

It is obvious that a major part of project expenses go for the salaries of the employees. This section describes the computations concerning salary funds. The starting point is the distribution of per day salaries that the employees are expected to receive. It looks like this: project manager - 400MDL, sales manager - 300 MDL, developer - 380 MDL, graphics designer - 350 MDL.

Table 4.5 – Salary expenses

Employee	**Work fund (days)**	**Salary per day (MDL)**	**Salary fund (MDL)**
Project Manager	80	400	32000
Developer	110	380	41800
Graphics Designer	55	350	19250
Sales Manager	40	300	12000
		Total	105050

Given the data in the table 4.5 it is necessary to compute the amounts of money that should be payed to social services fund and medical insurance fund. After this it is possible to sum everything up and obtain the total work expense amount.

This year the social service fund is approved to be 23%, therefore the salary expenses are computed according to the relation (4.2).

$$\begin{aligned} FS &= F_{re} \cdot T_{fs} \\ &= 105050 \cdot 0.23 \\ &= 24161.50, \end{aligned} \tag{4.2}$$

where FS is the salary expense, F_{re} is the salary expense fund and T_{fs} is the social service tax approved each year. The medical insurance fund is computed as

$$\begin{aligned} MI &= F_{re} \cdot T_{mi} \\ &= 105050 \cdot 0.045 \\ &= 4727.25, \end{aligned} \tag{4.3}$$

where T_{mi} is the mandatory medical insurance tax approved each year by law of medical insurance and this year it is 4.5%.

The total work expense fund can now be computed, given the amounts of social service and medical insurance taxes.

$$\begin{aligned} WEF &= F_{re} + FS + MI \\ &= 105050 + 24161.50 + 4727.25 \\ &= 133928.75, \end{aligned} \tag{4.4}$$

where WEF is the work expense fund, FS is the social fund and MI is the medical insurance fund. In that way the total work expense fund was computed.

4.3.4 Individual Person Salary

Along with total work expense fund, it is necessary to compute the annual salary for the developer. Considering that the developer has a salary of 380 MDL per day and there are totally 250 working days in a year, so the gross salary (GS) in MDL that the developer gets is:

$$GS = 380 \cdot 250 = 95000, \tag{4.5}$$

Social fund and medical insurance taxes have the following values in MDL:

$$\begin{aligned} SF &= 95000 \cdot 0.06 = 5700 \\ MIF &= 95000 \cdot 0.045 = 4725 \end{aligned} \tag{4.6}$$

To compute the income tax it is necessary to now the amount of taxable salary:

$$\begin{aligned} TS &= GS - SF - MIF - PE \\ &= 95000 - 5700 - 4725 - 10128 \\ &= 74447, \end{aligned} \tag{4.7}$$

where TS is the taxable salary, GS – gross salary, SF – social fund, PE – personal exemption, which this year is approved to be 10128.

The last but not the least thing to be computed is the total income tax, which is 7% for income under 29640 MDL and 18% for income over 29640 MDL:

$$\begin{aligned} IT &= TS - ST \\ &= 29640 \cdot 0.07 + (74447 - 29640) \cdot 0.18 \\ &= 2074.80 + 8065.30 = 10140.10, \end{aligned} \tag{4.8}$$

where IT is the income tax, TS – the taxed salary and ST – the salary tax.

With all this now it is possible to find out the value of the net income:

$$\begin{aligned} NS &= GS - IT - SF - MIF \\ &= 95000 - 10140.10 - 5700 - 4725 \\ &= 74434.90, \end{aligned} \tag{4.9}$$

where NS is the net salary allocated for the employee, GS – gross salary, IT – income tax, SF – social fund, MIF – medical insurance fund.

4.3.5 Indirect Expenses

Besides direct expenses described in the previous sections there are such things like electricity, internet traffic, water and other utilities that are used in the office. These expenses are called indirect and the table 4.6 shows in what amounts they can impact project costs.

Table 4.6 – Indirect expenses

Material	**Specification**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
Internet	Moldtelecom Subscription	200.00 per month	3	600
Transport	Public bus trip	3.00 per trip	110	330
Electricity	Union Fenosa	1.92 per kWh	250	480
			Total	1410

4.3.6 Wear and Depreciation

Another important part of economic analysis is the computation of wear and depreciation. It is a well known fact that any product decreases its value with time. Depression will be computed uniformly for the whole project duration, so that there are no accountancy issues. In other words, if a product is planned for 3 years, it should be divided into 3 uniform parts according to each year.

Straight line depreciation will be applied. Normally wear is computed regarding to the type of asset. The notebook and single-board computer are usable for a period of 3 years. Licenses will last for a single year. At first tangible and intangible assets are summed up and then the salvage costs of each of the items at the end of their period of use has to be subtracted:

$$\begin{aligned} TAV &= \sum (AC - SV) \\ &= (66000 - 46000) + (18000 - 9000) + (4900 - 0) \\ &= 33900, \end{aligned} \tag{4.10}$$

where TAV is the total assets value, AC – assets cost, SV – salvage value.

To obtain the yearly wear, the total asset value is divided by 3 years of use.

$$\begin{aligned} W_y &= TAV/T_{use} \\ &= 33900/3 \\ &= 11300, \end{aligned} \tag{4.11}$$

where W_y is the wear per year, TAV – total assets value, T_{use} – period of use. Relation (4.11) included tangible assets which will last for 3 years and intangible assets which last only one year. The initial value of assets in MDL was

$$\begin{aligned} W &= W_y/D_y \cdot T_p \\ &= 11300/365 \cdot 135 \\ &= 4489, \end{aligned} \tag{4.12}$$

4.3.7 Product Cost

With all the project expenses computed, it is easy to compute the product cost which includes direct and indirect expenses, salary expenses and wear expenses as shown in Table 4.7.

Table 4.7 – Total Product Cost

Expense type	**Sum (MDL)**	**Percentage (%)**
Direct expenses	820	0.7
Indirect expenses	1410	1.2
Salary expenses	105050	90.04
Intangible asset expenses	4900	4.19
Asset wear expenses	4489	3.84
Total product cost	**116669**	**100**

4.4 Economic Indicators and Results

When it comes to selling the product, it is sometimes difficult to decide what price to set for each sold copy or unit. In general there are two main strategies that can be applied: sell less with a high price or sell more with a lower price. For the purpose of this thesis it is assumed that the expected profit is 15% of the total product cost and the number of sold copies is estimated at 500.

$$\begin{aligned} GP &= C_{total}/N_{cs} + P_p \\ &= 116669/500 \cdot 1.15 \\ &= 268.34, \end{aligned} \tag{4.13}$$

where GP is the gross price, C_{total} – total product cost, N_{cs} – number of copies sold, P_p – chosen

profit percentage. This is not the price of the end product, since it is necessary to add sales tax (VAT), which represents 20% and is added to the gross price.

$$\begin{aligned} P_{sale} &= GP + TX_{sales} \\ &= 268.34 \cdot 1.20 \\ &= 322.00, \end{aligned} \tag{4.14}$$

where P_{sale} is the sale prices including VAT, GP – gross price, TX_{sales} – sales tax. The net income is computed by multiplying gross price and the number of expected copies to be sold, which will be

$$\begin{aligned} I_{net} &= GP \cdot N_{cs} \\ &= 268.34 \cdot 500 \\ &= 134170, \end{aligned} \tag{4.15}$$

where I_{net} is the net income, GP – gross price, N_{cs} – number of copies sold. Moreover it is necessary to compute the gross and net profit. The indicators are GPr – gross profit and NPr – net profit.

$$\begin{aligned} GPr &= I_{net} - C_{production} \\ &= 134170 - 116669 \\ &= 17501 \\ NPr &= GPr - 12\% \\ &= 17501 \cdot 0.88 \\ &= 15400.88, \end{aligned} \tag{4.16}$$

where I_{net} is the net income, $C_{production}$ – cost of production. The profitability indicators are C_{profit} – cost profitability, S_{profit} – sales profitability computed as percentage.

$$\begin{aligned} C_{profit} &= GPr/C_{production} \cdot 100\% \\ &= 17501/116669 \cdot 100\% \\ &= 15.0\% \\ S_{profit} &= GPr/I_{net} \cdot 100\% \\ &= 17501/134170 \cdot 100\% \\ &= 13.0\%. \end{aligned} \tag{4.17}$$

4.5 Marketing Plan

After the product has come to the end of the production process, it needs to be sold to clients. This, however, is not as simple a task as it sounds. In order to find the right audience and choose the right way to offer the product, a series of activities should be performed. This set of activities is called *marketing* and is here to guide the flow of products from the producer to the clients. Marketing

is a system of economical activities about price setting, promotion and distribution of products and services to satisfy current and potential consumers requests. Marketing is the science and art of exploring, creating, and delivering value to satisfy the needs of a target market at a profit.

Marketing is described as having the following functions:

- Analysis of external environment;
- Analysis of consumer behavior;
- Development of product;
- Development of distribution;
- Development of promotion;
- Price setting;
- Social responsibility;
- Management marketing.

The game that is at the focal point of this thesis might be a hard thing to commercialize, but there are some ways in which it could bring profit. In order to identify the direction in which a product development path should lay, it is necessary to perform an extensive market research. Even though market research should be present to some extent in the whole development cycle, its initial stage is expected to provide a starting reference point for the product.

Market research can be separated in the stages outlined below:

- Identifying the problem;
- Developing program of research and gathering;
- Establishing specific information (internal, external);
- Establishing methods for collecting data;
- Performance of research;
- Information analysis, drawing conclusions.

The very beginning of the product evolution could be the most expensive for a company launching a new product. The cost of development is quite high in this period while the market is still very small. In case the product finds success on the market, it usually experiences a stage of strong growth in sales and profits, this in turn triggers the activity of the company as now it can cover easily the development expenses as well as invest in promotional activity. This results in profit maximization and the realization of the full potential of the product. When the product matures, the main goal of the company is to maintain the product on the market as at this moment a lot of competitors are coming out of shadows. In such conditions product managers must come up with improvements and modifications to the product, that would give it a competitive advantage. After this, usually, comes the declining stage in which the market of the product shrinks as a result of market saturation and sales start to become smaller as well. At this point the product is most likely evolve into a completely different one.

4.6 Economic Conclusions

The commercialization of a product is definitely not an easy task and the 'Snowfight' game is no exception to this observation. Moreover, the game itself is intended as an illustration of another product, the communication framework and holds a rather small value by itself. On the other hand it can serve as a gateway into a market of such kind of applications.

This section has exposed an extensive economic analysis of the game as a standalone product. It gave a perspective on the amounts of potential expenses that might be involved in the production process of the game as well as an estimate of how well the product should be sold in order to cover its costs and, on top of that, bring profit.

As a result of this analysis it became clear that in order to perform such a project in a commercial environment it is crucial that an exhaustive market research is performed in order to find the right niche for such a product and protect the company from failure. Still, at this stage it can be observed that 'Snowfight' might be suitable for the entertainment business and if it cannot bring money, it sure can bring fun.

Conclusions

Now that it can be said that computers are a vital part in human lives, it is important to keep in mind that this must not be at the expense of other things like communication and interaction among humans. Technology is often viewed as having a negative influence on society, but it sure can be used for the opposite as it depends solely on people themselves what technology can, and what it cannot do. This thesis represents the result of an effort to boost human interaction through the use of modern web technology.

'Snowfight' is a web-based multiplayer arcade that employs real-time communication to enable players to use their mobile phones as game controllers. It can be used at parties and team-building events in order to entertain and bring people together in a common activity. The game in itself is a proof-of-concept application designed to illustrate the use of a toolkit assembled as the main outcome of the thesis. The obtained toolkit is a collection of libraries and technologies suitable for this kind of tasks. By the end of the project the toolkit contained the following items:

PeerJS - a communication library which is a wrapper for the WebRTC technology;
HammerJS - a framework for handling touch screen gestures;
Utility code - a jQuery plug-in for creating the trackball control element.

During development it could be observed that real-time communication technology in the web context is still in active development as there where cases when certain parts of the specification were not supported by some browsers, moreover, even specifications are still available only as drafts. This results in a very high probability of breaking changes in future versions of the specs, that is, a lot of existing code will have to be rewritten in most cases in order to support new features, and it is likely that the relevant components of the aforementioned toolkit will be replaced.

At the same time, such a development toolkit has a great potential of evolution, especially when the specifications concerning RTC technologies will be more or less standardized. It can be transformed into a full-fledged framework that would give developers the ability to use in their games a wide range of template control elements for different purposes and an expressive API for bidirectional communication between the game and the controller. In the future it could also have support for motion-detection through the smart-phone's accelerometer and gyroscope, which would open new opportunities and ideas for game designers.

As for the 'Snowfight' game itself, it could be further refined and improved in terms of gameplay balance. It would also greatly benefit from a total visual refactoring as at the moment it uses stock spritesheets and on custom textures whatsoever. Such changes would appeal to potential users and might stimulate the development of the framework as well as a set of completely new games.

References

[1] Nintendo Wiki. *Wii Remote*. 2013. URL: http://nintendo.wikia.com/wiki/Wii_Remote.

[2] S. Salsano G. Wilkins S. Loreto P. Saint-Andre. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. 2011. URL: https://tools.ietf.org/html/rfc6202.

[3] A. Melnikov I. Fette. *The WebSocket Protocol*. 2011. URL: https://tools.ietf.org/html/rfc6455.

[4] Google Chrome team. *WebRTC Main Page*. 2013. URL: https://webrtc.org/.

[5] Sam Dutton. *Getting Started with WebRTC*. 2012. URL: http://www.html5rocks.com/en/tutorials/webrtc/basics/.

[6] Sam Dutton. *WebRTC in the real world: STUN, TURN and signaling*. 2013. URL: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/.

[7] C. Jennings J. Uberti. *Javascript Session Establishment Protocol*. 2013. URL: http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03.

[8] Google. *Chrome Experiments*. 2016. URL: https://www.chromeexperiments.com/.

[9] Lucasfilm Ltd. *Lightsaber Escape*. 2015. URL: https://lightsaber.withgoogle.com/.

[10] Google. *Super Sync Sports*. 2015. URL: https://www.chrome.com/supersyncsports/.

[11] Photon Storm Ltd. *Phaser.js*. 2015. URL: http://phaser.io/.

[12] Ralph Johnson John Vlissides Erich Gamma Richard Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994.

[13] Robert Nystrom. *Game Programming Patterns*. 2014. URL: http://gameprogrammingpatterns.com/.

[14] Jorik Tangelder Alexander Schmitz Chris Thoburn. *Hammer.js*. 2012. URL: http://hammerjs.github.io/.

[15] Eric Zhang Michelle Bu. *Peer.js*. 2012. URL: http://peerjs.com/.