FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# ASSIGNMENT OF BACHELOR’S THESIS

| | |
|---|---|
| **Title:** | ArtilEcho - a strategy game for blind people |
| **Student:** | Tomáš Jozífek |
| **Supervisor:** | Ing. Filip Křikava, Ph.D. |
| **Study Programme:** | Informatics |
| **Study Branch:** | Web and Software Engineering |
| **Department:** | Department of Software Engineering |
| **Validity:** | Until the end of summer semester 2019/20 |

## Instructions

Design and implement Artillery for blind users. Artillery is a classical turn-based strategy game in which tanks fights each other in combat simulation, a theme relating to one the origin uses of computer for calculating the trajectories of rockets and other military-based calculations.
The game should support multiplayer over a network with a user interface suitable for blind users.
Analyze approaches for the game user interface design targeting visually impaired people.
Implement the selected approach including proper documentation and testing.
Evaluate the game with the target audience.

## References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague December 18, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

# ArtilEcho – a strategy game for blind people

***Tomáš Jozífek***

Katedra softwarového inženýrství
Supervisor: Ing. Filip Křikava, Ph.D.

May 15, 2019

# Acknowledgements

Many thanks to my friend Lukáš Tyrychtr for introducing me to Python programming and helping me whenever I got stuck. To my brother Martin, thanks to whom I got interested in audio games in the first place. To Jakub Blažek for sharing his broad knowledge of this topic and my supervisor Filip Křikava for nudging me towards sensible solutions.

# Declaration

## Citation of this thesis

Jozífek, Tomáš. *ArtilEcho – a strategy game for blind people.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

# Abstrakt

Videohry jsou velmi populární forma zábavy, jenž díky použití grafiky není bohužel přístupná lidem s těžkým zrakovým postižením. Naproti tomu audiohry pro interakci používají hlasovou syntézu se zvuky, a tak umožňují hraní počítačových her nevidomým. Tato práce se zaměřuje na tvorbu tahové audiohry, konrétně hry typu artilerie pro více hráčů podobné hře ShellShock Live. Práce také poskytuje přehled o audiohrách a o jejich návrhu.

**Klíčová slova** audiohra, artilerie, implementace audiohry, nevidomým přístupné hry, zvuková odezva, Python

# Abstract

Video games are an extremely popular form of entertainment, but cannot be enjoyed by people with severe visual imparity since they use computer graphics as the primary source of interaction. Audio games, on the other hand, rely on speech synthesis and sounds making it possible for even fully blind people to play computer games. This thesis focuses on the creation of a turn-based multiplayer artillery-type audio game similar to ShellShock Live. It also provides a general overview of audio game design.

**Keywords** audio game, artillery, audio game implementation, blind friendly games, accessible computer gaming, audio feedback, Python

# Contents

# List of Figures

# Introduction

Computer games are extremely popular. Through distribution networks like Steam[1], Google Play Store or Apple App Store, they are more accessible than ever before. According to Newzoo [1] market researchers, the gaming industry reached a new all-time high with over 2.3 billion active players and an estimated revenue of $137.9 billion. Unfortunately hardly any game contributing to those numbers can be played by people with severe visual imparity. With a relatively low number (footnote) of people suffering from vision loss, the industry is not incentivized to make the games blind-friendly. Instead, visually impaired have to rely on largely amateur-developed audio games for the same kind of entertainment, missing many of the ideas and concepts from video games because of that.

An audio-game is a game that uses sound as its primary means of feedback to the player. The commonly used methods for this are speech synthesis, sounds and tone generation. From which sounds and tone generation can pass a lot of non-complex information in an instant compared to speech synthesis, which can describe almost anything, but requires much more time to do so.

I come from a family with many members visually impaired. As such, we know many more with this kind of disability. Growing up, I played many video and audio games and wondered which game mechanics could be adapted to the audio form. During the time I was learning to code, I came across an artillery game named ShellShock Live[2]. After figuring out how the game could be converted and confirming with my friends there was nothing alike available, in 2012 I created the first prototype. Since then, no one made anything similar, so I chose to undertake this project again as part of this bachelor's thesis with the intention of releasing the finished game.

ShellShock Live is a turn-based multiplayer game. At the beginning, each player gets a tank and chooses or is assigned a side on which they will fight.

[1]`https://store.steampowered.com/`

[2]`https://www.shellshocklive.com/flash/`

The game is two dimensional. The X axis is assigned to width while the Y is height, meaning that the player sees everything from the side as a cross-section of the terrain with tanks placed on top. When it is their turn, the player can control their tank's movement, gun angle and the power with which the gun will shoot. The goal is to score more points than the enemy team. Points are assigned to the team whose player manages to hit one of the enemy tanks, with the amount given depending on the accuracy and the type of ammunition used.

To port such a game into an audio form, I will

– research ways in which the game can provide coherent feedback exclusively through sound,
– implement the game using this information,
– document and test the implementation, and
– evaluate the game with the target audience.

Chapter 1

# Audio Games

## 1.1 Overview

As mentioned, audio games communicate information to their players through sound. Some feature an accessible GUI (graphical user interface) while other rely purely on sounds. Those relying on accessible GUI have an advantage in the form of pre-made components for entering and displaying information. The other can better customize their feel with custom voices and sounds. Their complexity ranges from a simple button masher to a strategy game requiring a player's full concentration to succeed.

## 1.2 History

I find the history of audio games quite interesting, especially in the Czech Republic where some blind used a PC before the general public did.

### 1.2.1 The Beginning

The first audio game I was able to find was named Touch Me and was made by Atari in 1974[3]. The objective was to remember a constantly growing sequence of buttons. The machine would play the sequence, represented by sounds and lights and the player was supposed to repeat it. This game was housed in an arcade cabinet and although not very popular, its portable clone Simon was.[4] Both games and their subsequent variations could be enjoyed by almost everyone as they opted out of using only the visuals as feedback.

Although playable by the blind, Touch Me was never intended to be, as well as any of the text-based adventure games, which thanks to the advent of

[3]`https://www.arcade-museum.com/game_detail.php?letter=T&game_id=12694`

[4]`https://web.archive.org/web/20060527173859/http://www.hasbro.com/simon/pl/page.newscelebrate/dn/default.cfm`

speech synthesis in the eighties were playable. So when the industry moved away from text and embraced graphics, this natural compatibility faded away.

Everything up to this point was inaccessible to blind people in Eastern Europe that was cut off by the cold war. Apart from a few who managed to import machines like VersaBraille[5], the most technologically advanced thing used by the blind in the Czech Republic before the regime changed were portable cassette recorders and players.

### 1.2.2 First Computer for Blind in the Czech Republic

After the iron curtain fell, however, the blind got their first computer. But not just a computer, a portable one. The government even subsidized the purchase through the Social Security Administration so that people could afford it. This was the computer to have for almost a decade, as the most active users provided support for the less invested ones.

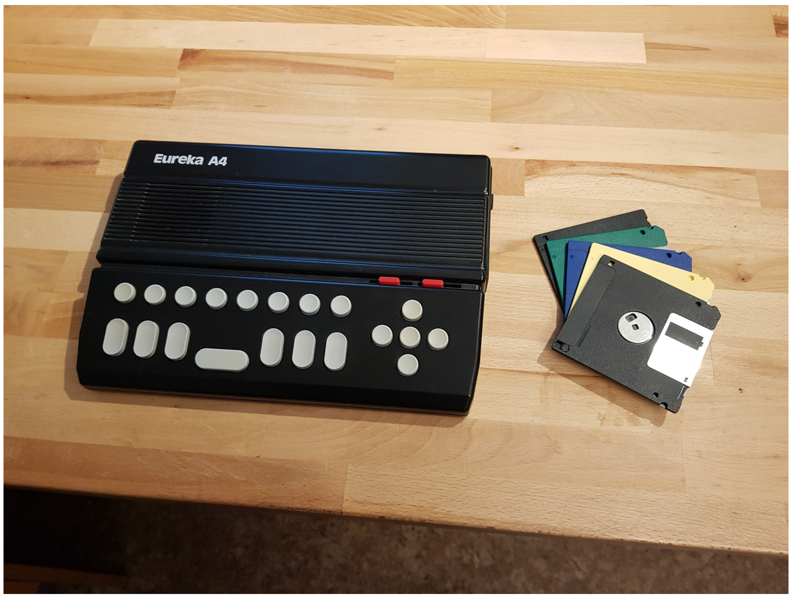


Figure 1.1: Eureka A4

Named Eureka A4[6] this 8-bit computer was made to be a personal secretary. It had no graphical output, only a speech synthesizer, and the keyboard consisted of only 20 buttons — seven Perkins style ones for typing, eight function keys, four cursor keys, and a shift key. The rest of the IO (input output) consisted of two RJ-11 ports for dial-up, a parallel port for printing, a DIN connector for an external keyboard among other things and a floppy

[5]https://victoriancollections.net.au/items/58d08b1fd0d0103314f1e328
[6]https://www.sensorytools.com/eureka.htm

disk drive, which served as the only long term storage option, as the computer did not let the user into its internal memory. The features included a text editor, date and time-related functions, calculator, a music editor that supported polyphony of up to four voices, telephone book, the dial-up related suite, various disk functions and an interpreter of Eureka flavored Basic. This interpreter allowed the community to develop all kinds of software, from social support amount calculators, through timetable applications to games.

### 1.2.3 Games on the Eureka A4

Audio games as the name implies, rely heavily on the ability of a system to produce sound. The games here were constrained by a mono sound output, an inability to conveniently play back prerecorded sounds, and I have never heard its tone generator change volume. The kinds of games available included classic text adventures, math, memory, and other quizzes and only a few that required real-time user input.[7]

One of which was the Hockey game. First, it could explain the rules and controls, then it would ask for the number of shots they want to catch. Next, it would play a random sound animation representing the approach followed by four different tone and noise sequences corresponding to a puck coming from above or the left, center or the right. The player had to react by pressing the right key, or key combination to catch the puck. Finally, the score would be announced, and the player asked if they want to play again.

The most interesting game to me was named Rockets. There were two tones. The player controlled one with their arrow keys, while the other was slowly lowering its frequency one half-tone at a time. The objective was to match that tone and press a space to shoot, which increased their score. Then the whole thing repeated, but with faster and faster descending tones aka rockets, until the player could not keep up.

### 1.2.4 Standardization

After Eureka A4 came MS-Dos running Aria[8] from the same company. It traded some of Eureka's features for its size, but due to the absence of an integrated code editor, the games had to be programmed externally, which resulted in a pack of only eight games, mostly ports from Eureka A4 being released here.

There were and are many more braille note-taking devices with varying feature sets before and after Eureka, especially in the west [2]. But I was not able to find reliable information about gaming on the earliest ones.

During the nineties, MS-Dos and then Windows gained popularity and as the platforms were used not only by the blind, the capabilities of systems

[7]List or many Czech Eureka A4 games: `http://ftp.braillnet.cz/e_games/`

[8]`https://www.sensorytools.com/aria.htm`

running them were vast. With the right hardware, all the IO (input and output) used by today's audio games was there. Screen reader, tone generation networking and in the later years stereo could all be used in conjunction to create games not much different from those played today.

### 1.2.5 Stagnation

Throughout the two-thousands, audio games flourished. The Blind Eye[9], explored the limits of binaural audio. It allowed its players to explore a virtual city simulating ambient sounds around them, although its replayability was limited, this game was quite unique especially for its time. The game Technoshock[10] was a Doom-inspired first-person shooter made in 2007, with a coherent story, enemies to shoot and puzzles to solve in each level. The player had a mission to free hostages from an advanced droid filled facility.

This goes to show how sophisticated games were. However, over the years the interest in the development of new and exciting titles seems to have faded. The main driving force behind new releases reduced to a few successful individuals and a handful of computer savvy blind gamers. There are new releases, but the games do not use the full potential of today's hardware. Some games are just renamed iterations of older ones, and some are straight copies with different sounds. There is also a very widely used framework called BGT (Blastbay Game Toolkit)[11] written for use with Angelscript[12], which significantly reduces the amount of coding experience required for one to make a game, but limits its users in various ways as the project is long abandoned. Games made using it are constrained to a single processor thread, non-binaural sound, the Windows operating system and the inbuilt networking implementation among other things.

## 1.3 Audio Game Genres

Today, there are many audio games and audiogame genres. This section provides a basic overview of some of them.

### 1.3.1 Copies of Real Card and Board Games

Even in the Eureka A4 era, people were remaking classical card and board games into the digital form. Among the older ones were the card game Prší, the Czech variant of Mau-Mau with multiplayer support[13], or games like Bat-

[9]https://audiogames.net/db.php?action=view&id=theblindeye
[10]http://www.tiflocomp.ru/games/ts/index.php?lang=en
[11]https://www.blastbay.com/bgt.php
[12]https://www.angelcode.com/angelscript/
[13]http://www.suscik.cz/program.php

tleship[14], Concentration[15], Backgammon[16], and many more. These were developed independently so many lacked multiplayer and were hard to make.

This changed in 2010 when the creators of an already popular monopoly adaptation added a new game and re-branded it to RS Games Client[17]. The client has a list of games, each with its lobby where people can chat and a list of rooms. Those could be created and joined by each player. This created a place where people could meet and play together without the hassle of creating a game server. I need to mention QuentinC's Playroom[18] as it is similar in concept but offers a different set of games.

The game rooms consist of standard UI elements like lists, tables, text input boxes, etcetera. The games are turn-based, and the action accompanying sounds usually do not convey any new information over what is described by the voice synthesizer. This makes the games easier to develop as every action can use already existing infrastructure for interaction with the player.

### 1.3.2 Racing games

Racing games like Top Speed[19] commonly use stereo sound to indicate, where on the track the player's car is. When the sound of the car's engine comes from the left for example, the car is on the left side of the road, and the player is expected to compensate. Each game also commonly features an announcement system, so the player knows what turn comes next. The problematic part is overtaking and how to distinguish cars in front of the player from those behind them.

### 1.3.3 Simulators

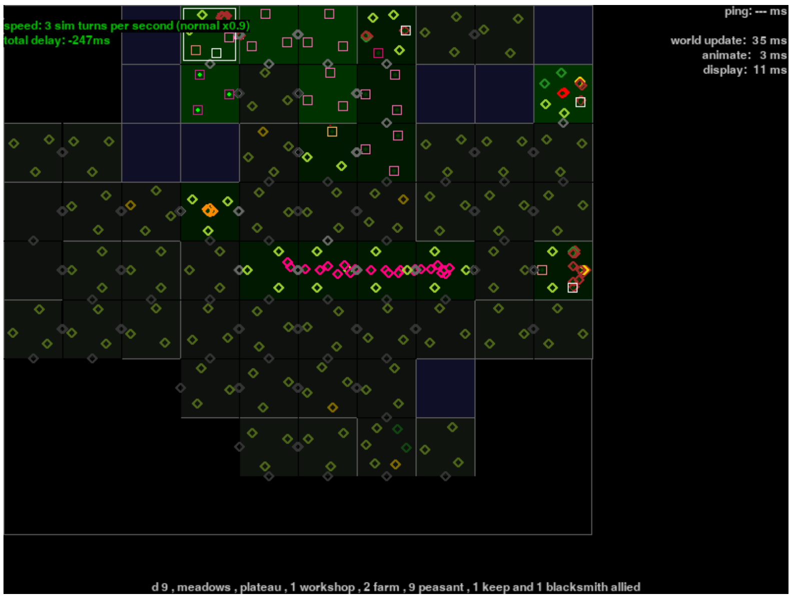
People sighted or blind, want to experience things they will not be able to in the real world. Like flying a plane, being a train or a subway conductor or a truck driver. These depend on detailed sound design in order to immerse their players in the virtual world. The challenge is to provide the necessary information for the games to be engaging, but not to interfere too much with their main objective. To simulate what the particular experience sounds like.

[14] `https://audiogames.net/db.php?action=view&id=Accessible%20Battleship`
[15] `https://audiogames.net/db.php?action=view&id=Wincon`
[16] `http://www.azabat.co.uk/downloads.html`
[17] `https://rsgames.org/downloads/`
[18] `https://www.qcsalon.net/`
[19] `http://www.playinginthedark.net/download_e.php`

### 1.3.4 Real Time Strategies

My favorite audio game SoundRTS[20] attempts to recreate the feel of Warcraft[21] a popular franchise created by Blizzard. The game map is 2D featuring a top-down view. It is a landscape with trees, mines, water, and pastures. Each player gets some starting resources like a few workers, and a base of some sort. Players use workers to gather resources and construct buildings like houses, farms, barracks and so on. Each building has a purpose, some can accept workers with resources, some can recruit/create new units, and some allow the player to recruit a new kind of unit or to upgrade other buildings. Each player has the same goal, to eliminate other players units and buildings completely.

To make the game accessible SoundRTS split the map into squares named the same way as a checkerboard. The player can select a square using arrow keys. When on a square, units, and buildings on it can be selected. The stereo is used to indicate where each object is relative to the center. The game also describes them using cardinal directions. There are many keyboard shortcuts for selecting different kinds of objects like buildings, workers and military units. Instead of single trees or large open spaces, the game contains forest objects for wood extraction and meadow objects which can be built on.


speed: 3 sim turns per second (normal x0.9)
total delay: -247ms

ping: --- ms

world update: 35 ms
animate: 3 ms
display: 11 ms

d 9 , meadows , plateau , 1 workshop , 2 farm , 9 peasant , 1 keep and 1 blacksmith allied

Figure 1.2: SoundRTS

The game includes graphics, which can be best described as rudimentary, consisting only of colored circles and squares. However, the inclusion

[20]http://jlpo.free.fr/soundrts/

[21]https://en.wikipedia.org/wiki/Warcraft

of graphics meant that siblings with different levels of visual imparity could enjoy the game together, which is rarely the case with any audio or video game.

### 1.3.5 First Person Shooters

Games of this genre give their players a virtual gun and place them on the battlefield. With objectives ranging from capture the flag, where the team able to fight their way through to the flag and hold the ground around it for the longest wins, to free for all, where killing the most enemies gets the player to the top of the scoreboard. The games utilize stereo much the same way a conventional first-person shooter would, the only change being the addition of sound markers to objects of interest, such as enemies, objectives, and items. Some games get rid of the third dimension altogether to help with orientation. Some games employ a radar which tells the player whether there is empty space, a wall or an enemy directly in front of them. Other games use ambient noise like a humming generator or a running television to aid in positional awareness. One very successful game not deviating too much from this formula is SWAMP[22]. It is set in a zombie apocalypse-themed wasteland with large maps to explore and cooperative missions to complete. Players can capture bases using in-game food and go scavenging together in search of new weapons, ammunition, armor, and other items. This is one of a few commercial successes among audio games. Its creator even released a head tracking device with support for his and other games, which makes SWAMP a virtual reality audio game, possibly even the first.

### 1.3.6 Text Adventures

Although mentioned in the history section, this genre persisted. From interactive audio stories where decisions were made by selecting a file with the correct number, adventure game engines like the Czech made Dousabel Sound Adventure[23], to the text-based hacking game Code 7 (a video game with accessibility features). These games provide something so many audio games lack. A story.

### 1.3.7 Video Games

Some blind people try and play video games designed for sighted players. The most popular are fighting games like Mortal Kombat where the player is able to orient by listening to the stereo sound effects emitted by characters. They can also gauge the distance between them and the enemy by throwing a projectile and observing how long it takes to impact.

[22]`https://www.kaldobsky.com/ssl/audiogames.php`
[23]`http://www.dousabel.cz/dsa/`

### 1.3.8 Browser Games

Although not video games, browser games were a popular form of entertainment in the two-thousands enjoyed by both blind and sighted as the games used text for most of their interaction. Often a player would take care of a virtual object of interest like a warrior or a village. Progress was made by giving commands like upgrade a building or go hunting which cost time. Players could interact with others in non-interactive battles where math together with randomness decided about success or failure. One example from this genre would be Legend of the Green Dragon[24]

[24]`http://lotgd.net/home.php?`

Chapter 2

# Design

As mentioned in the introduction I have chosen to convert the game named ShellShock Live.

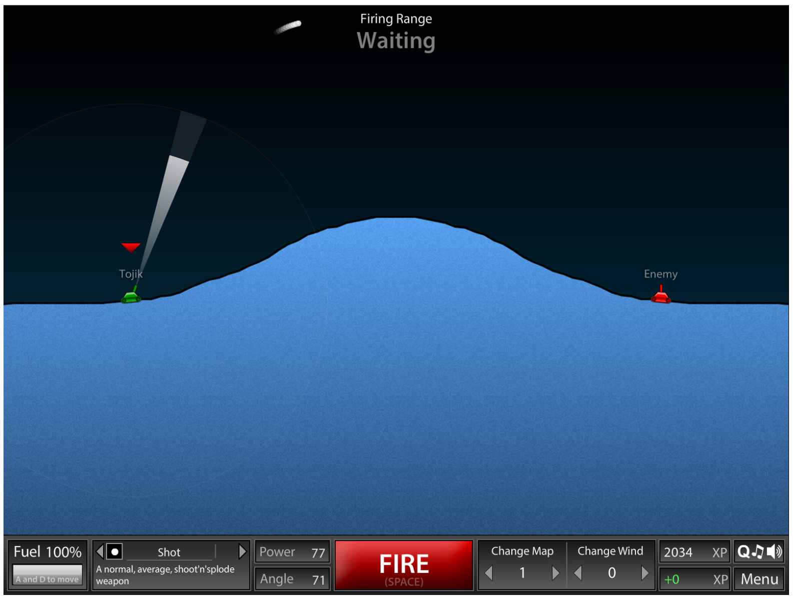


Figure 2.1: ShellShock Live

## 2.1 Making the Core Mechanics Accessible

Enabling players to see my game through their ears is what makes or breaks this project. I know the game concept works when well executed because I am essentially copying an existing game that managed to stay alive for over ten years. This section focuses on how do I translate information from graphics to sound.

### 2.1.1 The Terrain Exploration

The first thing I thought of when converting the game was the terrain. From the three tools available, prerecorded sounds will not help, because the map changes throughout the match. Using the speech synthesis surely is possible, but in my opinion not practical as reading out the map description with the info about where the craters are would be too much for the imagination of even the most advanced players. With this reasoning, I chose to use a tone generator.

Tone generator/oscillator a piece of software made to generate waveforms. The basic ones are sine, square, saw and triangle. Some oscillators have many settings, especially those used in the music industry. The settings I am most interested in are the wave shape, frequency, volume, and pan. Pan, also called stereo balance, controls how loud the left channel will be compared to the right. I will also need a noise generator.

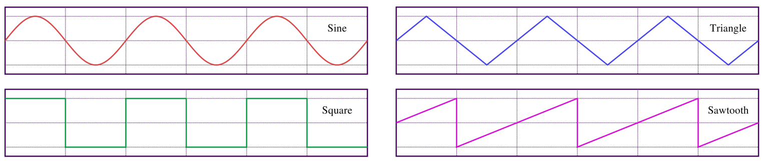


Figure 2.2: Waveform visualization [3]

The map is represented by an array of numbers and describing it is quite simple. I create a sine oscillator, set its pan, so the generated sound is coming mostly from the left channel and its frequency to match the leftmost terrain point value. Higher frequencies represent the higher points on the map and lower frequencies the lower ones. I do this in a loop for each point on the map, and each time I adjust the pan, so the tone moves from the left channel slowly to the right. For example, the tone representing the hilltop will be pitched quite high and reproduced by the left channel with the same amplitude as by the right one. I refer to this as the center of the perspective. Here the perspective is static. The player looks at the center of the map and like in the real world, objects on the left are heard from the left.

This is the backbone of the map exploration function. However, there are also tanks in the picture. To represent them I could change the waveform of the oscillator if a tank currently occupies the point. That is how I did it in the prototype. However, there is quite a bit more information presented to the sighted player by the original game. The player has an arrow above their tank to indicate who is currently playing, their nick is visible and the tanks are color coated so nobody can mistake their ally for an enemy.

Even though there are two perspectives mixed, the result does not seem to be confusing and provides almost every piece of information as the video game version.

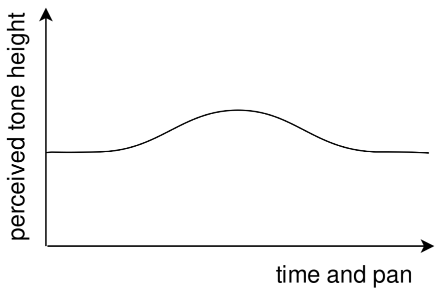


Figure 2.3: Oscillator properties

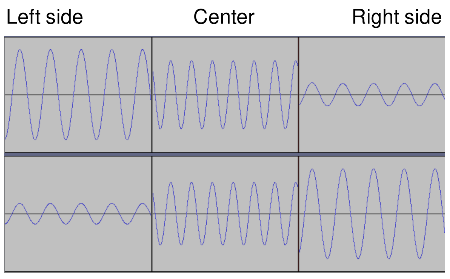


Figure 2.4: Sound curve

### 2.1.2 Tying it Together

Let us look at the rest of the accessibility features through a typical gameplay scenario, from launching the game to the winner announcement.

When the game launches, a window appears. That window contains a menu bar at the top, from which the user can select to join a room, create his own and a few other things like set their nick or view the credits window.

When the user wants to connect to a room, they select a menu item labeled *List rooms*. A dialog window with the list of currently available rooms opens. They can then go through the list which contains text information about the room's name, its capacity and the number of currently connected players. They can also use the map exploration function to determine what map is currently set in each room. Upon finding the desired one, they push the connect button.

This causes the room list dialog to close and a new room dialog to appear. As a room master the player can set various things like the map, the room size and the number of rounds, they can also kick players and start the game if everyone connected is ready. When connecting to the room, only the option to choose a team is available. Each player can also get the room status report which informs them if the game can be started, or why it cannot. When ready, the player presses the button labeled *Ready*.

After the game starts, each player is told, whose turn it is, if theirs, they hear a sound notification instead of an announcement. If unsure, the game offers a keyboard shortcut which repeats the information. At this point, I would expect the player to go through all player nicks and listen to the sounds they got by pressing the number row keys. Each plays the info sounds representing friendliness and such, their greeting sound and then their nickname is pronounced by the speech synthesis. After that, the player could use the map exploration function to hear where the tanks got placed. If unsure about the placement of any one, in particular, they can start the exploration just for a small area around each tank. Finally, they are expected to choose a weapon, take aim and fire, if not in the mood for such violence, the option to skip their turn is also available. After the shot finishes its business, the synthesizer announces how many points the person scored. That is added towards their team's score, which can be accessed by another keyboard shortcut.

After the game ends, a winning team is announced, along with both of the team's scores.

Chapter 3

# Implementation

## 3.1 Choosing Implementation Technologies

### 3.1.1 Programming Language

For the development of this game, I chose Python mainly because I know it the best, but also because I used it to create the first prototype back in 2012, so I knew all the necessary libraries providing an accessible GUI (Graphical User Interface), or bindings to speech engine APIs (Application programming interface) are available for it.

### 3.1.2 User Interface

I am using wxWidgets[25] for the GUI. It is an open source multiplatform GUI toolkit, using native UI elements on each supported OS (operating system). This allows screen readers native to each one to cooperate without any additional changes to any code. Moreover, wxWidgets is freely licensed allowing it to be used in any kind of project from open-source to a monetized proprietary one. WxWidgets is not written in Python, so I need a bindings library wxPython[26] to be able to use it.

### 3.1.3 Sound

For the sound feedback I am using a cross-platform proprietary audio engine FMOD Core developed by Firelight Technologies[27] specifically designed for use in games. Their other product the FMOD Studio is a game audio design suite, using the core as its backend. Together they are very popular used in many games like the Crysis series or Kingdom Come Deliverance, which might

[25]`https://www.wxwidgets.org/`
[26]`https://wxpython.org/`
[27]`https://fmod.com/`

have contributed to their quite benevolent licensing structure. This structure is split into tiers based on the game development budget. Unsurprisingly I fit in their lowest tier with game development budget under 500,000$, which grants me the permission to use all the features of their products for the small price of free, provided I include their logo on the splash screen and the text "Made with FMOD Studio by Firelight Technologies Pty Ltd." somewhere in the credits. Again, this library is not written in Python, so I use the work of my friend Lukáš Tyrychtr who made a bindings library Pyfmodex[28] allowing me to use it.

### 3.1.4 Model Classes

At first, I wrote my own serializable model classes. After discovering this solution is not viable, I moved to Google Protocol Buffers. It is a very widely used library for model class implementation, but its Python API deviates too much from its core coding style. That is why I chose Pydantic[29]. It provides serialization and type checking functionality, has a much friendlier API and an option to add a custom data validation.

### 3.1.5 Network Communication

During the idea phase of this project, I thought about making a web-based client. This is why I decided on WebSockets for network communication. There was one problem though, the WebSockets[30] library for Python is asynchronous, meaning it can be called only from asynchronous python functions. Since I was new to asynchronous programming, my friend Lukáš Tyrychtr helped me to develop a wrapper that converts this asynchronous API into a synchronous one, with some additional features. We called this library Python Websockets Communicator or Pywco for short.

### 3.1.6 Speech Synthesis API

Many speech synthesizers have an API for use by external software, but each one is a little different. That is why I chose accessible_output2[31] as it unifies mainly Windows-based speech synthesizer APIs into one. This means I lose access to some of the more specific features of some synthesizers, but gain confidence, that my game will be compatible with almost any players speech software of choice.

[28] `https://www.pydoc.io/pypi/pyfmodex-0.5.1/`
[29] `https://pydantic-docs.helpmanual.io/`
[30] `https://websockets.readthedocs.io/en/stable/intro.html`
[31] `https://web.archive.org/web/20190410084503/hg.q-continuum.net/`

### 3.1.7 Event Manager

I use Blinker[32] for event management. Its job is to pass information from Pywco, user interface and model services into the controller. For that, it employs signals which can be sent and connected to. These signals can be named, or distinguished by their instance address (every instance being unique). I use both types, named for Pywco, as I cannot pass instances over the internet and nameless for the rest of my application.

## 3.2 Architecture

The application is split into three parts, the code needed exclusively by the server, the client and the code shared between them. This common code consists of model classes, non-serializable data types, communication protocol info, and shared set of utilities like a self-destructing event handler or an animation/timer service.

The client only code contains:
- **model**: an instance containing the game data,
- **controllers**: handling each event, dictating what shall be done,
- **user interface**: communicating with the player,
- **speech and sound**: providing additional feedback to the player,
- **services and utilities**: map exploration, shot animation and other code that I was not able to put anywhere else.

The server only components are:
- **controllers**: again handling various events,
- **model**: another instance containing the server data,
- **model services**: manipulating data in the model,
- **other services**: for complex operations with different types of data.

I am taking inspiration from the game room clients described earlier. That means I chose to have one default server to which every client connects. Each room gets a new process running on a different port, which makes the default server more stable as in-game bugs are unlikely to bring it down. It should also help with scalability as in the future these instances could be spawned on an entirely different server.

First only the default server (in the code named `room_dispatcher`) and the client are connected. When the client wishes to create a room, a new process is spawned and given only the port number it is supposed to run on through the command line arguments. Immediately after starting, it connects to the default server and asks for the rest of its information, like its name and id. Upon receiving those, it tells the default server it is ready. The default

[32] `https://pythonhosted.org/blinker/`

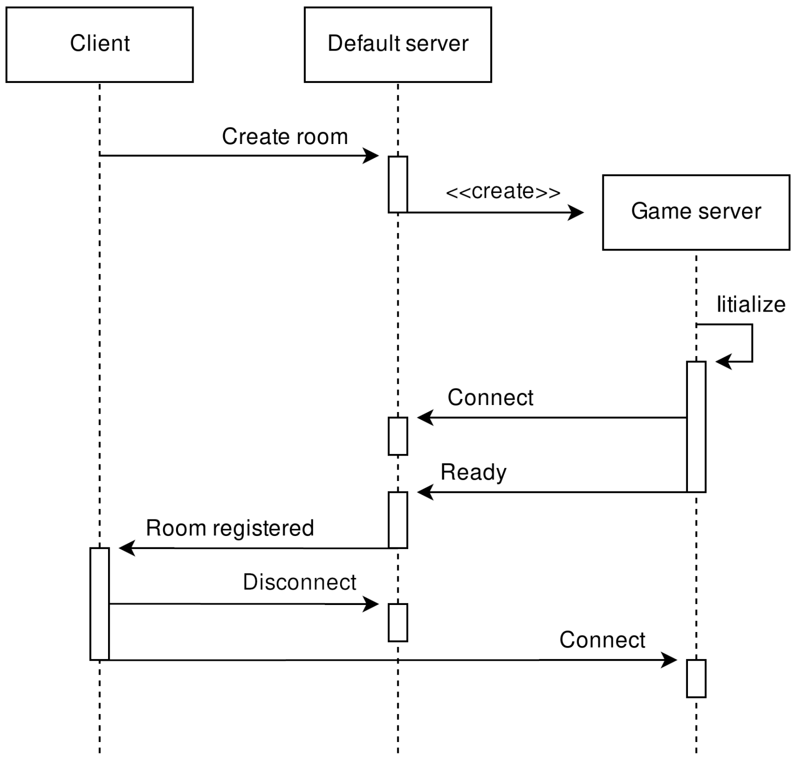


Figure 3.1: Spawning the game server

server then sends a message to the client informing them on which port and address the room was created, after which the client uses that information and connects. The code is not expecting a failure. If it occurs, usually no message is displayed. The only exception is, when the client connection fails, the client will be informed. If everything goes smoothly, at the and, there will be two connections, one between the default server and the room/game server and the other between the room/game server and the client. The former is used to update the room status like the number of players connected or the map selected and the latter obviously for transferring game data.

The client is only used for input and output, the validation and computing are done on the server side. A typical path for an event propagation looks like this:

1. client changes the number of rounds in the user interface,
2. user interface handles the event sending a blinker signal with the new value,
3. client controller handles the blinker signal and sends a command to the server,
4. server controller calls the model service to set the value,
5. model service validates the value, if OK, the value gets set and a deferred event instructing the server to update it on all clients is generated,

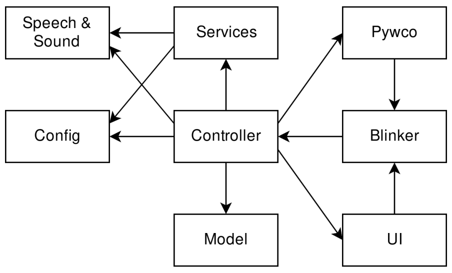


Figure 3.2: Client components

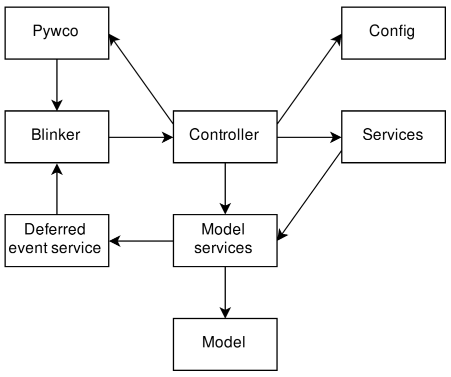


Figure 3.3: Server components

6. after the server controller finishes executing the handler, deferred event service is called to execute all queued events,
7. the queued event causes the server controller to be called and broadcast the new room game settings,
8. client controller receives the new settings and saves them directly to its model,
9. the UI is called to update the displayed data.

This introduces some latency problems. Because aiming waits for the server to validate whether the player is allowed to aim before playing the aim sound, Some redundant code would be necessary, to provide the client with a better experience.

I am using intermediary layers for almost any component, for example, I interface with the speech API through an execution queue, because the library essentially requires calls to originate from a single OS thread, the same as some UI components. These layers also allow me to implement some features like a speech buffer/history into the speech API one.

### 3.2.1 The Network Communication Protocol

Networking wrapper, Pywco, uses string values for commands. Those are supplied in the form of an enum class to its constructor. The instance then accepts any command that is present in said enum paired with any keyword arguments containing serialized data. If the transfer succeeds and if the other side has connected a handler to a blinker signal with the same name (the same enum value), the handler will be called along with the same keyword arguments. Because Pydantic produces a dictionary as its serialization output and accepts the same unpacked dictionary on the other side, transferring objects is quite simple.

```
# Protocol definition - common
from enum import Enum

class RoomCommands(Enum):
...
    UPDATE_ROOM = "Update room"
    UPDATE_PLAYER_LIST = "Update player list"
...

# Model class - common
from pydantic import BaseModel

class PlayerList(BaseModel):
    players: Dict[int, Player] = {}

# Server - sending new data to all clients
def broadcast_player_list(self, sender):
player_list: PlayerList = self.model.player_list
    self.room_pywco.broadcast(RoomCommands.UPDATE_PLAYER_LIST,
                              player_list=player_list.dict())

# Client - receiving updated data
def update_player_list(self, sender, player_list):
    player_list = PlayerList(**player_list)
    self.model.player_list = player_list
```

Listing 1: Example from implementation – object transmission

## 3.3 Unit Tests

At the time of writing, the implementation contains 146 unit tests for tank movement, game room setup, and math services. I am using the inbuilt Python library named unittest, which integrates well with the PyCharm[33], the integrated development environment I am using.

[33]https://www.jetbrains.com/pycharm/

Chapter 4

# Assessment

I intend to make the game presented in this thesis available to the community of blind and visually impaired. Before making a public release, I wanted to gather some early feedback. In this chapter, I describe the setup and the results of a short experiment I did for that purpose. To make the test come to fruition, I had to:

1. create a portable game binary,
2. set up a public server,
3. record a game tutorial explaining all the different sounds,
4. make the questionnaire, and finally
5. post a download link on AudioGames.net[34] forum, which is a popular site for downloading and discussing audio games.

## 4.1 Experiment Setup

I decided on a public test. The idea was to make the current version of the game public for a limited amount of time, asking each player to fill in a small form after testing.

### 4.1.1 Evaluation Form

I wanted to see whether the game is understandable to newcomers while being aware of their familiarity with audio games in general, as I do not consider my game to be easy to get into for people who never played one before. I would like for this project to make some amount of money that could be put towards server maintenance or commissioned music, but I was unsure which monetization model to use. Lastly, while personally wanting to develop the

[34]`https://AudioGames.net`

game further, I wanted a second opinion along with some inspiration for future features.

The questions were:

1. **How familiar are you with audio games?** (1 Not familiar – 5 Extremely familiar)
2. **Was the game fun?** (1 Not fun – 4 Very fun)
3. **Which game mechanic did not make sense?** (Options)
   - Map exploration
   - Soundpacks
   - Aiming
   - Shot flight
   - Other
4. **Should I continue development after graduation?** (1 No – 4 Yes)
5. **Would you pay for the finished game?** (Options)
   - Make it free, I swear I will donate.
   - Make it free, with features like custom soundpacks as a premium.
   - Make it paid, but no subscriptions!
6. **Feedback box** (Text)
7. **Feature suggestions?** (Text)

### 4.1.2 Results

The game was available on Saturday, May 4th from 17 pm to 1 am CEST (Central European Summer Time). In this time, around a dozen people tried playing the game and six filled in the form. The actual results are shown on 4.1 and 4.2. Overall the results are very encouraging as the game received on average three-point-sixty-six out of four points under the question about fun, so I think it has the potential to be entertaining. The majority of players were fairly familiar with audio games as expected from users of an audio gaming forum. The map exploration feature was marked as confusing while another person praised its simplicity, so no conclusion could be made. Fortunately, it was the only one marked as not making sense. The most voted for monetization model was the *free game with paid premium features.* Although winning only by one vote, given the ShellShock Live used the same model, I am planning on implementing it.

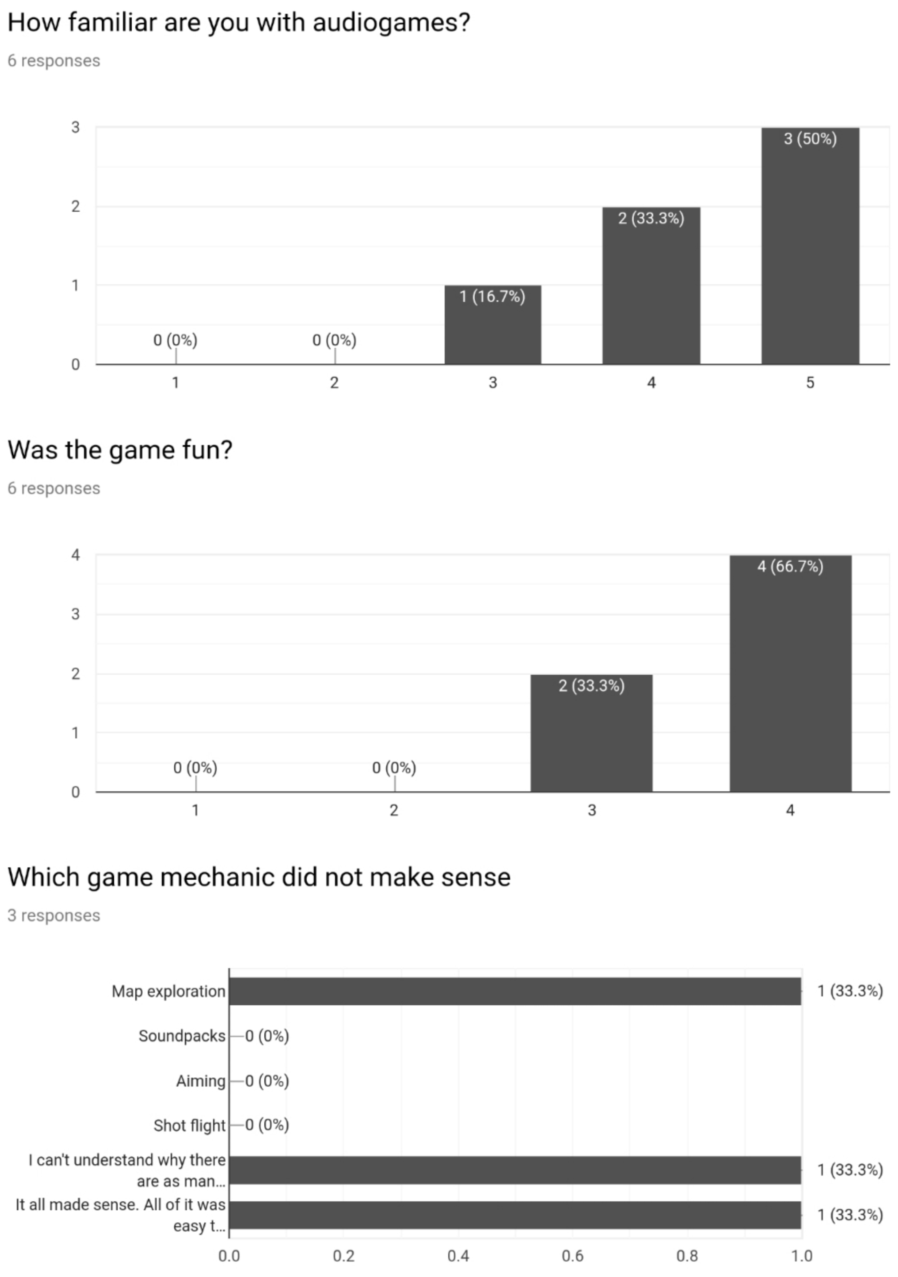
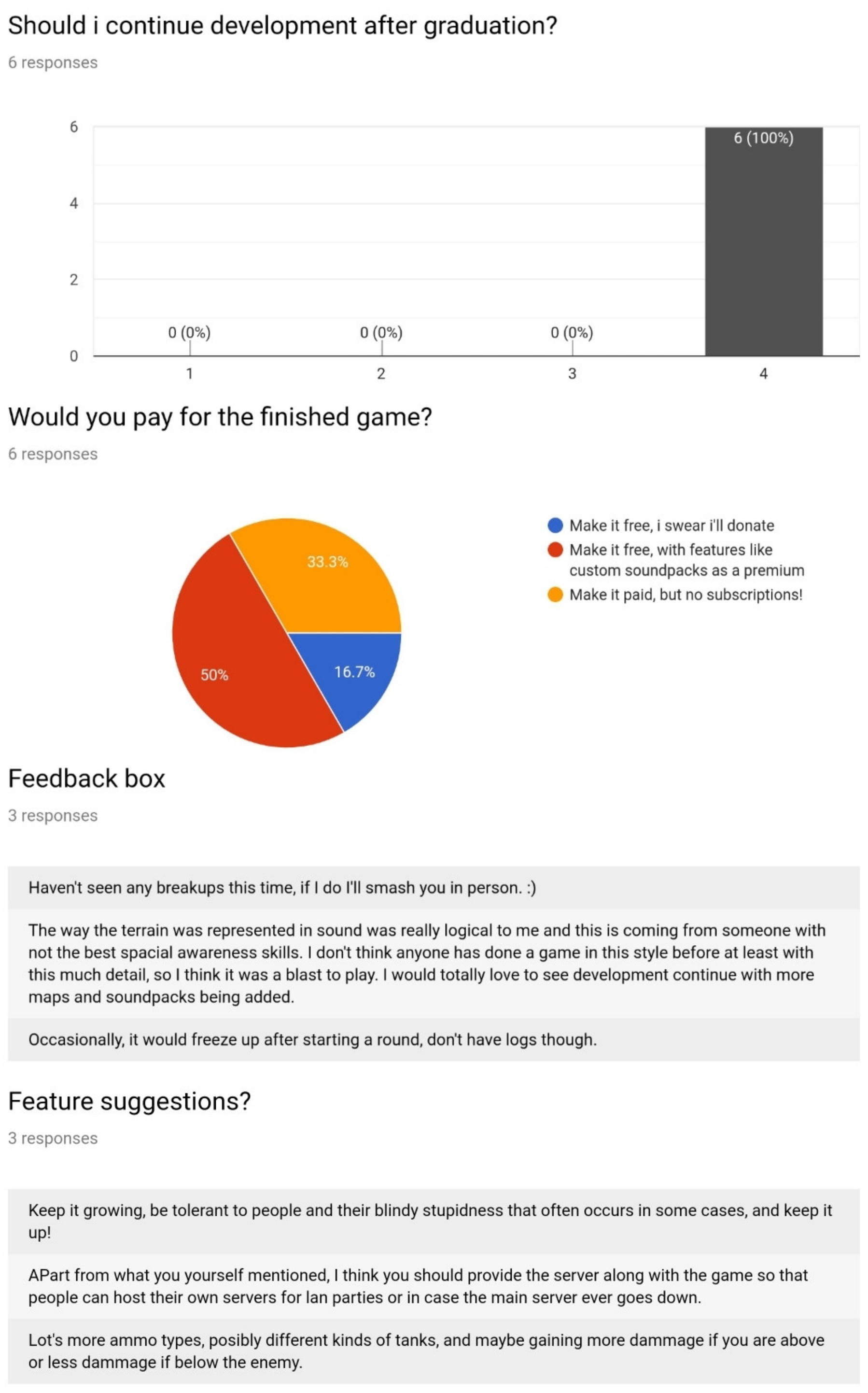


Figure 4.1: Evaluation form results – 1


6
4
2
0
0 (0%)
0 (0%)
0 (0%)
6 (100%)
1
2
3
4
33.3%
50%
16.7%
● Make it free, i swear i'll donate
● Make it free, with features like custom soundpacks as a premium
● Make it paid, but no subscriptions!

Should i continue development after graduation?

6 responses

Would you pay for the finished game?

6 responses

Feedback box

3 responses

Haven't seen any breakups this time, if I do I'll smash you in person. :)

The way the terrain was represented in sound was really logical to me and this is coming from someone with not the best spacial awareness skills. I don't think anyone has done a game in this style before at least with this much detail, so I think it was a blast to play. I would totally love to see development continue with more maps and soundpacks being added.

Occasionally, it would freeze up after starting a round, don't have logs though.

Feature suggestions?

3 responses

Keep it growing, be tolerant to people and their blindy stupidness that often occurs in some cases, and keep it up!

APart from what you yourself mentioned, I think you should provide the server along with the game so that people can host their own servers for lan parties or in case the main server ever goes down.

Lot's more ammo types, posibly different kinds of tanks, and maybe gaining more dammage if you are above or less dammage if below the enemy.

Figure 4.2: Evaluation form results – 2

# Conclusion

Unlike in video games where the interaction is based on the game providing graphical feedback to the player, audio games rely on speech synthesis and sounds. This allows people with severe visual imparity to enjoy playing computer games. This thesis presents the design and implementation of an audio game. Concretely, a port of an artillery-type video game to the audio form called ArillEcho.

I have researched existing audio games and their sound feedback to gain a better understanding of the topic, which helped me to successfully create ArtillEcho. It is an over-a-network multiplayer game in which teams of players compete to determine who can score more points by accurately shooting their enemies. The game can describe terrain, tank positions, shell flight and explosions without speech which is used for game statistics like a score or the number of rounds remaining. The server implementation is partly covered by unit-tests. The eight-hour-long beta-test of the game was announced on AudioGames.net forums, and about a dozen people tried playing it. Out of them, six filled in my questionnaire giving the game three point sixty-six out of four star rating.

In the future, I will focus on two things. First to improve the documentation, which in the current version is regrettably absent. Second, I would like to improve game stability and add features that will make the game even more fun to play. This includes password protected game rooms, chat, connection recovery and encryption, more weapons, maps, and better sound design.

# Bibliography

1. NEWZOO. Newzoo's 2018 Report: Insights Into the $137.9 Billion Global Games Market. In: [online]. 2018 [visited on 2019-05-08]. Available from: `https://newzoo.com/insights/articles/newzoos-2018-report-insights-into-the-137-9-billion-global-games-market/`.
2. GERVEN, Clara Van; TAYLOR, Anne. The Information Age Braille Technology Timeline. In: [online]. 2011 [visited on 2019-04-26]. Available from: `https://web.archive.org/web/20160327200657/https://nfb.org/Images/nfb/Publications/fr/fr28/fr280109.htm`.
3. OMEGATRON. Waveforms.svg. In: [online]. 2006 [visited on 2019-04-26]. Available from: `https://commons.wikimedia.org/wiki/File:Waveforms.svg#filehistory`.

Appendix A

# List of Used Abbreviations

| | |
|---|---|
| **GUI** | Graphical user interface |
| **OS** | Operating system |
| **API** | Application programming interface |
| **IDE** | Integrated development environment |
| **IO** | Input output |

Appendix **B**

# Contents of the Attached CD

```
readme.txt.................short guide on running the server and client
├── bin.............directory containing the executable version of the client
├── src
│   ├── implementation.......................................source code
│   └── thesis.........................................source files in LaTeX
└── doc...................................................text of this thesis
    └── thesis.pdf........................text of this thesis in PDF format
```