
## Honours Project - MHW225671

# FINAL REPORT

**2020-2021**

Department of Applied Computer Games (DACG)

**Submitted for the Degree of:**

## BSc Computer Games (Software Development)

| | |
|---|---|
| **Project Title:** | **Investigating the impact of integrating voice recognition technology on players' game experience in First Person Shooter games.** |
| **Name:** | **Dawid Kubiak** |
| **Matriculation Number:** | **S1717551** |
| **Project Supervisor:** | **Mario Soflano** |
| **Second Marker:** | |
| **Word Count:** | **10,464** |

(Word count excludes contents pages, figures, tables, references and Appendices)

**"Except where explicitly stated, all work in this report, is my own original work and has not been submitted elsewhere in fulfilment of the requirement of this or any other award"**

**Signed: Dawid Kubiak** **Date: 22/04/2021**


## Abstract

With the rapid development of voice recognition technology, new opportunities to feature the technology in various fields have been created. Nearly all of us have access to speech recognition, it is available in our phones, personal assistants such as Amazon's Alexa or Google's Home or even Microsoft's Word software. However, this technology does not have to be limited only to be used to make our lives easier, keep us safer or help disabled people in their everyday activities. It can also be introduced into our entertainment, specifically speaking – video games. This research paper reviews how such technology can be incorporated into First Person Shooter video games in order to improve players' experience and immerse them even more into the game's universe. A brief description of how games and input methods evolved to intensify players' gaming experience, what speech recognition technology is and how it works, how the games used for evaluation were created and what were the results of the experiment in terms of investigating the impact of integrating voice recognition technology on players' game experience in First Person Shooter games. In order to explore this subject, two mini games have been developed. Both feature an in-game ally that the players can communicate with, however one game features Microsoft's Cognitive Services Speech SDK which recognizes speech and the second game can only communicate with the ally using keyboard keys. The evaluation has been done using both, qualitative and quantitative data, using the methodologies described in the document and has resulted in a conclusion that, undoubtedly, the speech recognition technology can revolutionize the gaming market and provide players with an enhanced gaming experience.

## Acknowledgements

Throughout the research I have received a lot of support, expertise knowledge and assistance from my supervisor, Dr Mario Soflano, whom I would like to thank. I would also like to express my gratitude to all the Lecturers and colleagues I met during my studies at Glasgow Caledonian University for the wonderful four years I had.

# Table of Contents

## Table of Figures

# 1 Introduction

This chapter describes First Person Shooter genre, how games have been evolving in both, virtual and physical aspects, gives an overview of voice recognition technology, shows how such technology can be used in FPS games and specifies the objectives of the research.

## 1.1 First Person Shooter games

First-person shooter (FPS) games increased in popularity over the years, accounting for 'only' 11% of the video game sales in 2008 (Weber et al., 2009) and reaching almost 21% of game sales in United States in 2018 (Gough, 2020) losing only to action type games which reached around 27%. FSP is actually a genre of action video games that is played from the point of view of the central figure of the game's narrative. This type of games usually "*map the gamer's movements and provide a view of what an actual person would see and do*". (Technopedia, 2011) These games usually require the players to develop different strategies to quickly react to fast moving objects and adapt their behaviour to the constantly changing game environment.

## 1.2 Games evolvement

### 1.2.1 Virtual development

Games evolve along with the technology. And they need to evolve constantly, to keep the players engaged by new, different methods. The moment the engagement is neglected, the player will stop playing it. (Schoenau-Fog, 2011) Players are kept engaged throughout various means, the more diversified methods are used in the game, the wider the audience will be gained for the game. Video games usually have a start and end point, which a player can accomplish by completing specific objectives. Moreover, games can also include various secondary missions or collectibles to gather throughout the story. This will motivate the players to play again even after completing the main story line. People can also play games that focus mostly on exploration of the world, creation, modification, or destruction. The players are kept in front of the screen credit to the audio, story, graphics. Virtual Reality made a huge step forward in the direction of bringing the players 'into' the game, however, nowadays the games still tend to feel as if they reached a certain point and they only progress in visual aspects, focusing solely to give the gamers high end graphics.

### 1.2.2 Input innovations

Games' evolvement does not only take place in the virtual world as there are other, different ways of engaging the players – the way they communicate with the game also matter. And there has been some significant development in that area. Atari Home Pong console (1975), was just about rotating a single dial to slide player's paddle. Sony's PlayStation 1 (1994) innovative controller introduced the players to the world of triangle, square, circle and cross which helped change the way we play now. When taking Sony's controllers under the loop, it can be easily spotted how the input methods changed along the years, and it is not only by adding more controls on the game pad, such as Analog Controllers, but also how they interact with the players. With their DualShock pads, they have introduced vibrations, pressure-sensitive buttons, force feedback and touch pads. Although not very successful, but still worth to mention, Sony has also created PlayStation Move controller which was used for motion control. The above mechanisms are responsible for immersing the players into the game, allowing them to physically feel what is happening on the screen which plays a vital role in enhancing their experience.

## 1.3 Voice recognition technology

Voice recognition, or also called speech recognition, is a technology that recognizes spoken words. It started its growth back in 1976, when it was limited to around 1000 words. (Rouse, 2018) In order to use such a system, the only thing that is needed is a microphone. Currently, it is used in most of people's life by functioning as our virtual assistants – Amazon's Alexa, Apple's Siri or Google's Google Assistant. They understand commands such as to add an alarm, play a film on a TV, answer questions about the weather or change the temperature in your house. It can also help disabled people to work without the use of mouse or keyboard or people with dyslexia to write more fluently, accurately and quickly. (AbilityNet) Speech recognition is now widely available as it has recently grown mostly thanks to Artificial Intelligence and machine learning that helps them to understand the patterns created by user's speech.

## 1.4 New perception

By introducing speech recognition to games, we could open another, new way of engagement and interaction between the player and the game. Currently, when playing FPS games, only the player's sight and hearing is stimulated, and sometimes, if the player is using a game controller, the touch sense as well. The latter is caused by the controllers vibrating when, for example, the player is near an explosion. If another sense to that list was added, such as pronunciation, how could that impact players participation in the game? What feelings would accompany them, knowing that what they say, would have a visible influence on their game? Instead of pressing a key on their keyboard, they could literally say that they need air support on certain coordinates, call for a medic if they are running low on health or ask non-playable characters for ammunition supply if they need it. The integration with the game could be even more addictive for the players, making them feel as if they are genuinely a part of the whole game and in the middle of the battlefield. Therefore, this research will investigate the impact of integrating voice recognition technology on players' game experience in First-Person Shooter games. Mainly quantitative data will be collected through a questionnaire that will be given to two different groups after playing the created game. Moreover, qualitative data will also be gathered from the interactions with the voice recognition technology by examining voice data.

## 1.5 Objectives

In order to properly investigate the stated above hypothesis, the paper will dive into the details of voice recognition technology, how it works and how it can be used in a game. Moreover, two similar First-Person Shooter mini games will be developed. One of the games will additionally include the use of voice recognition technology, whereas the other one will only be played in a traditional way, using mouse and keyboard. Finally, the games will be tested by two different groups, and the gathered feedback will be analysed to determine how such speech recognition technologies, if at all, can impact the players' game experience.

# 2 Literature and Technology Review

The chapter gives a comprehensive review of First Person Shooter games' features, how players are currently immersed in the gameplay using both, virtual and physical methods. Moreover, a more in-depth definition of Speech Recognition technology is provided, with a focus on Microsoft's Cognitive Services and also including how it has been used so far in various games. Finally, the chapter gives an overview on Levenshtein distance algorithm which can be used along with speech recognition to determine most probable phrase the user spoke.

## 2.1 First Person Shooter Features

FPS games can either be played in a single player campaign or online multiplayer, where different players face each other. Single campaign generally provides the player with a story line to follow, making him, for example, a soldier in a tactical assault team, that needs to infiltrate the enemy's lines and execute a certain target. Online multiplayer modes typically include several different ways game modes, such as Team Deathmatch, Capture the Flag, Search and Destroy or, to a certain extent, quite new to the market, Battle Royale. Multiplayer modes usually include certain add-ons for the players when reaching specific goals, for example, killing three players without getting killed. Again, these vary by games, for Call of Duty Modern Warfare's the player can acquire a personal radar, UAV, care package, support helicopter, cruise missile and other. Once gained, the player is notified that a certain bonus is available. These killstreak rewards can then be activated using certain keys on the keyboard or a game controller. The controls trigger an animation of the player speaking a phrase through a walkie-talkie radio and the reward becomes active. Everything is available under a touch of a button. Looking at this from a point of view that the player basically uses a key to tell his character to say something, it does not seem to be much engaging and realistic or play any major role in immersing the player. Some of the special features are also available in the single campaign modes, at specific, scripted moments, where, for example, the player just needs to click on an area that is viewed through binoculars and the support team will perform an air strike. This leads to the same conclusion as previously described, although the games genuinely advanced in diverse terms, the player interactions are in most cases still limited to simple player inputs using devices such as mouse, keyboard or game controllers.

## 2.2 Player immersion methods

This section describes the virtual and physical methods current games or peripheral manufacturers use to improve players' gaming experience.

### 2.1.1 Virtual methods

As already stated before, current games immerse the players into the gameplay and enhance their experience by providing excellent sound effects, beautiful visuals and absorbing storylines. Sound effects are a primary factor for immersion, especially in games played from first-person perspective, by leading to number of emotions. (Nacke & Grimshaw, 2011) On the other hand, Gerling, Birk and Mandryk (2013), in their study, explained that better graphics result in better motivation, more immersion in the game and most importantly, in enjoyment. Furthermore, Kevin Murnane (2018) states that graphics and the gameplay experience interact in essential ways but cannot exist as separate entities, they need to bond in order to create a world that attracts the player. Finally, game's narrative is also a major factor in how players perceive a game. "*The advantage of video games is that, unlike with*

*other types of narrative, the player is part of the story*". (Suarez) Great narratives allow the players to feel involved in the game, be cheerful when succeeding or disturbed when failing. All of above concludes to the fact that the players do not just want to strike some keys on the keyboard. They want to be involved in the games physically, to perceive the games as real, hence this might be the reason why the world currently experiences prosperity in Virtual Reality (VR) headsets. (Vailshery 2021) Therefore, integrating voice recognition technology into a game, might positively impact on how the players perceive the game, as they would be involved in the gameplay even more, physically.

### 2.1.2 Physical methods

As mentioned previously, companies also try to immerse the players by improving the peripherals. For example, by adding a touch pad, used on PlayStation 4 controllers, which can be used as a way of navigating the map or inventory screen. Although this feature might not be considered successful and might not be used to its full potential, it is still a way of allowing the player to interact with the game in a different way. However, it is not just about adding new controls on top of the game pad, but also adding features that enhance the players' experience by allowing them to physically feel what is occurring in the virtual world. Vibrations, pressure-sensitive buttons, force feedback integrated into game controllers and gaming steering wheels, are mechanisms that are related to the term haptic technology. Haptics applies to a technology that aims to simulate the sensation of touch by applying forces or vibrations to the user. (Virsabi) As an example, vibrations can be used to allow the player to feel the way his virtual gun is shooting and make it more difficult to control the recoil of the weapon. Pressure-sensitive buttons are often used to control the operation of gas pedal in racing simulators. This means, that instead of pushing the pedal to the floor every time, the player can precisely control it. Force feedback can be mistakenly referred to as another, simple vibration mechanism, however, it has a more advanced approach to enhancing the players' experience. For example, in steering wheels, force feedback is used to provide the player with a true to life feeling of driving a car by creating resistance dependant on conditions such as slipperiness, dirt or bumps in the road. (Jesse Parker, 2020)

## 2.3 Voice recognition

The technology allows users to feed data in a computer by using their own voice. According to Asad Butt's (2020) statistics, by the end of year 2020, 50% of all searches across the internet will be voice-based, 30% of all searches will done using a device without a screen and 40% of adults use mobile voice search at least once a day already. Above clearly shows that this technology is becoming used more often and is more accessible, which will result in people using it more comfortably and more frequently.

### 2.1.3 Techniques

The speech recognition technology works by translating the vibrations we create as we speak through an analog-to-digital converter (ADC) into digital data. The speech recognition program then matches the signal that is divided into small segments to known phonemes and examines and compares them to known words, phrases or sentences. (Grabianowski) Furthermore, to increase the speed of recognition, the software loads the vocabulary into RAM to access it as it is much more efficient to find it cached in memory rather than on the hard drive. (Rouse, 2018) An example of processing of speech recognition system is illustrated in Figure 2.1.

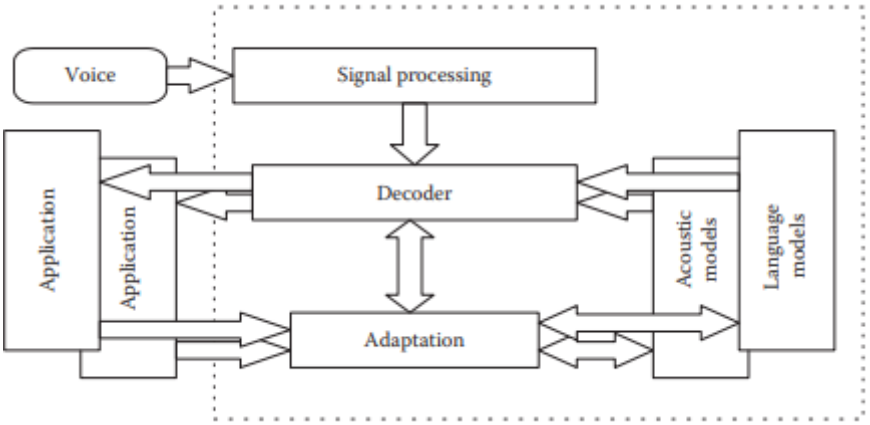



*Figure 2.1 Speech Recognition system architecture*
*(Source: Huang et al., 1996)*

After the voice data is fetched from the microphone, the it is pre-processed by dividing it into smaller parts. This digital data is then passed to Adaptation model which is the main speech recognition model. Pre-processing is done by changing the sound wave into a number by recording the magnitude of the voice wave at equally spaced points and dividing the data into groups. The speech recognition system then uses its memory to decide the possibility of receiving upcoming letters. (Amberkar et al., 2018) The results are returned by taking into account the Acoustic and Language Models by finding a similar or most probable word in the memory.

### 2.1.4 Microsoft Cognitive Services

Microsoft Cognitive Services, or also named Azure Cognitive Services, are REpresentational State Transfer (RESTful) Application Programming Interface (API) services that allow for "*natural user interaction on any platform on any device*". (Del Sole, 2017) These APIs can help make smarter decisions by detecting anomalies, extract meaning from text, convert speech to text, translate speech in real-time, analyze image and video content, detect and identify people or their emotions in images and even more. All above is done by simple ".NET API" calls that do not require the developers to master machine-learning. The main point of focus, which is the Speech Recognition Software Development Kit (SDK), enables audio processing with speaker recognition, voice verification, speech to text conversion, text to speech conversion and real-time speech translation. In order to achieve 5.1% error rate by the automatic speech recognition system, that surpasses the human accuracy level (Xiong et al., 2017), Microsoft developers use convolutional neural networks, bidirectional long-short-term memory, senones, speaker adaptation, sequence training and frame-level modeling for their Acoustic Models.

#### Convolutional neural networks

Convolutional neural network (CNN) is a Deep Learning algorithm and is used in order to model the sentences spoken by the user. CNN consist of a convolutional layer that performs an operation on data called "*convolution*". (Brownlee, 2019) This is an operation that involves applying two-dimensional array filters across each row of features in the sentence matrix. Specifically speaking, it multiplies the vector of weights and vector of inputs viewed as a sequence. (Kalchbrenner, Grefenstette & Blunsom, 2019) This results in capturing most relevant words to the spoken sentence. According to Abdel-Hamid et al. (2014), experimental results shown that CNNs reduced the error rate even by 10% compared to Deep Neural Networks.

### Bidirectional long-short-term memory

Bidirectional long-short-term memory (BLSTM) is an algorithm used for processing sequential data. The purpose of it, is to store the context for any position of a particular sentence sequence by reading the input sequence from left to right and right to left (Reczko & Thireou, 2007). This results in creating a context for each word in the given sentence that depends on both, its past as well as its future words. This approach allows to fill in gaps for missing words as it would fill a hole in the middle of an image instead of expanding it on the edge. (Schuster & Paliwal, 1997).

### Frame-level modelling

Frame-level modelling is used in Artificial Intelligence as a data structure which contains information such as how to use the frame, what to expect and how to handle situation when set expectations are met or not. Frame modelling allows the speech recognition technology to group related data. In Microsoft Cognitive Services, the frame-level modeling is a "*combination of senone posteriors from multiple acoustic models*", Xiong et al. (2017) also further describe that the model is limited by the fact that the underlying senone sets must be identical. This results in a low word error rate.

### Speaker adaptation

Speaker adaptation is a number of different techniques which supports the voice recognition system by adapting to specific, acoustic features of the user. (Shinoda, 2005) In Microsoft's Cognitive Services it is based on conditioning the network on a 100-dimensional i-vector that is generated for the audio of the speech. Since this approach would not be beneficial when using in conjunction with CNNs, a learnable weight matrix is added. This then serves for the CNN as an additional speaker-dependent bias to each layer. (Xiong et al., 2017)

## 2.1.5 Use of Voice Recognition in games

Gradually, new games that use voice recognition technology are coming out. Some of the games, that are available on Amazon's or Google's personal assistants, are Netflix's The 3% Challenge, LEGO: Duplo Stories or When in Rome. The above games keep the players entertained in a variety of ways. The 3% Challenge game is a prequel narrative to the first season of "3%" series. (Takahashi, 2019) The players join the process where the candidates are challenged and vetted. The used voice-based challenges test players' skills in hearing, memory or teamwork. (Doppio Games) On the other hand, LEGO: Duplo Stories, which is targeted at younger audience, brings physical play with interactive story telling. The main principal of this game is to allow the listeners to play a themed story that is narrated by the personal assistant. There are multiple scenarios to choose from (Ong, 2018), which can then be dependent on what the player chooses with his voice. When in Rome is a game developed for Alexa, that combines playing a board game and using voice recognition. Amazon's assistant is used to read the instructions, run the game, keep track of the score, check answers to questions about certain cities (Chin, 2018) that earn the players points if correct. It also adds sound effects to further improve the game experience. However, use of voice recognition in games is not only limited to 'personal' assistant devices. It is also used in a game called The Inpatient that was released for PlayStation 4. The Inpatient is a horror game with psychological elements. (Bowen, 2018) The game uses PlayStation's VR headset to allow the player to fully engage with the characters by speaking out loud. It uses the voice recognition technology by giving the player two potential responses that must be spoken rather than chosen by pressing a button on a controller. "*It's a relatively minor thing, but when combined with the immersive power of VR, it makes me never want to pick a dialog*

*option in another VR game ever again*". (JAGNEAUX, 2017) David's statement might indicate that using speech recognition technology can excessively impact on gameplay experience and move it to a whole new level. A different horror game, Dead Rising 3, uses Microsoft's Kinect's voice commands. This allowed the players to navigate through the game menu, make the survivors follow the player by giving a "Follow" command or instruct the survivors to attack nearby zombies by saying "Attack". The game also includes an option to taunt zombies by saying "Come over here" or "You're crazy" which made them attack the player. Moving into the action genre, Ubisoft used voice commands in their Tom Clancy's branding. When releasing their Tom Clancy's EndWar game, Tony Key mentions that "*voice command's natural accessibility delivers a more immersive and realistic gameplay experience*". (BUSINESSWIRE, 2008) The players were able to select specific units by saying "Unit" and the unit's number, order them to move to a certain place, attack a particular hostile unit or disengage from an enemy and retreat to a deployment zone. In their newer title, Tom Clancy's Splinter Cell: Blacklist, which belong to the First-Person shooter genre, the voice commands used in game made it possible for the players to use their voice in order to distract enemies or call in an air strike.

### 2.1.6 Understanding player's voice

In order to achieve the most realistic feeling for the player, the ordering of the commands or features of the game should not be based on player saying exactly the same words as requested by the game. For example, if there is a feature available to tell the player where his enemies are, he should not be requested to say "Where are the enemies" with the need to completely match that phrase but accept that the player should be able to speak naturally and freely. Therefore, the game should answer the above question when the player says "I want to know where the enemies are" or "Can you tell me if there are any enemies". This might be simple to achieve, as all that is needed is to specify common phrases for each command at the development stage. Then, in order to find the command that the player wanted to execute, a straightforward calculation may be used that determines the match of player's words and the command. Furthermore, since the speech recognition technology might struggle to capture all the spoken words correctly, either due accent or pronunciation, and instead use a similar sounding word, the Levenshtein algorithm can be used to find the most accurate match between said words and stated commands. The Levenshtein distance is used for measuring difference between two string sequences. It basically returns a number that describes how different the two strings are, meaning that the returned value is the least number of needed modifying operations to match one string with another. (CUELOGIC INSIGHTS, 2017) The equation for the Levenshtein Distance is shown on Figure 2.2, where a is the first string, b is the second string, "i" stands for the terminal character position of first string and "j" is the terminal character position of the second string.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

*Figure 2.2 Levenshtein Distance*
*(Source: Nam, 2019)*

In the conditional ($a_i \neq b$□), "$a_i$" refers to the character of string "a" at position "i" and "b□" refers to the character of string "b" at position "j". (Nam, 2019) The Levenshtein distance between "Dawid" and "David" is 1, since 1 change is required. The distance between "kill the enemy on the right" and "kill the enemy in the night" is 2. The latter example shows how advantageous using such algorithm might be in terms of understanding players intentions in the game. If the player executed a kill order on the enemy to his right, but the speech recognition has not precisely converted speech to text, the kill order would still be matched correctly.

# 3 Methodologies

This section explains what approach has been taken to gather and evaluate the results. This includes the types of data collected and sampling method.

## 3.1 Collecting data and analysing results

Both types, quantitative and qualitative, of data were collected in order to fully understand and analyse if the use of voice recognition technology has contributed to immersing the player and providing better gameplay experience. The gathered data was acquired online and stored confidentially, securely and did not require any personal details of the

participants apart from their voice recordings. Acquiring the data online helped follow Covid-19 health and safety regulations as the participants could use their own equipment to play the game and answer the questionnaire provided to them at the beginning of their test session.

### 3.1.1 Qualitative data

Throughout the game that included speech recognition, the players' voice and screen was constantly recorded. The gathered voice data was used in order to give an indication of players' experience, emotions and ways of using their voice in order to interact with the game's environment. This was also helpful to understand in what kind of situations the speech recognition technology might struggle, if at all.

### 3.1.2 Quantitative data

Quantitative data was collected throughout two questionnaires using Google Forms, which can be viewed in Appendix 1 (group 1) and 2 (group 2), that was supplied at the beginning of the participants' contribution. After providing consent to take part in the experiment, the participants were supposed to fill out the first part of the form and after finishing the game, the second section of the form. The first element of the questionnaire was about their gaming experience and the second one about their thoughts about the game. The answers were interpreted with statistical analysis, hence providing a scientifically objective (Carr, 1994), rational and measurable view of how use of speech recognition technology affected how players perceive the game.

## 3.2 Sampling

The aim was to recruit the participants from the Glasgow Caledonian University. Three participants were found online to gather unprejudiced results. The participants were briefed about what the project is about and about their rights, including the fact that their voice will be recorded if playing the game including the use of speech recognition technology. They were also notified that they were able to request any data – questionnaire answers or recordings - to be deleted at any point without giving a reason. Only non-vulnerable, adult participants were recruited. Due to limitations caused by Covid-19, a non-probability sampling method, convenience sampling, was used as it was easy, fast and cost effective.

## 3.3 Evaluation

The twelve, randomly selected participants were divided into two control groups that determined the results of the research. Both groups played a First Person Shooter game. One group played the game that featured speech recognition technology that was available for them to use to perform certain interactions with the game. Whereas, the second group played an analogous game, however, in this instance the in-game commands could only be utilized using keyboard keys. Both groups completed questionnaires that were adapted to the game they have played. Playing the game and answering the questionnaire took the participants around 10 minutes. Before starting the game, the players were told what the aim of the game is, what are the win and lose conditions, what features are available and were also briefed with a map as seen on Figure 3.1.

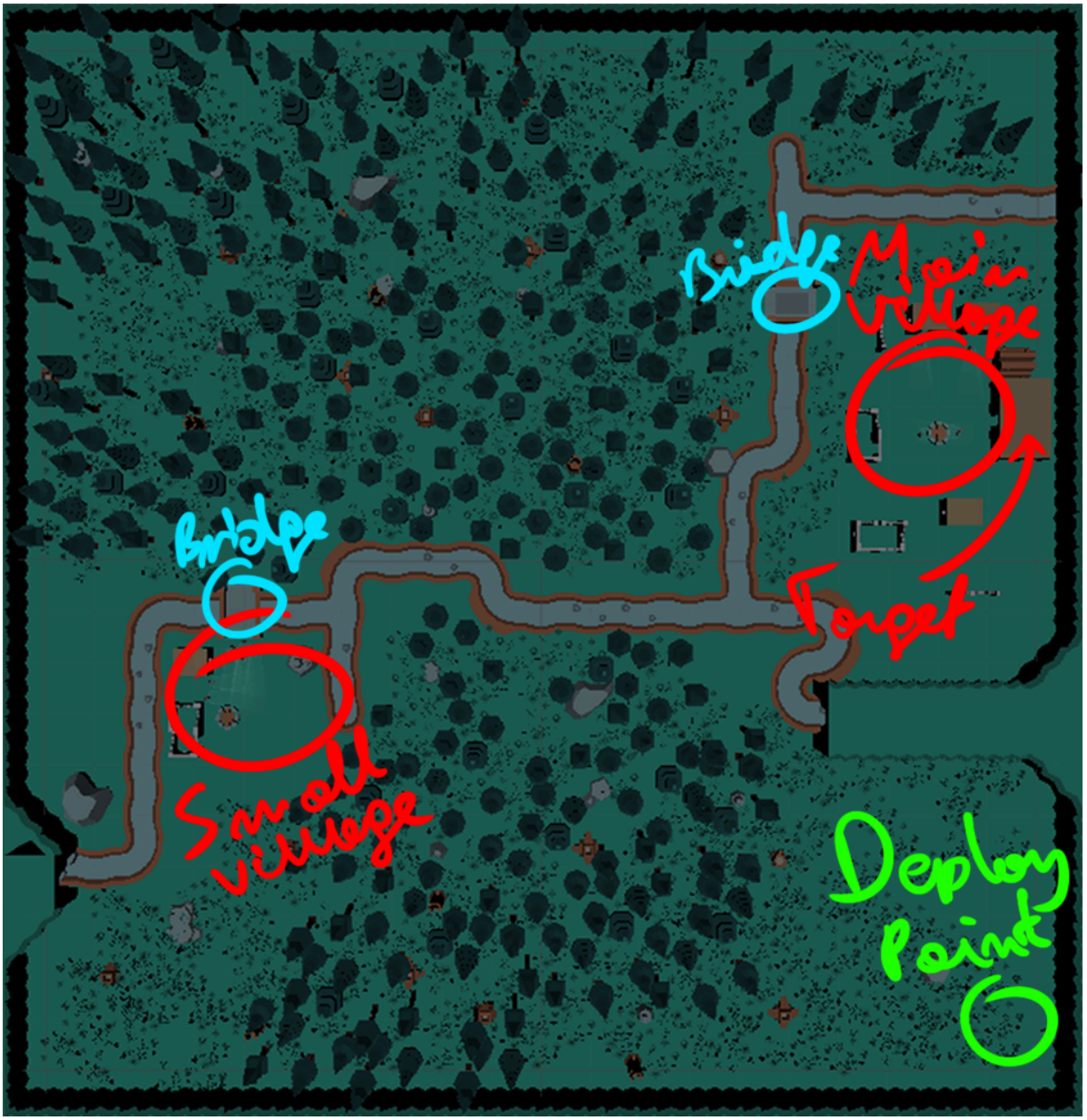



*Figure 3.1 Map used for briefing the players.*

# 4 Execution

The chapter provides the specification of how the project has been developed, including the tools and methodology used to create the game, describes the game's design, features and implementation details. All assets used in the game can be viewed in Appendix 3.

## 4.1 Development environment

The part explains what kind of tools, such us IDE, language, speech recognition service and methodology have been used to create the game.

### 4.1.1 Tools

#### Integrated Development Environment

Unity is a cross-platform game engine developed by Unity Technologies which supports building applications for mobile, desktop, web, console and virtual reality platforms. Unity was used as the Integrated Development Environment in order to create the game with Visual Studio, from Microsoft, acting as the source-code editor.

#### Language

C#, developed by Microsoft, programming language was used for the development of the whole project due to being the main language used for Unity and the Voice Recognition software.

#### Speech Recognition

Microsoft Cognitive Services, or Azure Cognitive Services, is a service developed by Microsoft used for developing Artificial Intelligence applications and services. Speech SDK, which is included in the Speech services of Cognitive Services, was integrated into Unity and used to convert player's voice into text that was used by various scripts in the project.

### 4.1.2 Methodology

#### Agile

Agile software development method has been used to develop the final product. Thanks to being an iterative development process, as seen on figure 3.1, Agile helps to quickly test and develop new features into the final product.

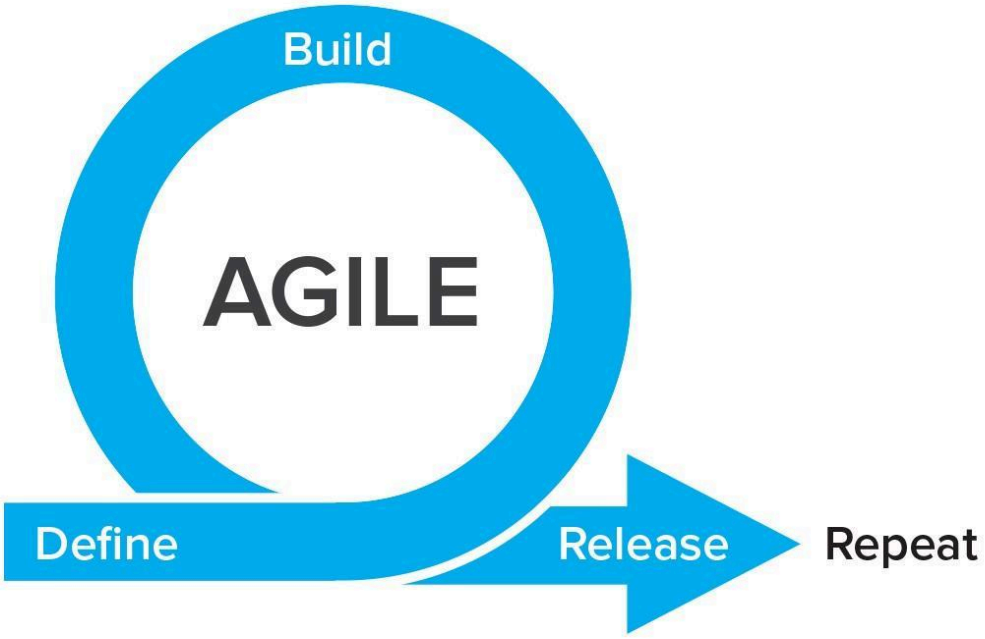



*Figure 4.1 Agile Development Methodology*
*(Source: Ibanez, L., 2017)*

This approach allows to be responsive to changes or any issues that my occur during the development of the project. The costs and efforts of fixing any defects are reduced as any bugs are discovered at the implementation stage. Figure 3.2 shows a comparison on how the relative costs of fixing defects change depending on project stage.

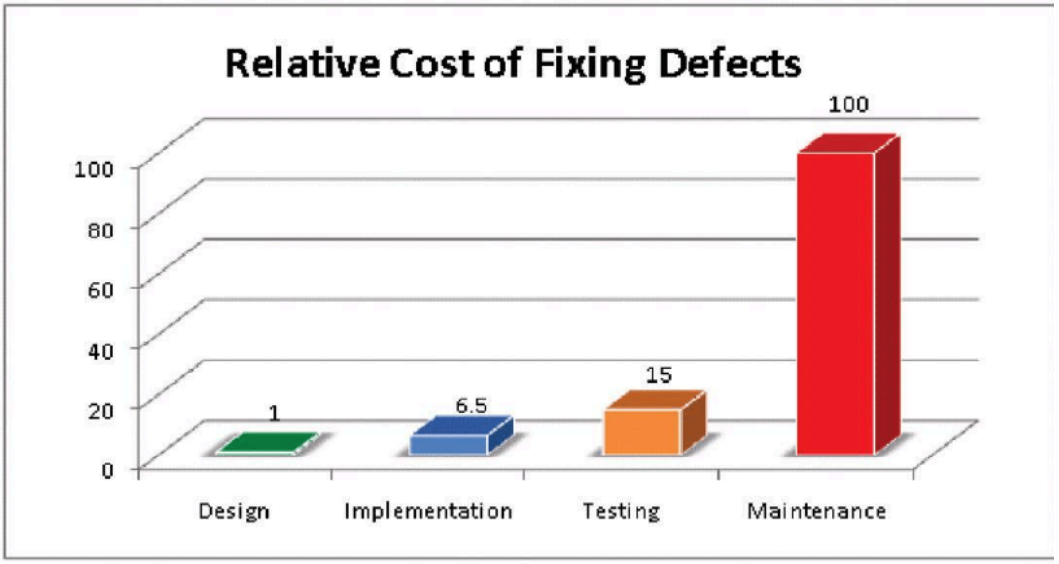



*Figure 4.2 Relative Cost of Fixing Defects*
*(Source: Dawson, M., Burrel, D. & Rahim, E., 2010)*

Agile development process was crucial when developing the game as it provided the opportunity to spot any potential issues with integrating the speech recognition, finding matching commands for spoken phrases and general gameplay playability at a very early stage. The methodology has actually helped notice a matter with finding the most accurate command depending on recognized speech using the Levenshtein Distance. Moreover, the approach allowed to implement the project stages step by step with the certainty that they were high-quality. The final product was accomplished by splitting it into the following structure of iterations:

1. Integrating Speech Recognition in Unity
2. Game environment (design and map)
3. Gameplay (objectives, commands and enemies)
4. Recognizing Commands (most appropriate command depending on user speech)

## 4.2 Early prototype (Integrating Speech Recognition in Unity)

As stated above, using Agile methodology helped test if the speech recognition can work correctly and produce reliable results in Unity at a very early stage of the project lifecycle. An initial prototype has been developed in Unity, which included the functionality to convert speech into text, as shown in Figure 3.3, and to use certain voice commands to control the cube's movement.

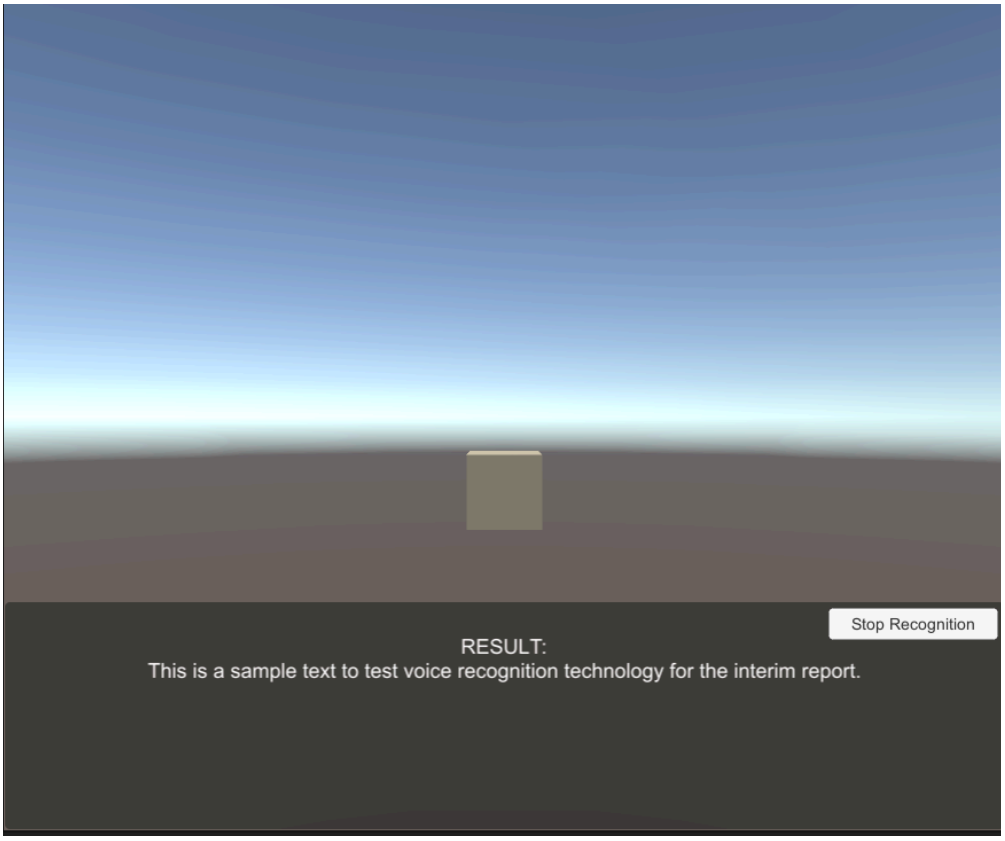



*Figure 4.3 Result of generating text from speech in the prototype*
*(Source: Personal Collection)*

The prototype was produced to analyse possible ways of determining commands and assess the accuracy of Microsoft's Speech SDK when converting speech to text. Basic movement instructions for the cube have been implemented, such as "Move up", "Move left", "Move forward" et cetera, in all possible directions: up, down, left, right, forward, back. Furthermore, a simple interpretation mechanism of the spoken words has been created which works in a way that if the user negates the action, the cube will not move. For example, if the recognized speech is "Move down, but not left", the cube will move one coordinate down and will ignore the command to move left. Moreover, in order for the cube to move, the user must indicate that he expects an action to be performed, for example by saying "move", "shift" or "step". As mentioned before, the speech is recognized using Microsoft's Cognitive Services. This is done by sending the data gathered by the microphone to a resource created in the Azure cloud platform. The resulting string, containing the recognized speech, is then returned by the Speech Recognition service and analysed in one of the developed scripts to perform the movements on the cube.

## 4.3 The game

This section gives a detailed explanation of how the game has been designed, specifies the objectives and available commands for the player throughout the various stages of the game.

### 4.3.1 Design (Game Environment)

Describes the "Game Environment" iteration of the project. Explains the choices of visual effects and map layout.

### Graphics

In terms of visual aspects, from the beginning of the project the intention was to use a similar visual styling such as the one that can be found in Battlelands Royale mobile game (Figure 3.4).


*Figure 4.4 Screenshot from the Battlelands Royale mobile game*
*(Source: App Store, Apple Inc.)*

The low-poly graphics style was not too problematic to achieve, which allowed to focus mainly on delivering a high-end speech recognition system and various commands for the player to trigger. Furthermore, as previously mentioned, since the graphical aspects may define if the game is immersive or not, the following visuals would not bias the participants' opinion as realistic graphics could. The full list of models used in the game can be found in Appendices 1.

The game uses a dark colour palette with green colour dominating the whole environment as seen on Figure 3.5. This is due to the story of the game being set at night-time and the player's character using night vision.

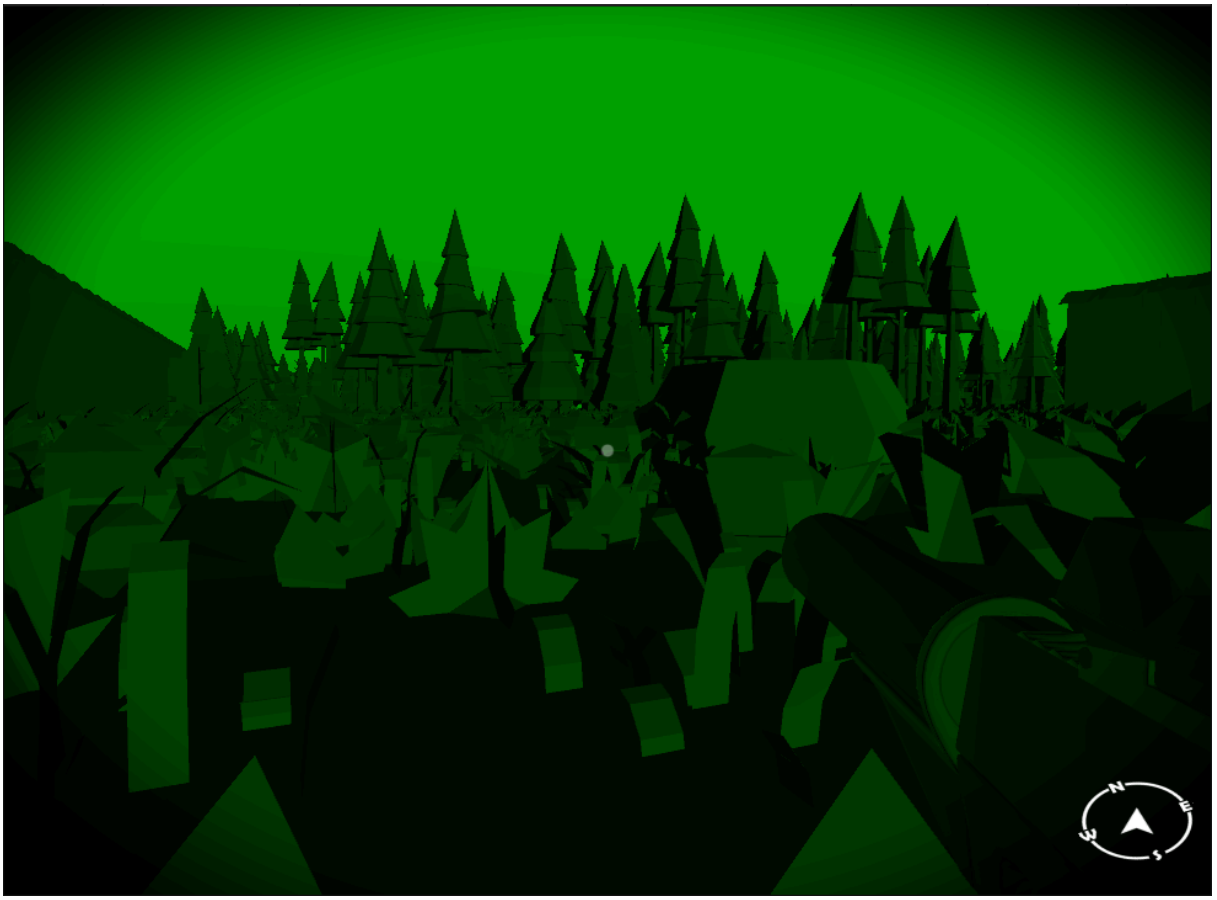



*Figure 4.5 Screenshot from the developed game of player's view*
*(Source: Personal Collection)*

### Map

The game takes place in a forest. The layout of the map was designed in such a way that the player has to visit and pass by certain points on the map. The top view of the map can be seen on Figure 3.6. The design of the map forces the player to move according to the expectations and certain events, such as change of possible commands to use, can be executed. "A" marks the player's starting position. "B" identifies the small village, where the player firstly faces a group of enemies and then two enemies he needs to kill. "C1" and "C2" show the bridges the player must use to get through the river. "D" marks the main village where the player faces four enemy soldiers. Finally, "E" is used to illustrate the building where the main target is located.

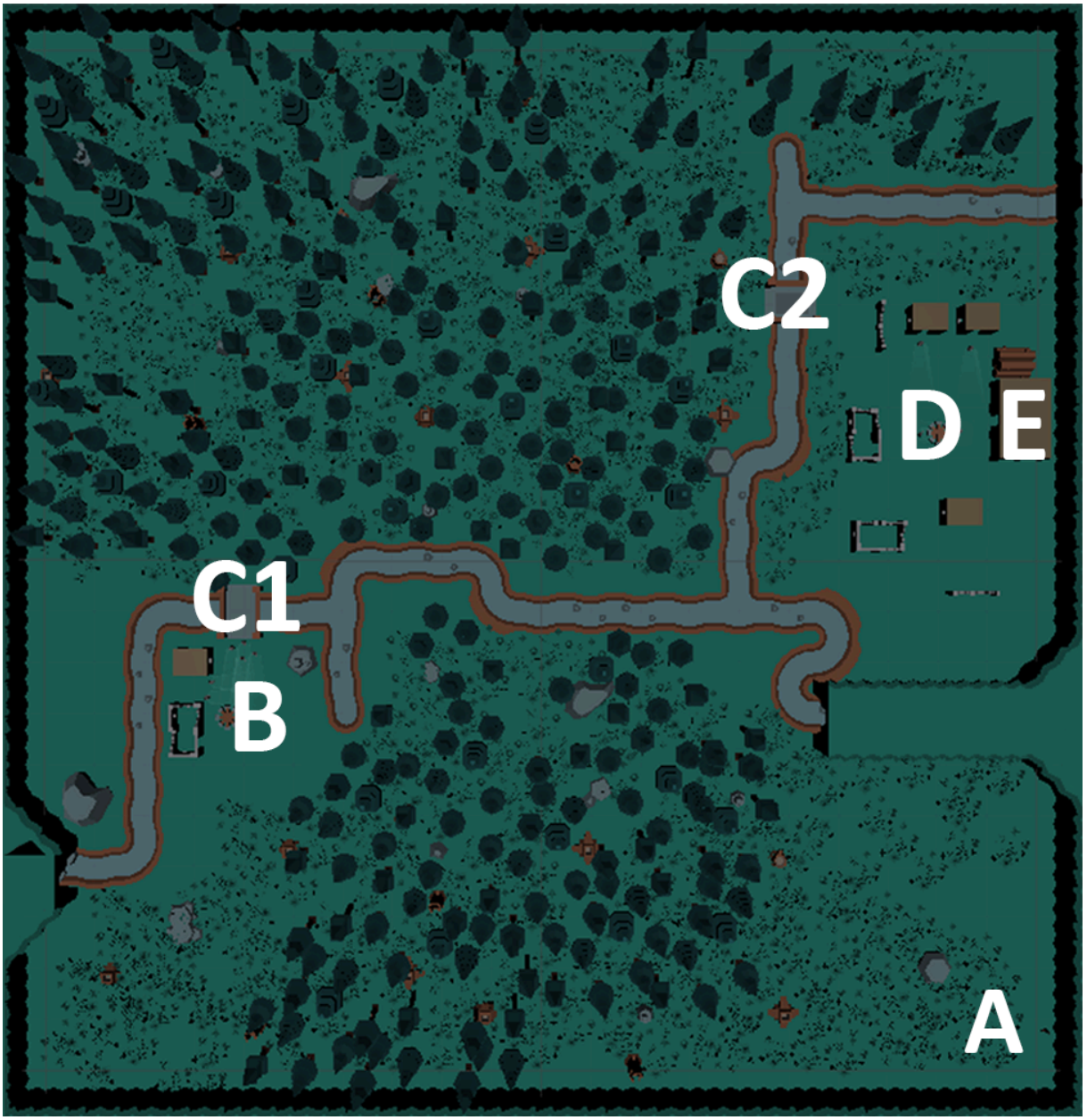



*Figure 4.6 Top view of the map from the developed game*
*(Source: Personal Collection)*

### 4.3.2 Gameplay

Developing the gameplay has been the biggest batch of changes done simultaneously in one Agile iteration. This part of the report specifies the features of the game.

#### Aim of the game

The primary aim of the game is to execute the boss soldier that is located in one of the buildings located in the main village. Throughout the journey, the player cannot allow the enemy soldiers to raise an alarm. The player wins after killing the prime target and can lose if one of the enemy soldiers has the player's character in sight for a particular amount of time or spots another enemy soldier that is dead. In the latter case, the alarmed is also not raised instantly, but after a very short period of time. In order not to lose, the player must act and kill the suspicious soldier.

### Ally Feature

The so-called Ally in the game, is a sniper companion for the player that answers the player's commands or helps him execute enemies.

### Commands

Depending on current state of the gameplay, the player has the ability to ask the Ally questions or order the Ally to execute a specified target. Throughout the whole game, the player can ask standard questions such as what he needs to do, where to go or if there are any enemies near him. The Ally will answer accordingly to current mission, for example, that the player needs to kill two enemies in the village and then use the bridge to get across the river. For the latter question the Ally will tell the player the number of enemies with their geographical direction (if present), for example "One soldier at North and two soldiers at West". Since the game is not only about asking questions but also making the player's speech interact with the Ally, the user can also tell him to kill an enemy to his relative position during missions. For instance, the player can ask Ally to kill the enemy on his left or right or even the one closer or further to him. Apart from that, the player is able to ask the Ally to confirm the target he is aiming at and also their status (alive or dead). Finally, once the player reaches the main village, he is advised to kill the moving enemies first, and then deal with the two soldiers near the campfire. The player can then 'force' the Ally to help him with the enemies near the fire first by saying a similar phrase to "Let's kill enemies near campfire first". This provides the player with an opportunity to alter the storyline of the game.

### Enemies

In the game, there are eleven enemy soldiers in total, including the boss soldier. A group of four soldiers is scripted, so that when the player approaches the small village, they move away from the bridge to the left bottom corner of the map (top view). The player cannot engage this group as he is not capable of killing all four enemies without raising an alarm. Two more soldiers are located in the small village. These soldiers just stand in front of one of the buildings and need to be killed with the help of the Ally in order to allow the player to move to the bridge unnoticed. In the main village, there are two soldiers near the campfire, boss soldier in one of the buildings and two enemies that wonder around the village. As previously described in the "Aim of the game" section, the enemies are equipped with a script that allows them to spot the player or dead enemies nearby. If that is the case and sufficient amount of time has passed, they will raise an alarm and the player loses the game.

## 4.4 Implementation details

This part of the report gives a better overview of how the project's most significant features have been developed. This includes the enemies' behaviour, integration of speech recognition, details about in-game commands, how player's intentions were determined and the functionality of the Ally. Full project can be viewed in Appendix 4.

### 4.4.1 Enemy Soldiers

Enemy script can be viewed in Appendix 5. The enemies in the game are designed to have two behaviour types. Some enemies are just defending one single point, whereas other move around predefined points on the map. The time they spend standing at each point is randomized every time they reach the spot. The minimum and maximum random wait time is defined for each soldier separately for even better unpredictability. Apart from above, they operate alike. Their main functionality is to look for the player or any dead enemies near them. In order to consider the player or enemy visible, it must appear in front of the soldier

within a 90° angle and the distance between the visible object must be within three game units. However, in order to make the behaviour slightly more realistic, if the player is moving within 0.8 distance game units behind the soldier, the enemy soldier will rotate towards the player as he has been 'heard' and will be noticed. Once above conditions are met, a Raycast from enemy is sent in the direction of the player or dead soldier to make sure that there are no other objects between them such as a solid wall. The code responsible for above functionality can be viewed in Figure 4.7.

```
bool CheckIfPlayerVisible()
{
    Vector3 targetDir = playerObject.transform.position - transform.position;

    float angleToPlayer = Vector3.Angle(targetDir, transform.forward);

    float distToPlayer = Vector3.Distance(playerObject.transform.position, transform.position);

    if ((angleToPlayer >= -45 && angleToPlayer <= 45 && distToPlayer < 2.5f) || distToPlayer < 0.8f)
    {
        RaycastHit hit;
        if (Physics.Raycast(transform.position + new Vector3(0, 0.07f, 0), targetDir, out hit, 4))
        {
            Debug.DrawRay(transform.position + new Vector3(0, 0.07f, 0), targetDir * hit.distance, Color.yellow);

            if(hit.transform == playerObject.transform)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            Debug.DrawRay(transform.position, targetDir * 1000, Color.white);
            return false;
        }
    }
    else
        return false;
}
```

*Figure 4.7 The code liable for checking if the player is visible*
*(Source: Personal Collection)*

If the player or dead enemy is visible, the soldier will rotate towards their direction. If the player is spotted, he has four seconds to either kill the soldier that noticed him or run away. If a dead enemy is spotted, the time is reduced to two seconds and the soldier must be killed to prevent him raising an alarm. Last but not least, in order to kill the enemy, the player must shoot him in the head once or anywhere else, twice. If the enemy is not killed with a single shot, the player has, again, two seconds to kill the soldier or otherwise he will raise an alarm.

### 4.4.2 Speech Recognition

In order to use Microsoft's Cognitive Services an Azure account was required and Speech resource was created. Microsoft's Speech SDK is available from Azure-Samples account on GitHub, which can be seen on Appendix 6. Since the Speech SDK is available in a ".unitypackage" format, the integration into Unity was very straightforward. The package was simply imported as any other custom Unity package. Before using Speech SDK, the Speech Service API Key and Speech Service Region were specified during recognizer creation. The API Key and Region were accessed from the created Speech resource under "Keys and Endpoint" section as seen on Figure 4.8.

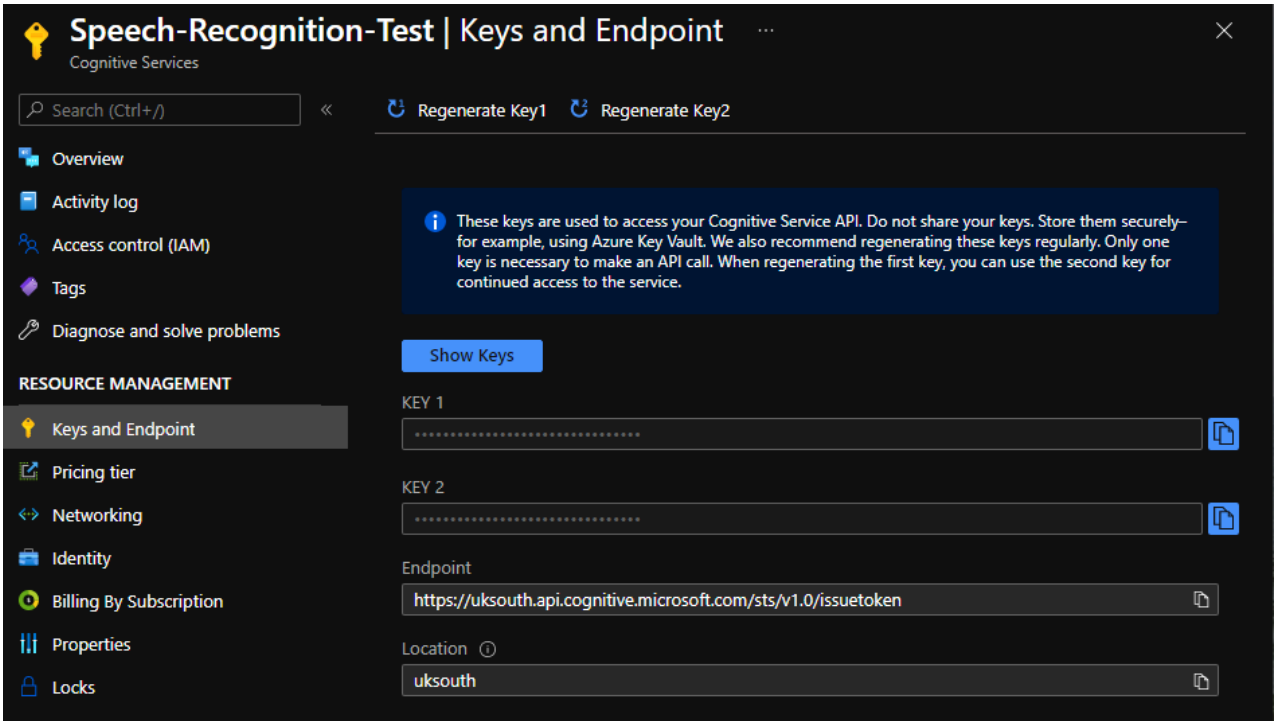



*Figure 4.8 Keys and Endpoint section of Speech Azure resource*
*(Source: Personal Collection)*

The speech recognizer was subscribed to the following event handlers:

- **Recognizing Event Handler**
  This event is started every time interim results are returned during recognition.
- **Recognized Event Handler**
  This event is triggered when the speech recognizer determines that the utterance has ended.
- **Canceled Event Handler**
  This event is called if the server encounters some kind of error.

In order to prevent the user's speech being recognized constantly, which could result in exploiting the resources and misinterpreting commands, the player is required to hold "T" key to communicate with the Ally, thus have his speech recognized. However, the recognizing could not be stopped as soon as the key was released, but once the server processed the speech. Therefore, two Boolean variables have been used – "recognizeText" and "recognizedText". The first one had a true value assigned as long as the key was held down. This allowed to avoid the player passing more words to be recognized than intended as within the Recognizing Event Handler the value of this variable was checked at the beginning of the method and if it was set to false the function was terminated. This allowed the server to take its time to process the information and easily determine the end of the utterance which then triggered the Recognized Event Handler. Due to the fact that the recognized handler was sometimes unexpectedly called or called multiple times, the latter Boolean variable was used. This helped make sure that the result will only be shown on the screen as a Text UI object once and will clear itself in the expected time – five seconds from when the result was gathered. The final recognized string is passed to the script responsible for

checking the text and interpreting player's intentions in regards of using the commands. Full configuration of the speech recognition script can be viewed in Appendix 7.

### 4.4.3 Commands

One of the scripts, which can be viewed on GitHub in Appendix 8, is responsible for dealing with the recognized speech converted into the string format, it searches for most appropriate action the player might have requested. As mentioned before, the available commands change depending on current state of the game. In order to achieve this functionality, the command operations have been split into Ally action type groups. As shown on Figure 4.8, the groups have been saved as an Enumeration type and these are:

- **Telling about enemies**
  Notifies the player about enemies nearby. Specifies the number of enemies and their geographical directions.
- **Telling what to do or where to go**
  Depending on the current objective, notifies the player what he should do and / or where to go.
- **Determining the enemy to kill**
  Defines the enemy the Ally should kill once instructed. This can be either an enemy on the left or right side of the player or closer or further to the player.
- **Confirming target**
  Acknowledges which enemy the Ally will kill when told to.
- **Telling about enemy status**
  Reports whether the enemies the player and Ally should kill are dead or still alive.
- **Enemy kill order**
  Kills the enemy specified by the player
- **Forcing mission four**
  When in the main village, the player is instructed by the Ally to deal with moving enemies first and then, with his help, kill the enemies near the campfire. The player can command to deal with the enemies near the campfire first.

```
public enum actionType{
    TellAboutEnemies,
    TellIfEnemyDown,
    TellWhereToGoWhatToDo,
    DetermineEnemyToKill,
    ConfirmTarget,
    KillEnemyOrder,
    ForceMissionFour
}
```

*Figure 4.9 The Enumeration type used to group actions*
*(Source: Personal Collection)*

At the start of the game, before the first frame update, a Dictionary is populated with possible texts and action types such text should execute. The texts (string) are used as keys

and related actions (enum actionType) as values. As an example, Figure 4.9 shows how the Dictionary can be set up with possible words to execute the “Tell about enemies” action.

```
// TellAboutEnemies();
string[] enemyNames = { "target", "tango", "enemy", "soldier" };
for (int i = 0; i < enemyNames.Length; i++)
{
    string pluralName = enemyNames[i] == "enemy" ? "enemies" : enemyNames[i] + "s";
    actionsDict.Add($"Where is {enemyNames[i]}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"Where is the {enemyNames[i]}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"Where are {pluralName}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"Where are the {pluralName}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"Are there any {pluralName}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"Are there any {pluralName} nearby", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"Can you see any {pluralName}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"Positions of {pluralName}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"What are the positions of {pluralName}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"Do you see any {pluralName}", Action.actionType.TellAboutEnemies);
    actionsDict.Add($"How many {pluralName} do you see", Action.actionType.TellAboutEnemies);
}
```

*Figure 4.10 Setting up the Dictionary with possible texts to call the "Tell about enemies" action (Source: Personal Collection)*

Apart from the action groups, the player can also use words such as “Roger”, “Understood”, “Got it” or “Thanks” to clear the Ally’s response that appears on the screen as a Text UI object, for example, after answering player’s question about the enemies’ status. Since certain groups of actions can only be requested during specific game stages, it means that depending on that stage, only appropriate action groups are checked. The “Tell about enemies” and “Tell what to do or where to go” are always included in finding the most appropriate match, however, for instance, the “Determine enemy to kill” is only searched when the Ally is waiting for the target to be specified by the player. And accordingly, once the player determines which enemy the Ally should kill, only the “Confirm target”, “Enemy kill order” and “Tell about enemy status” will be searched through.

### 4.4.4 Understanding Player’s Speech

Lievenstein algorithm, that was described in the Literature and Technology Review chapter, was initially used to determine most appropriate command match based on player’s speech. A List variable has been created that was responsible for storing all the ‘distances’ between player’s speech and the strings in the actions Dictionary. This value was collected when iterating through the dictionary and comparing the keys with converted speech. Once the whole Dictionary has been searched through, the position of the minimum value from the List is determined, which is also the position of the most probable element in the Dictionary. Finally, before calling the action, the percentage of match between both strings was calculated, and if higher than sixty percent, the appropriate Ally action was called. However, what was considered a superiority at the beginning, appeared to be a huge disadvantage. Due to Lievenstein Distance working on single characters rather than whole words, it sometimes matched commands even when the expression contained something excessively different and irrelevant to the subject. As a result, the way the speech is matched to actions has been modified to work on whole words rather than the characters. This means that the utterance’s whole words are compared one to one against the Dictionary’s keys. In order to allow for some flexibility for the player, so that he does not have to speak out exactly the same sentence as is stored in the Dictionary, the same percentage, as when using Lievenstein Distance, of the match allows to call an action. This also means that the List that used to

store the Lievenstein “distances”, was saving the percentage match now. Therefore, when finding the most appropriate match this time, rather than looking for the minimum value, the maximum value was searched for. The calculation of the percentage of matching words in both strings can be seen in Figure 4.11.

```
public static int MatchingKeywordsPercentage(string source, string target)
{
    if (string.IsNullOrEmpty(source) || string.IsNullOrEmpty(target))
        return 0;

    string[] sourceWords = source.Split();
    string[] targetWords = target.Split();

    int sLen = sourceWords.Length;

    int matchingKeywords = 0;

    for(int i = 0; i < sourceWords.Length; i++)
    {
        for(int x = 0; x < targetWords.Length; x++)
        {
            if (sourceWords[i].ToLower() == targetWords[x].ToLower())
            {
                matchingKeywords++;
                break;
            }
        }
    }

    return (int)(matchingKeywords * 100 / sLen);
}
```

*Figure 4.11 Calculation of the percentage that strings match*
*(Source: Personal Collection)*

The percentage of the match is calculated by, firstly, splitting the words from both strings into separate arrays. The “source” string is the key from the Dictionary and “target” parameter is the player’s utterance. Secondly, each word from the source’s words iterates through the target’s words, and if they are the same, the number of found matching words is incremented. Finally, the function returns the percentage of the match. The new matching functionality has been tested against Levenshtein algorithm and during the tests it appeared to work better and produce more accurate matches. The individual texts’ results can be viewed in Appendix 9.

### 4.4.5 Ally

The Ally was a fictional in-game character that answered player's questions or executed his orders. The script can be viewed in Appendix 10. As mentioned above, the actions were determined in another script and the Ally was only responsible to return results for specific actions. In some instances, to improve the realism of the Ally, his answers were randomized. For example, if the Ally shot the enemy on player's request, he would confirm the kill using one of the four available texts:

- Alpha, enemy down.
- Alpha, tango down.
- Alpha, target dead.
- Alpha, enemy killed.

Similar functionality, randomized answer, was also applied to when the Ally struggled to identify the target specified by player.

# 5 Evaluation and Discussion

## 5.1 Quantitative data

As mentioned in the methodologies, quantitative data was gathered using a questionnaire created on Google Forms. Apart from the consent section, the participants had to complete two sections, the first section was about their gaming experience and the second one contained questions related to the game they played. It was important to find out about players' gaming experience to evaluate whether that also might have had an influence on the final answers. Since the participants were divided into two groups, two separate questionnaires were created. One for the group that played the game without speech recognition (group 1), and the other one for the group that played the game with the speech recognition (group 2) functionality included. In total, twelve participants undertook the experiment, with six for each group. As stated before, both forms can be viewed in Appendix 1 and 2.

### 5.1.1 Gaming Experience Section

This section covers the results from section one of the questionnaire from both groups.

#### Question 1

First question was about the participants rating themselves in regards to being a gamer – "How would you rate yourself in regards to being a gamer?" – with one being a casual gamer and five a hardcore gamer. The answers from both groups can be seen on Figure 5.1 (group 1) and Figure 5.2 (group 2). The question was asked to indicate whether being a hardcore gamer may influence rating the game lower and if the game features are also enjoyable for casual gamers. The results show that the player types from both groups spread quite evenly which is an ideal situation as answers to questions regarding the game should not be biased by gaming expertise level.

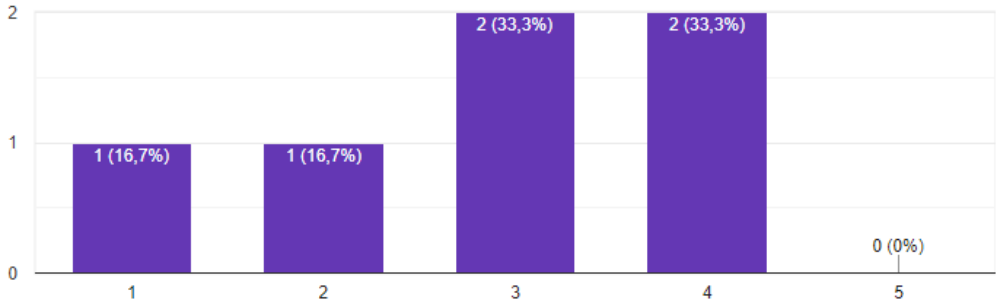



*Figure 5.1 Group 1 Question 1.1 answers (Source: Personal Collection)*

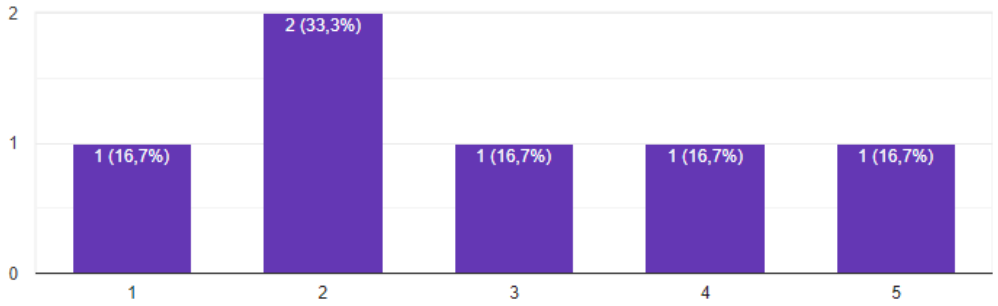



*Figure 5.2 Group 2 Question 1.1 answers (Source: Personal Collection)*

## Question 2

"How would you rate First Person Shooter games?" was asked to determine whether enjoying playing First Person Shooter games may influence the overall experience with the game. One represents "Don't like them at all" and five "Just love them - it's the only type of games I play". From the results for the first group, Figure 5.3, it is noticeable that most of the participants do not mind playing FPS games. There is a tendency towards "loving" these types of games rather than not "liking" them. On the other hand, Figure 5.4 shows the answers from group 2 and the likeliness of FPS genre is spread out a bit more. Since group 1 is leaning towards enjoying this genre, they are more probable to judge the game better than it is. Contrarily, group 2 may assess their game a bit lower.

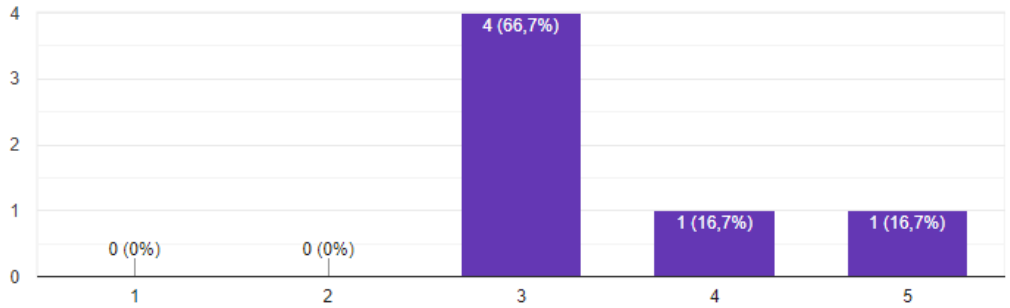



*Figure 5.3 Group 1 Question 1.2 answers (Source: Personal Collection)*

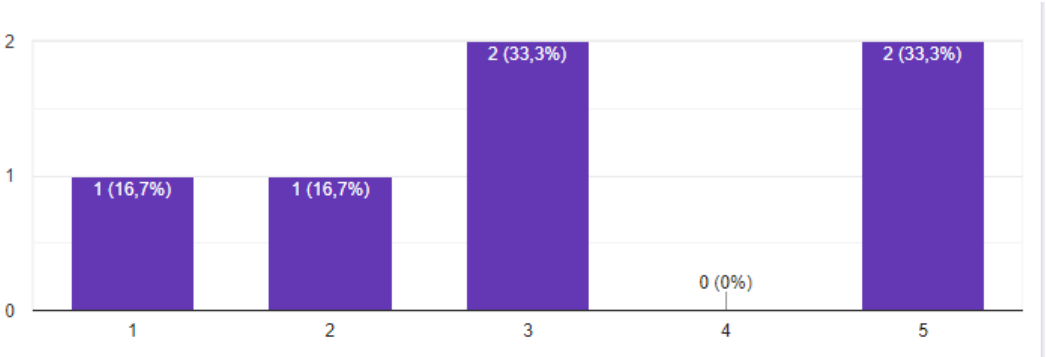



*Figure 5.4 Group 2 Question 1.2 answers (Source: Personal Collection)*

#### Question 3

Question number 3, “Which equipment do you usually use when playing?” was asked to determine whether in normal circumstances it would be possible for the players to fully enjoy the features of the game since a microphone is required. The answers from both groups can be combined for the purpose of evaluating that. In total, eight participants answered that they usually use headphones (66.70%), and four that use speakers (33.30%). None of the participants answered that they use speakers and a microphone. This indicates that the speech recognition technology would not interfere much with players’ usual playing style.

### 5.1.2 Playing without speech recognition

This section covers the results of the questionnaire given to the group that played the game without using speech recognition technology. To interact with the Ally, the players used keyboard keys.

#### Question 1

The first question was only used for personal reference whether the participants had the objectives and features of the game properly explained.

#### Questions 2 and 3

The questions “Did you enjoy the feature of having an ally in the game that you can communicate with?” and “Was it easy for you to communicate with the ally?” were both answered “Yes”. None of the participants answered that they did not communicate with the Ally. They were used to evaluate whether the Ally features was used, was easy to use and enjoyable. Such a great enjoyment and easiness of use level for the participants might indicate that having just the Ally features itself may already be enough to enhance the players’ experience.

#### Question 4

“Without taking the visual effects into account, did you feel you had a realistic experience playing the game?” was used to help assess if the participants had a realistic experience which is often related to having a positive gameplay experience due to stimulating people’s emotions. Figure 5.6 shows that just half of the participants considered their game involvement realistic which may demonstrate that the game does not fully engage the players.

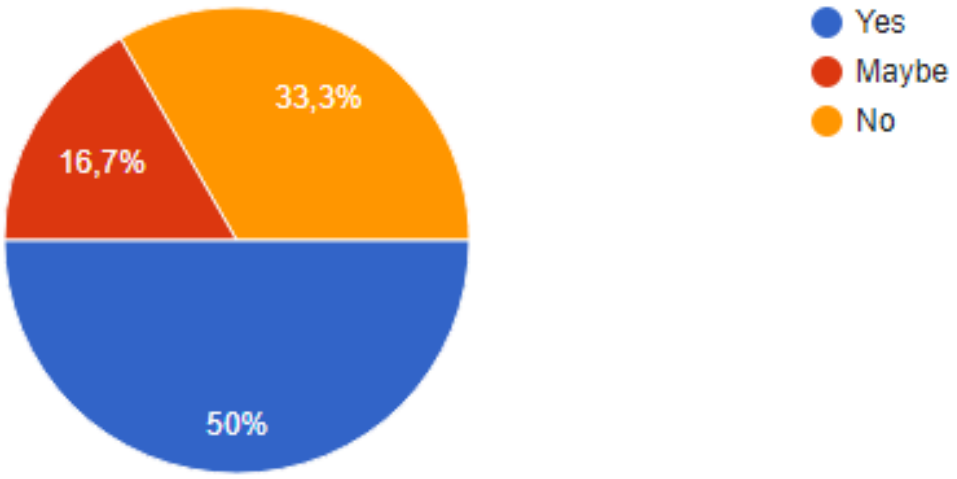



*Figure 5.6 Question 2.4 (Source: Personal Collection)*

### Question 5

In order to assess if the Ally feature was enjoyable, the "What did you like the most about the game?" question helps to find out whether a non-required and not predefined question would also be used by the participants to highlight this feature even more. As seen on Figure 5.7, to some extent, the answers can be grouped and three out of five mention the Ally feature which demonstrates that it was genuinely appreciated.

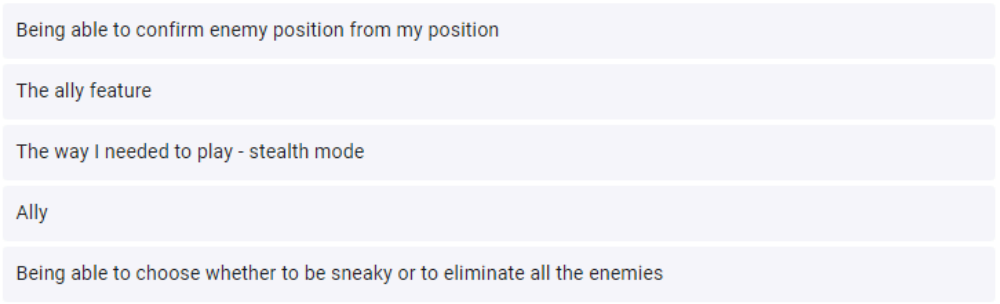



*Figure 5.7 Question 2.5 (Source: Personal Collection)*

### Question 6

"What did you not like at all in the game?" question was asked solely for the purpose of letting the participants also communicate about the negatives of the game. All the answers were focused on the graphical aspects.

### Question 7

The last question from the survey, "How likely are you to recommend this game to your friends?", assisted in determining that although the Ally feature was enjoyable for the players, it is not a major selling point for them and their friends. One stands for "Not likely at all" and five for "Very likely". The answers can be seen on Figure 5.8.

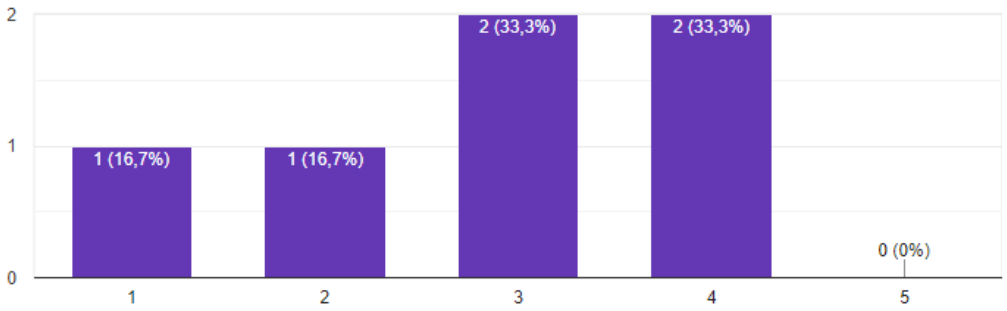



*Figure 5.8 Question 2.7 (Source: Personal Collection)*

### 5.1.3 Playing with speech recognition

This section covers the results of the questionnaire given to the group that played the game with speech recognition functionality included. In order to interact with the Ally, the players could only use their voice.

#### Question 1

The first question was only used for personal reference whether the participants had the objectives and features of the game properly explained.

#### Questions 2 and 3

The questions "Did you enjoy the feature of having an ally in the game that you can speak to?" and "Was it easy for you to communicate with the ally?" were both answered "Yes". None of the participants answered that they did not speak to the Ally at all. Exact same results as for group 1 mean that using the Ally feature is enjoyable and easy to use on the same level as when using keyboard keys.

#### Question 4

"How often did you speak to ally?" was asked to analyse how often the players communicated with the Ally. "Very rarely" is marked by one and "All the time" by five. The outcome, presented in Figure 5.13, demonstrates that the players spoke to the Ally regularly which then might indicate that it provided them with a positive experience.

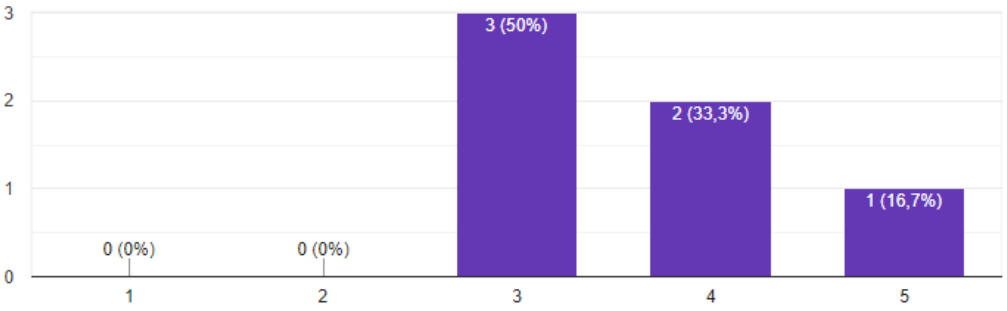



*Figure 5.13 Question 2.4 (Source: Personal Collection)*

#### Question 5

Figure 5.14 displays shows the answers for the "How often did you encounter issues with communication using the speech recognition technology?" question, where one stands for "Not at all" and five for "All the time", the participants answered that they did not have many

issues with recognizing their words. It is worth noting that according to these responses, each participant had at least one problem with recognizing their speech.

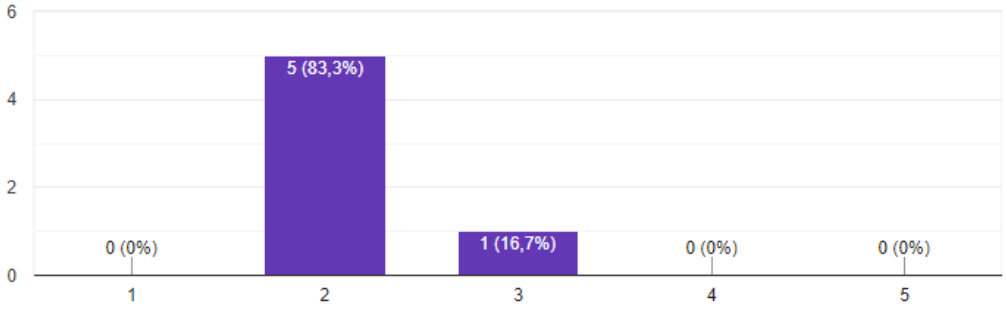



*Figure 5.14 Question 2.5 (Source: Personal Collection)*

## Question 6

The question “Without taking the visual effects into account, did you feel you had a realistic experience playing the game?” was not a single time answered “No” which may indicate that the players had a positive gameplay experience and were immersed by the speech recognition functionality. Figure 5.16 displays how the participants answered.

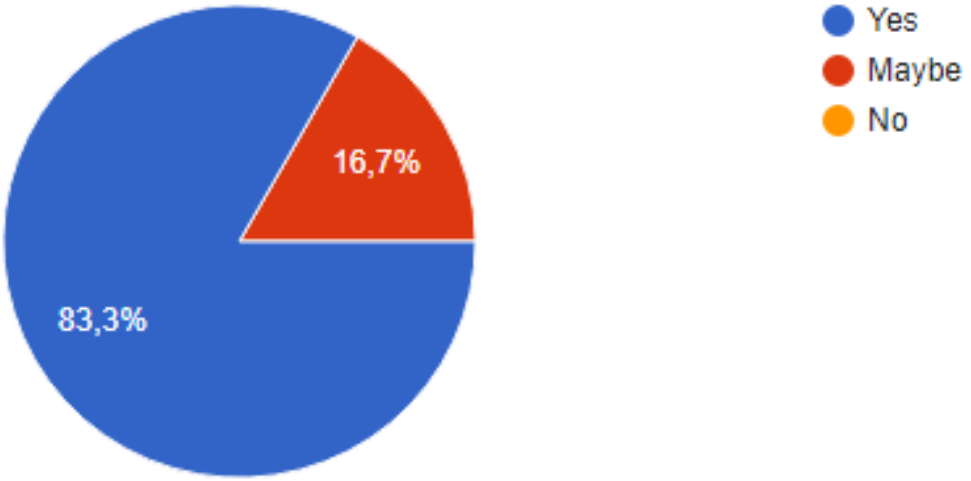



*Figure 5.16 Question 2.6 (Source: Personal Collection)*

## Question 7

In order to assess if the possibility to speak to the Ally was considered the main advantage of the game, the “What did you like the most about the game?” was used to find out whether it still would also be chosen for a question that is non-required and not predefined. Figure 5.17 displays the answers, which can be grouped. This would result in all four answers exalting the speech recognition technology.


*Figure 5.17 Question 2.7 (Source: Personal Collection)*

### Question 8

"What did you not like at all in the game?" question was asked solely for the purpose of letting the participants also communicate about the negatives of the game. Most of the answers were focused on the colour scheme used.

### Question 9

Figure 5.18 displays the results for question "How likely are you to recommend this game to your friends?". One stands for "Not likely at all" and five for "Very likely". The outcome clearly shows that the game was appreciated by the participants and would be by their friends.

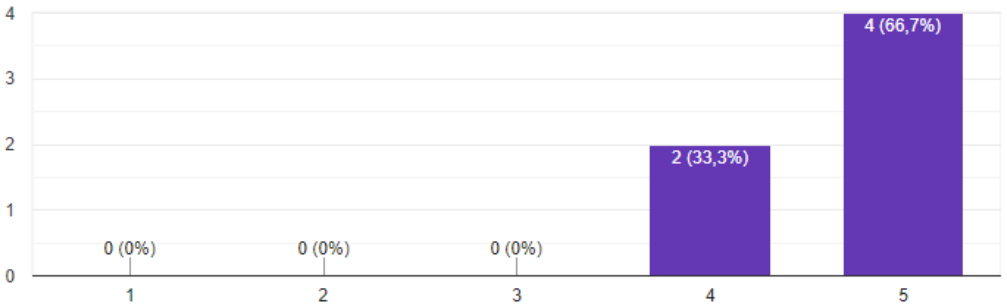



*Figure 5.18 Question 2.9 (Source: Personal Collection)*

### Question 10

Last question, "Would use of speech recognition technology in game influence your purchasing decision when buying a game?" did not have a "No" answer which. The results are displayed in Figure 5.19 and indicate that the players enjoyed the speech recognition feature to the extent that they would buy a game based on having such technology included into it.

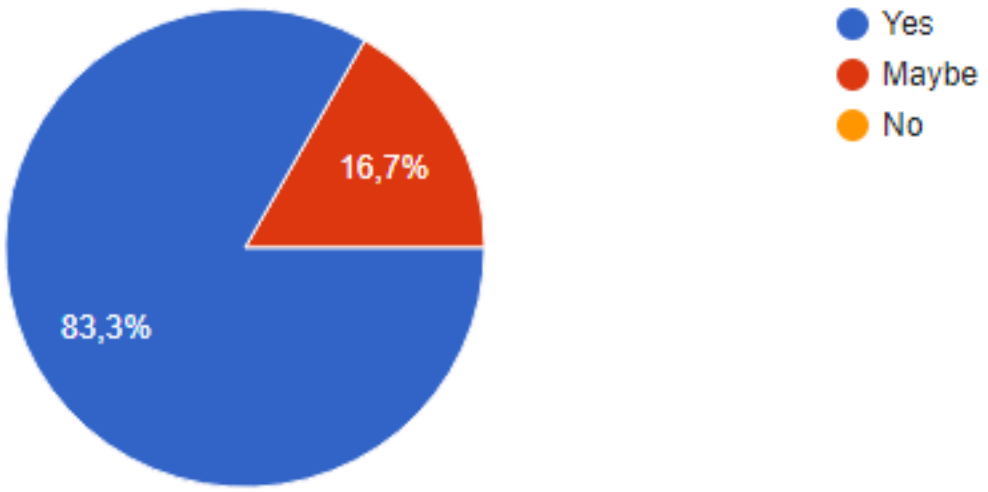



*Figure 5.19 Question 2.10 (Source: Personal Collection)*

## 5.2 Qualitative data

The qualitative data was collected when watching the players interact with the game. Throughout their gameplay their screen and voice has been recorded. The players' gameplay was evaluated during the participants' play and also after the experiment finished by examining the recordings once again. The qualitative evaluation of the players' interactions tried to investigate three main topics:

- Usage
- Quality
- Engagement

### Usage

This covers how the players used the speech recognition technology and in which situations. Was mostly focusing to investigate whether it is used correctly and if it is straightforward for the participants to use. The players usually tried out all the commands at the very beginning of the game before doing anything else. Afterwards a tendency could be observed that the longer they played the more often the feature was used. It looked as if the players were trying to get used to it.

### Quality

Watching the players play or recordings of their gameplay helped determine the overall quality of speech recognition. The speech recognizer usually struggled with heavy Scottish accents, however, once the players noticed that, they slightly adjusted the way they spoke and this generally reduced the amount of misunderstandings. One of the most common mistakes recognizer produced was when the players determined the enemy to kill by defining it using the "further" word.

### Engagement

This topic related to how the players used the speech recognition technology in regards of how they communicated their needs to the Ally. Generally speaking, when the participants got used to the fact that they have an Ally they can speak to, exchange of sentences could be observed. Although at a simple level, the players got more engaged in communicating with the Ally and would create natural conversations.

### 5.3 Ethical Issues

Without any doubt, using voice recognition technology requires to value player's privacy. Since the speech recognition system was used by calling an external API, that belongs to Microsoft, the voice data was needed to be shared between the game and the API in a secure and private manner. Furthermore, certain regulations such as General Data Protection Regulation requires voice data to be stored in a way that the user can request to have it deleted due to being considered as personal data (GDPR, 2016). Due to that matter, all the voice recordings were named in a way to be easily identified which allowed them afterwards to be removed if requested to do so.

## 6 Conclusion and Further Work

This chapter covers the outcome of the evaluated results and provides recommendations on possible further work and expansion of the project.

### 6.1 Conclusion

The results can be considered reasonably trustworthy as they should not be biased by players' gaming expertise level since it was spread out quite evenly. Moreover, most players usually use headphones when playing which indicates that the speech recognition integrated into a game could be widely accessible to use for the majority of the players. Without doubt, both groups enjoyed having an Ally feature that they can communicate with which might be considered sufficient to give the players more involving gaming experience, however, it is undoubtedly the speech recognition technology used along with it that helps make the most out of it. Both groups mentioned that it was easy to communicate with the Ally, which means that using voice is as straightforward as when using keys. The game lacking such feature was not scored well in terms of recommending it to the participants' friends. On the contrary, even though all the players using speech recognition had at least one issue with the recognizer not recognizing their speech, group two is "Very likely" to recommend the game to their friends. It is worth taking into account the fact that during the gameplay and recording analysis it was observed that issues with speech recognition recognizing players' utterances seemed to occur more often than marked on the questionnaires. This might indicate that the feature was so enjoyable and involving that the players subconsciously did not notice the errors so much. This implies that the players did have a more immersive game experience and felt as being a part of the game. The comparison of both questionnaires and the recordings clearly show that the speech recognition technology is something that was

the advantage of the game and was appreciated by the participants. Especially considering that all participants from group 2 might be influenced to buy a game based on having speech recognition included in it.

### 6.2 Further Work

Recommendations on further work would definitely include expanding the available commands for the players to use, even the ones that do not affect the story of the game. This would allow the players to create more natural conversations with the Ally. Moreover, a more advanced system for classifying the intent of the user could be used, preferably one that is based on deep learning. Finally, the research can be extended to other game genres such us Role Playing Games or even Sport games.

## References


Abdel-Hamid, O., Mohamed, A., Jiang, H., Deng, L., Penn, G., Yu, D., 2014 Convolutional Neural Networks for Speech Recognition, [online], [viewed 22 November 2020] IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING **22**, pp.1533-1545. Available from: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN_ASLPTrans2-14.pdf

AbilityNet, [viewed 24 October 2020]. Available from: https://abilitynet.org.uk/factsheets/voice-recognition-overview

Amberkar, A., Awasarmol, P., Deshmukh, G., Dave, P., 2018, Speech Recognition using Recurrent Neural Networks [online], [viewed 22 November 2020]. Available from: https://www.researchgate.net/publication/329316345_Speech_Recognition_using_Recurrent_Neural_Networks

App Store, Apple Inc., Battlelands Royale [online], [viewed 22 November 2020]. Available from: https://apps.apple.com/us/app/battlelands-royale/id1296181302

Bowen, N., 2018, THE INPATIENT [online], [viewed 22 November 2020]. Available from: https://www.supermassivegames.com/games/the-inpatient

Brownlee, J., 2019, How Do Convolutional Layers Work in Deep Learning Neural Networks? [online], [viewed 22 November 2020]. Available from: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

Businesswire, 2008, Ubisoft Revolutionizes Gaming With Voice Command in Tom Clancy's EndWar™ [online], [viewed 22 November 2020]. Available from: https://www.businesswire.com/news/home/20081106005518/en/Ubisoft-Revolutionizes-Gaming-With-Voice-Command-in-Tom-Clancys-EndWar-TM

Butt, A., 2020, Top Voice Search Statistics, Facts And Trends For 2020 [online], [viewed 22 November 2020]. Available from: https://quoracreative.com/article/voice-search-statistics-trends

Caroux, L., Isbister, K., Bigot, L. & Vibert, N., 2015, Player–video game interaction: A systematic review of current concepts [online], [viewed 22 November 2020], Computers in Human Behavior pp.366-381. Available from: https://www-sciencedirect-com.gcu.idm.oclc.org/science/article/pii/S0747563215000941

Carr, L., 1994, The strengths and weaknesses of quantitative and qualitative research: what method for nursing? [online], [viewed 22 November 2020]. Available from: https://pdfs.semanticscholar.org/a87b/ce9f2d5fe771005a2890c92da2cff8a03b32.pdf

Chin, M., 2018, The First Alexa Board Game Is Both Fun and Terrifying [online]. [viewed 24 October 2020]. Available from: https://www.tomsguide.com/us/when-in-rome-alexa-board-game-review,news-27560.html

Cuelogic Insights, 2017, The Levenshtein Algorithm [online], [viewed 17 April 2021]. Available from: https://www.cuelogic.com/blog/the-levenshtein-algorithm

Dawson, M., Burrel, D., Rahim, E., 2010, Integrating Software Assurance into the Software Development Life Cycle (SDLC) [online], [viewed 22 November 2020]. Available from: https://www.researchgate.net/figure/IBM-System-Science-Institute-Relative-Cost-of-Fixing-Defects_fig1_255965523

Del Sole, A., 2017, Introducing Microsoft Cognitive Services [online], [viewed 22 November 2020], Microsoft Computer Vision APIs Distilled pp.1-4, Available from: https://link-springer-com.gcu.idm.oclc.org/chapter/10.1007%2F978-1-4842-3342-9_1

Doppio, The 3% Challenge [online], [viewed 22 November 2020]. Available from: https://doppio.games/three-percent

General Data Protection Regulation, 2016, Article 4.1 [online]. [viewed 24 October 2020]. Available from: https://gdpr-info.eu/art-4-gdpr/

Gerling, K., Birk, M., Mandryk, R., 2013, The Effects of Graphical Fidelity on Player Experience [online], [viewed 22 November 2020]. Available from: https://www.researchgate.net/publication/262288972_The_Effects_of_Graphical_Fidelity_on_Player_Experience

Gough, C., 2020, Genre breakdown of video game sales in the United States in 2018 [online], [viewed 22 November 2020], Available from: https://www.statista.com/statistics/189592/breakdown-of-us-video-game-sales-2009-by-genre/

Grabianowski, E. How Speech Recognition Works [online]. [viewed 24 October 2020]. Available from: https://electronics.howstuffworks.com/gadgets/high-tech-gadgets/speech-recognition.htm

Huang, X., Acero, A., Hon, H., Spoken Language Processing A Guide to Theory, Algorithm, and System Development [online], [viewed 20 April 2021]. Available from: https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/Spoken%20Language%20Processing_%20A%20Guide%20to%20Theory%2C%20Algorithm%2C%20and%20System%20Development%20%5BHuang%2C%20Acero%20%26%20Hon%202001-05-05%5D.pdf

Jagneaux, D., 2017, PSX 2017 Hands-On: The Inpatient Uses Voice Recognition For In-Game Dialog [online]. [viewed 24 October 2020]. Available from: https://uploadvr.com/psx-2017-inpatient-voice-preview/

Jang, Y., Park, E., 2019, An adoption model for virtual reality games: The roles of presence and enjoyment [online], [viewed 22 November 2020], Telematics and Informatics **42.** Available from: https://www-sciencedirect-com.gcu.idm.oclc.org/science/article/pii/S0736585319301315

Kalchbrenner, N., Grefenstette, E. & Blunsom, P., 2019, A Convolutional Neural Network for Modelling Sentences [online], [viewed 22 November 2020]. Available from: https://arxiv.org/pdf/1404.2188.pdf

Kikel, C., How Voice Recognition Technology Works [online], [viewed 22 November 2020]. Available from: https://www.totalvoicetech.com/how-voice-recognition-technology-works/

Krompiec, P., Park, K., 2019, Enhanced Player Interaction Using Motion Controllers for First-Person Shooting Games in Virtual Reality [online], [viewed 22 November 2020], IEEE Access **7,** pp.124548-124557. Available from: https://ieeexplore.ieee.org/abstract/document/8817959

Kulshreshtha, N., 2020, How Does Voice Recognition Work: Secret Behind Your Voice Assistants [online], [viewed 22 November 2020]. Available from: https://blogs.fireflies.ai/how-does-voice-recognition-work/

Lazaro, I., 2017, Agile Development: A quick overview [online], [viewed 22 November 2020]. Available from: https://medium.com/theagilemanager/a-quick-overview-to-agile-5c87ffc9e0f2

Lu, L., 2016, Sequence training and adaptation of highway deep neural networks [online], [viewed 22 November 2020], 2016 IEEE Spoken Language Technology Workshop (SLT) , San Diego, CA, 2016, pp. 461-466. Available from: https://ieeexplore-ieee-org.gcu.idm.oclc.org/document/7846304

Murnane, K., 2018, Graphics And Gameplay Are About Mutual Interaction, Not Relative Importance [online], [viewed 22 November 2020]. Available from: https://www.forbes.com/sites/kevinmurnane/2018/02/06/graphics-and-gameplay-are-about-mutual-interaction-not-relative-importance/?sh=7469d58814e5

Nacke, L., Grimshaw, M., 2011, Player-game interaction through affective sound [online], [viewed 22 November 2020]. Available from: http://ubir.bolton.ac.uk/226/1/gcct_chapters-3.pdf

Nam, E., 2019, Understanding the Levenshtein Distance Equation for Beginners [online], [viewed 17 April 2021]. Available from:

https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0

Ong, T., 2018, Lego's new Alexa skill has voice-guided instructions and tells stories [online], [viewed 22 November]. Available from: https://www.theverge.com/2018/5/3/17314100/lego-duplo-stories-interactive-skill-amazon-alexa

Parker, J., 2020, What is Force Feedback in Racing Wheels? Do you need it? [online], [viewed 17 April 2021]. Available from: https://ezsimracer.com/force-feedback/

Reczko, M., Thireou, T., 2007, Bidirectional Long Short-Term Memory Networks for Predicting the Subcellular Localization of Eukaryotic Proteins [online], [viewed 22 November 2020]. Available from: https://www.researchgate.net/publication/6172053_Bidirectional_Long_Short-Term_Memory_Networks_for_Predicting_the_Subcellular_Localization_of_Eukaryotic_Proteins#

Rouse, M., 2018, voice recognition (speaker recognition) [online]. [viewed 24 October 2020]. Available from: https://searchcustomerexperience.techtarget.com/definition/voice-recognition-speaker-recognition

Schoenau-Fog, H., 2011,The Player Engagement Process – An Exploration of Continuation Desire in Digital Games. Proceedings of DiGRA 2011 Conference: Think Design Play [online]. [viewed 24 October 2020]. Available from: http://www.digra.org/wp-content/uploads/digital-library/11307.06025.pdf

Schuster, M., Paliwal, K., 1997, Bidirectional recurrent neural networks [online], [viewed 22 November 2020], IEEE Transactions on Signal Processing **45,** pp. 2673-2681. Available from: https://ieeexplore-ieee-org.gcu.idm.oclc.org/document/650093

Shinoda K., 2005, Speaker Adaptation Techniques for Speech Recognition UsingProbabilistic Models [online], [viewed 22 November 2020], Electronics and Communications in Japan, Part 3 **88**, pp. 371-386. Available from: https://onlinelibrary-wiley-com.gcu.idm.oclc.org/doi/epdf/10.1002/ecjc.20207

Stuart, K. 2019, Call of Duty: Modern Warfare review – great game, shame about the politics [online]. [viewed 01 November 2020]. Available from: https://www.theguardian.com/games/2019/oct/31/call-of-duty-modern-warfare-review

Stuart, K., 2019 Battle royale: the design secrets behind gaming's biggest genre [online], [viewed 22 November 2020]. Available from: https://www.theguardian.com/games/2019/feb/23/battle-royale-games-design-fortnite-pubg-call-of-duty

Suarez, J., The Importance Of Narrative In Video Games [online], [viewed 22 November 2020], Available from: https://wibbu.com/importance-narrative-video-games/

Takahashi, D., 2019, Netflix and Doppio Games partner to create voice-controlled game The 3% Challenge [online], [viewed 22 November 2020]. Available from: https://venturebeat.com/2019/10/08/netflix-and-doppio-games-partner-to-create-voice-controlled-games-and-entertainment/

Tankovska, H., 2020, Virtual reality (VR) market revenue in the United States from 2014 to 2025 [online], [viewed 22 November 2020]. Available from: https://www.statista.com/statistics/784139/virtual-reality-market-size-in-the-us/

Technopedia, 2011, First Person Shooter (FPS) [online], [viewed 22 November 2020]. Available from: https://www.techopedia.com/definition/241/first-person-shooter-fps

Vailshery, J., Unit shipments of virtual reality (VR) devices worldwide from 2017 to 2019 (in millions), by vendor [online]. [viewed 22 April 2021]. Available from: https://www.statista.com/statistics/671403/global-virtual-reality-device-shipments-by-vendor/

Virsabi, Everything you have to know about haptic technology [online], [viewed 17 April 2021].
Available from: https://virsabi.com/everything-about-haptic-technology/

Weber, R., Behr, K., Tamborini, R., Ritterfeld, U. & Mathiak, K. 2009, What Do We Really Know about First-Person-Shooter Games? an Event-Related, High-Resolution Content Analysis [online], [viewed 22 November 2020], Journal of Computer-Mediated Communication pp.1016-1037. Available from: https://academic.oup.com/jcmc/article/14/4/1016/4583562

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X. & Stolcke, A., 2017, THE MICROSOFT 2017 CONVERSATIONAL SPEECH RECOGNITION SYSTEM [online], [viewed 22 November 2020]. Available from: https://www.microsoft.com/en-us/research/wp-content/uploads/2017/08/ms_swbd17-2.pdf

# Appendices

**Appendix 1 – Google Form questionnaire for group 1**
Available from: https://forms.gle/6ZkMmw8QVKQd2j5NA

# Speech Recognition in FPS games Questionnaire

The following research has been constructed in order to investigate the impact of integrating voice recognition technology on players' game experience in First Person Shooter games. The data gathered from the questionnaire will remain strictly confidential and will be used solely for the purpose of the investigation by self-directed research.

Experiment
In this experiment, you will be required to play a First Person Shooter game that does not use speech recognition technology. Before starting, please fill out the "About your gamin experience" section. After finishing your game, you will need to fill out the "Playing without speech recognition" part of this questionnaire. You will then be asked to play a similar game that includes using speech recognition technology. Finally, you will need to fill out the last section of the questionnaire.

Eligibility
In order to take part in this experiment you must be over the age of 18, have access to internet and be willing to have your gameplay (video, audio) recorded.

Participation
At any time during the experiment you are free to withdraw your participation without providing a reason. Any collected data or filled out questionnaires will be deleted.

Data
Both visual and audio will be recorded during your game. This will be used in order to assess how speech recognition technology performs and how it is used by you.

Consent
- You agree to take part in the experiment and have your screen and audio recorded. The only collected personal data will be your voice which will be stored securely and you can request it to be deleted at any time without providing a reason.
- You agree to have the gameplay and questionnaire responses saved for self-directed research.

Contact
If you have any questions or want to request your data being removed please see the contact details below:

Dawid Kubiak
dkubia200@caledonian.ac.uk

*Wymagane

Do you agree to the above statement? *

- ◯ Yes
- ◯ No

Dalej

# Speech Recognition in FPS games Questionnaire

*Wymagane

## About your gaming experience

How would you rate yourself in regards to being a gamer? *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Casual Gamer | ○ | ○ | ○ | ○ | ○ | Hardcore Gamer |

How would you rate First Person Shooter games? *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Don't like them at all | ○ | ○ | ○ | ○ | ○ | Just love them - it's the only type of games I play |

Which equipment do you usually use when playing? *

- ○ Headphones
- ○ Speakers
- ○ Speakers and microphone
- ○ Inne: _____

Wstecz Dalej

# Speech Recognition in FPS games Questionnaire

*Wymagane

## Playing without speech recognition

Complete if you've played the game excluding speech recognition technology

After being provided with the brief description of the game's aim, how clear was it for you play the game from the start to the end? *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Not clear at all | ○ | ○ | ○ | ○ | ○ | Very clear |

Did you enjoy the feature of having an ally in the game that you can comunicate with? *

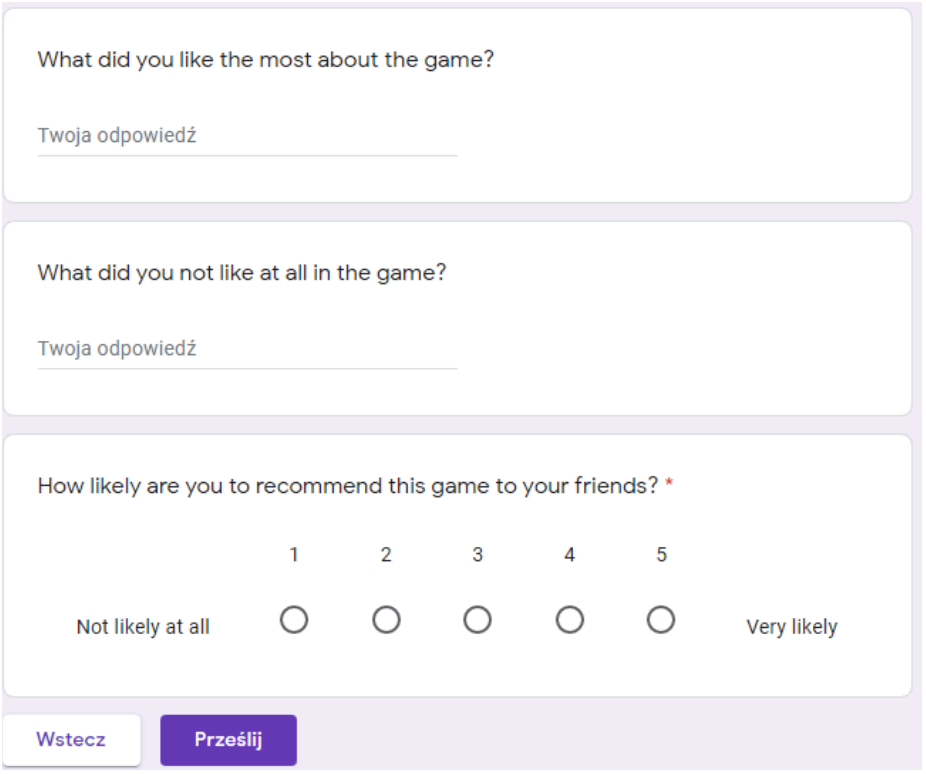
- ○ Yes
- ○ No
- ○ I did not communicate with the Ally

Was it easy for you to communicate with the ally? *

- ○ Yes
- ○ No
- ○ I did not communicate with the Ally

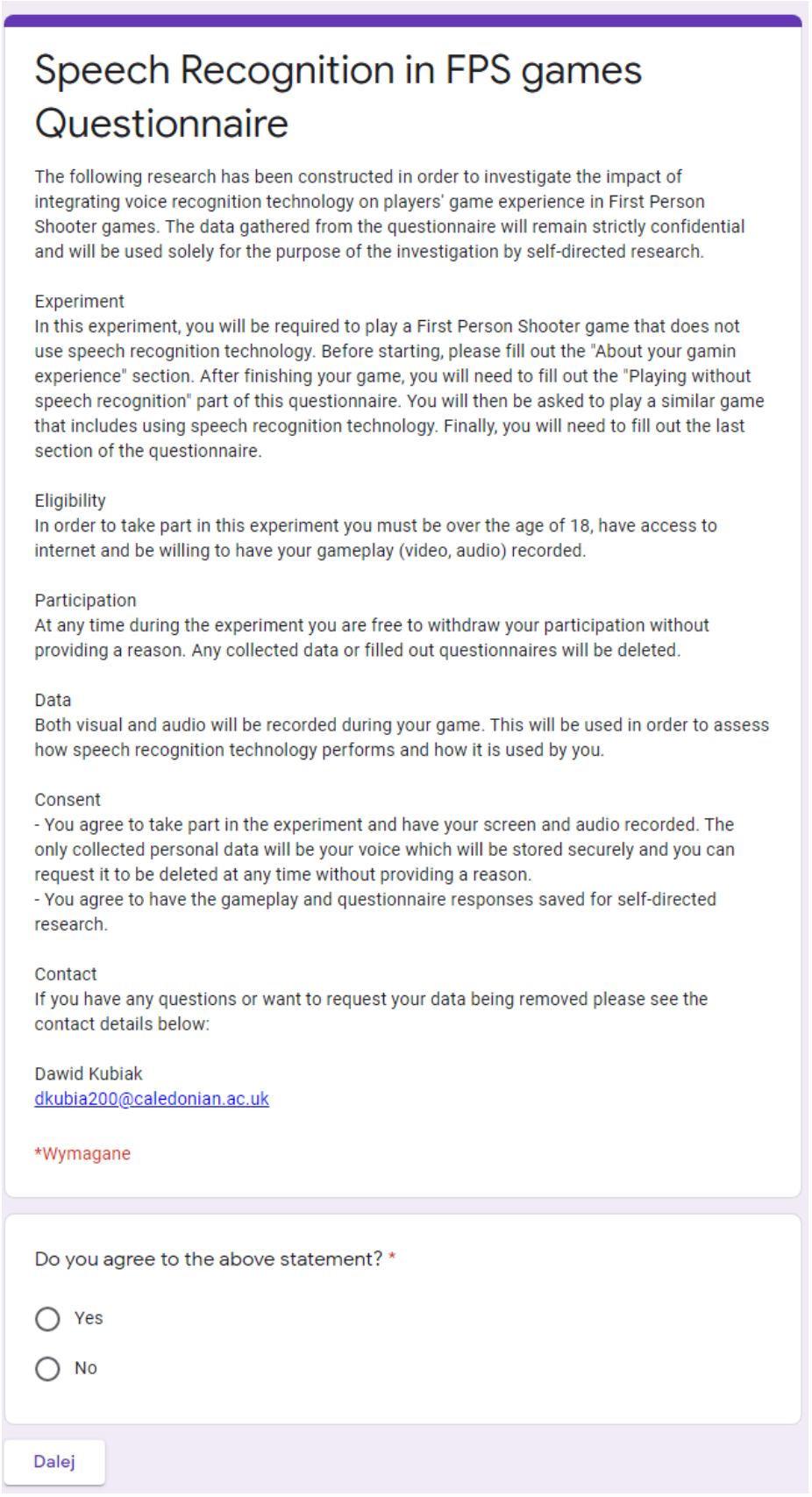
Without taking the visual effects into account, did you feel you had a realistic experience playing the game? *

- ○ Yes
- ○ Maybe
- ○ No


What did you like the most about the game?
Twoja odpowiedź
What did you not like at all in the game?
Twoja odpowiedź
How likely are you to recommend this game to your friends? *
1
2
3
4
5
Not likely at all
Very likely
Wstecz
Prześlij

**Appendix 2 – Google Form questionnaire for group 2**

Available from: https://forms.gle/5y5bAi23vn8KmKAt9



# Speech Recognition in FPS games Questionnaire

The following research has been constructed in order to investigate the impact of integrating voice recognition technology on players' game experience in First Person Shooter games. The data gathered from the questionnaire will remain strictly confidential and will be used solely for the purpose of the investigation by self-directed research.

Experiment
In this experiment, you will be required to play a First Person Shooter game that does not use speech recognition technology. Before starting, please fill out the "About your gamin experience" section. After finishing your game, you will need to fill out the "Playing without speech recognition" part of this questionnaire. You will then be asked to play a similar game that includes using speech recognition technology. Finally, you will need to fill out the last section of the questionnaire.

Eligibility
In order to take part in this experiment you must be over the age of 18, have access to internet and be willing to have your gameplay (video, audio) recorded.

Participation
At any time during the experiment you are free to withdraw your participation without providing a reason. Any collected data or filled out questionnaires will be deleted.

Data
Both visual and audio will be recorded during your game. This will be used in order to assess how speech recognition technology performs and how it is used by you.

Consent
- You agree to take part in the experiment and have your screen and audio recorded. The only collected personal data will be your voice which will be stored securely and you can request it to be deleted at any time without providing a reason.
- You agree to have the gameplay and questionnaire responses saved for self-directed research.

Contact
If you have any questions or want to request your data being removed please see the contact details below:

Dawid Kubiak
dkubia200@caledonian.ac.uk

*Wymagane

Do you agree to the above statement? *

- ◯ Yes
- ◯ No

Dalej

# Speech Recognition in FPS games Questionnaire

*Wymagane

## About your gaming experience

How would you rate yourself in regards to being a gamer? *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Casual Gamer | ○ | ○ | ○ | ○ | ○ | Hardcore Gamer |

How would you rate First Person Shooter games? *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Don't like them at all | ○ | ○ | ○ | ○ | ○ | Just love them - it's the only type of games I play |

Which equipment do you usually use when playing? *

- ○ Headphones
- ○ Speakers
- ○ Speakers and microphone
- ○ Inne: _____

Wstecz Dalej

# Speech Recognition in FPS games Questionnaire

*Wymagane

## Playing with speech recognition

Complete if you've played the game with speech recognition technology included

After being provided with the brief description of the game's aim, how clear was it for you play the game from the start to the end? *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Not clear at all | ○ | ○ | ○ | ○ | ○ | Very clear |

Did you enjoy the feature of having an ally in the game that you can speak to? *

- ○ Yes
- ○ No
- ○ I did not speak to the Ally

Was it easy for you to communicate with the ally? *

- ○ Yes
- ○ No
- ○ I did not speak to the Ally

How often did you speak to the ally? *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very rarely (couple times throughout the whole game) | ○ | ○ | ○ | ○ | ○ | All the time |

How often did you encounter issues with communication using the speech recognition technology? *

For example, the ally had problems understanding your questions / intentions.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Not at all | ○ | ○ | ○ | ○ | ○ | All the time |

Without taking the visual effects into account, did you feel you had a realistic experience playing the game? *

- ○ Yes
- ○ Maybe
- ○ No

What did you like the most about the game?

Twoja odpowiedź

What did you not like at all in the game?

Twoja odpowiedź

How likely are you to recommend this game to your friends? *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Not likely at all | ○ | ○ | ○ | ○ | ○ | Very likely |

Would use of speech recognition technology in game influence your purchasing decision when buying a game? *

- ○ Yes
- ○ Maybe
- ○ No

Wstecz Prześlij

**Appendix 3 – List of assets**

Nature models

https://opengameart.org/content/nature-kit

Landscape models

https://opengameart.org/content/landscape-asset-v1

Building models

https://opengameart.org/content/medieval-town-base-3d-assets

Player model

https://free3d.com/3d-model/free-low-poly-soldier-28299.html

Enemy soldier model

https://free3d.com/3d-model/low-poly-rigs-soldier-2319.html

Boss soldier model

https://free3d.com/3d-model/low-poly-rigs-soldier-25083.html

Fire visual effect

https://assetstore.unity.com/packages/3d/props/the-free-medieval-and-war-props-174433

Standard Unity assets

https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2018-4-32351

Flashlight model

https://assetstore.unity.com/packages/3d/props/tools/flashlight-pro-53053

Night vision shader

https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/deferred-night-vision-43423

Gun model

https://www.turbosquid.com/3d-models/futuristic-rifle-suppressor-max/818288

Other models

https://assetstore.unity.com/packages/3d/props/the-free-medieval-and-war-props-174433

Player shot sound effect

https://www.youtube.com/watch?v=FO_MYdT65Ns&list=PLQW1qYrbyrLsPQ9spZXyEWBq6EG4rU36G&index=5

Ally shot sound effect

https://www.youtube.com/watch?v=ixbpOC7EI9Q

**Appendix 4 – Full project**

https://github.com/dejwkubikson/Speech-Recognition-in-FPS-Games

**Appendix 5 – EnemyBehaviour.cs**

https://github.com/dejwkubikson/Speech-Recognition-in-FPS-Games/blob/main/Honours%20Project/Assets/Scripts/EnemyBehaviour.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyBehaviour : MonoBehaviour
{
    public Vector3[] moveToPoints = { new Vector3(-4, 0, 6), new Vector3(-8, 0, 6),
new Vector3(-10, 0, 10) };
    public bool moveToPoint = false;
    public bool pointReached = false;
    private int currPoint = 1;

    public float speed = 0.2f;

    public float timeMin = 3.0f;
    public float timeMax = 10.0f;
    private float randomWaitTime = 0.0f;

    public float health = 100;
    public bool playerSeen = false;
    private float playerSeenTimer = 0.0f;

    private bool startDmgTimer = false;
    private float dmgTimer = 2.0f;

    public bool bossSoldier = false;

    private GameObject playerObject;
    public PlayerScript playerScript;

    public bool dead = false;

    GameControl gameControlScript;

    public bool hasFlashLight = false;
    public GameObject flashLight;

    public bool deadEnemySeen = false;
    public GameObject deadEnemy;

    public GameObject wrapper;

    public Animator animator;

    private void Start()
    {
        gameControlScript =
GameObject.FindGameObjectWithTag("GameController").GetComponent<GameControl>();
        playerObject = GameObject.FindGameObjectWithTag("Player");
```

```
        playerScript =
playerObject.transform.GetChild(0).GetComponent<PlayerScript>();
        randomWaitTime = Random.Range(timeMin, timeMax);

        if (gameObject.name == "BossSoldier")
        {
            bossSoldier = true;
        }

        if (bossSoldier)
            return;

        flashLight.SetActive(hasFlashLight);
        animator = wrapper.GetComponent<Animator>();
        animator.enabled = false;
    }

    // Update is called once per frame
    void Update()
    {
        if (dead)
            return;


        if (bossSoldier)
            return;

        deadEnemySeen = CheckIfVisibleEnemyDead();

        if (CheckIfPlayerVisible())
        {
            playerSeen = true;

            RotateTowardsPoint(playerObject.transform.position);

            playerSeenTimer += Time.deltaTime;
            playerScript.inSight = true;
            playerScript.seenTimer = playerSeenTimer;

            if (playerSeenTimer > gameControlScript.timeToLose)
                gameControlScript.GameLost("The enemy spotted you.");
        }
        else
        {
            playerSeenTimer = 0;
            playerSeen = false;
        }

        if (deadEnemySeen && deadEnemy != null)
        {
            startDmgTimer = true;
            if(playerSeen == false)
                RotateTowardsPoint(deadEnemy.transform.position);
        }

        if (startDmgTimer)
        {
            dmgTimer -= Time.deltaTime;

            if (dmgTimer <= 0)
            {
                if(deadEnemySeen)
```

```
                    gameControlScript.GameLost("The enemy raised an alarm after
noticing a dead soldier.");
                else
                    gameControlScript.GameLost("The enemy raised an alarm after being
shot.");
            }
        }

        if(pointReached)
        {
            randomWaitTime = Random.Range(timeMin, timeMax);
            pointReached = false;

            if (currPoint < moveToPoints.Length)
            {
                currPoint++;
            }
            else
            {
                currPoint = 1;
            }
        }

        if (moveToPoints.Length > 0)
        {
            if (randomWaitTime <= 0)
            {
                moveToPoint = true;
                animator.enabled = true;
            }
            else if (moveToPoint == false)
            {
                randomWaitTime -= Time.deltaTime;

                animator.enabled = false;
            }
        }

        MoveToPoint(speed * Time.deltaTime);
    }

    void MoveToPoint(float step)
    {
        if(moveToPoint && playerSeen == false && deadEnemySeen == false &&
moveToPoints.Length > 0)
        {
            RotateTowardsPoint(moveToPoints[currPoint - 1]);
            transform.position = Vector3.MoveTowards(transform.position,
moveToPoints[currPoint - 1], step);
            if(Vector3.Distance(transform.position, moveToPoints[currPoint - 1]) <
0.01f)
            {
                moveToPoint = false;
                pointReached = true;
                randomWaitTime = 0.0f;
            }
        }
    }

    void RotateTowardsPoint(Vector3 pointToLookAt)
    {
        Vector3 targetDirection = pointToLookAt - transform.position;
```

```
        Vector3 newDirection = Vector3.RotateTowards(transform.forward,
targetDirection, 1, 0.0f);

        transform.rotation = Quaternion.LookRotation(newDirection);
    }

    public void ReceiveDamage(float amount)
    {
        startDmgTimer = true;
        health -= amount;

        if (health <= 0)
            Die();
        else
            RotateTowardsPoint(playerObject.transform.position);
    }

    void Die()
    {
        playerSeen = false;

        if (bossSoldier)
            gameControlScript.GameWon();

        gameObject.transform.eulerAngles = new Vector3(-90, transform.eulerAngles.y,
transform.eulerAngles.z);
        gameObject.transform.GetChild(0).gameObject.SetActive(false);
        gameObject.GetComponent<BoxCollider>().enabled = true;
        animator.enabled = false;
        dead = true;
    }

    bool CheckIfVisibleEnemyDead()
    {
        GameObject[] enemyObjects = GameObject.FindGameObjectsWithTag("Soldier");
        for(int i = 0; i < enemyObjects.Length; i++)
        {
            GameObject enemyObject = enemyObjects[i];
            Vector3 direction = enemyObject.transform.position - transform.position;
            float angleToEnemy = Vector3.Angle(direction, transform.forward);

            float distToEnemy = Vector3.Distance(enemyObject.transform.position,
transform.position);

            //Debug.Log(enemyObjects[i].name + " angle " + angleToEnemy + " dist " +
distToEnemy);

            if ((angleToEnemy >= -45 && angleToEnemy <= 45 && distToEnemy < 3f) ||
distToEnemy < 0.8f)
            {
                RaycastHit hit;
                if(Physics.Raycast(transform.position + new Vector3(0, 0.01f, 0),
direction, out hit, 4))
                {
                    Debug.DrawRay(transform.position + new Vector3(0, 0.01f, 0),
direction * hit.distance, Color.yellow);

                    if(hit.transform.tag == "Soldier")
                    {
                        EnemyBehaviour enemyBehaviour =
enemyObject.GetComponent<EnemyBehaviour>();
                        if (enemyBehaviour != null)
```

```
                        {
                            if (enemyBehaviour.dead)
                            {
                                deadEnemy = enemyObject;
                                return true;
                            }
                        }
                    }
                }
            }

            deadEnemy = null;
            return false;
        }

        bool CheckIfPlayerVisible()
        {
            Vector3 targetDir = playerObject.transform.position - transform.position;

            float angleToPlayer = Vector3.Angle(targetDir, transform.forward);

            float distToPlayer = Vector3.Distance(playerObject.transform.position,
transform.position);

            if ((angleToPlayer >= -45 && angleToPlayer <= 45 && distToPlayer < 2.5f) ||
distToPlayer < 0.8f)
            {
                RaycastHit hit;
                if (Physics.Raycast(transform.position + new Vector3(0, 0.07f, 0),
targetDir, out hit, 4))
                {
                    Debug.DrawRay(transform.position + new Vector3(0, 0.07f, 0), targetDir
* hit.distance, Color.yellow);

                    if(hit.transform == playerObject.transform)
                    {
                        return true;
                    }
                    else
                    {
                        return false;
                    }
                }
                else
                {
                    Debug.DrawRay(transform.position, targetDir * 1000, Color.white);
                    return false;
                }
            }
            else
                return false;
        }
}
```

**Appendix 6 – Cognitive Services Speech SDK**

https://github.com/Azure-Samples/cognitive-services-speech-sdk/blob/master/samples/csharp/unity/VirtualAssistantPreview/README.md

**Appendix 7 – SpeechRecognition.cs**

https://github.com/dejwkubikson/Speech-Recognition-in-FPS-Games/blob/main/Honours%20Project/Assets/Scripts/SpeechRecognition.cs

```
using System;
using System.Collections;
using System.Diagnostics;
using System.Globalization;
using System.Threading.Tasks;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;

public class SpeechRecognition : MonoBehaviour
{
    // Text objects to display recognized text and errors
    public Text RecognizedText;
    public Text ErrorText;
    private VoiceControl voiceControl;

    private string recognizedString = "";
    private string errorString = "";

    private bool recognizeText = false;
    private bool recognizedText = false;

    public float textClearTimer = 5.0f;

    // Recognition events are raised in seperate thread. Thread must be locked
    // to avoid deadlocks.
    private System.Object threadLock = new System.Object();

    // Object used for speech recognition
    private SpeechRecognizer recognizer;

    // Start is called before the first frame update
    void Start()
    {
        voiceControl =
GameObject.FindGameObjectWithTag("VoiceControl").GetComponent<VoiceControl>();

        StartRecognition();
    }

    // Update is called once per frame
    void Update()
    {
        if(Input.GetKey(KeyCode.T))
        {
            recognizeText = true;
            recognizedText = false;
        }
        else
        {
            recognizeText = false;
        }
```

```
        lock (threadLock)
        {
            if (recognizedString.Length > 0 && recognizedString.Contains("GATHERING")
== false)
                RecognizedText.text = $"Alpha: {recognizedString}";
            else
                RecognizedText.text = "";
            /*else if (recognizedString.Contains("GATHERING"))
                RecognizedText.text = recognizedString;*/

            if(errorString.Length > 0)
                ErrorText.text = $"ERROR: {errorString}";
        }

        if(RecognizedText.text.Length > 0)
        {
            textClearTimer -= Time.deltaTime;
            if (textClearTimer <= 0)
            {
                recognizedString = "";
                textClearTimer = 5.0f;
            }
        }

    }

    void CreateRecognizer()
    {
        UnityEngine.Debug.Log("CreateRecognizer()");

        if(recognizer == null)
        {
            SpeechConfig config =
SpeechConfig.FromSubscription("c0b086eaf0394ee1ad212ab0d731d22e", "uksouth");
            config.SpeechRecognitionLanguage = "en-us";
            recognizer = new SpeechRecognizer(config);

            if(recognizer != null)
            {
                // Speech events
                recognizer.Recognizing += RecognizingHandler;
                recognizer.Recognized += RecognizedHandler;
                recognizer.Canceled += CanceledHandler;
            }
        }

        UnityEngine.Debug.Log("CreateRecognizer() finished");
    }

    private async void StartRecognition()
    {
        UnityEngine.Debug.Log("StartRecognition()");
        CreateRecognizer();

        if(recognizer != null)
        {
            UnityEngine.Debug.Log("Recognizer found, starting");
            await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

            UnityEngine.Debug.Log("Recognizer started");
        }
```

```
        UnityEngine.Debug.Log("StartRecognition() finished");
    }

    // Stops the speech recognition.
    // Releasing all events and cleaning up resources
    public async void StopRecognition()
    {
        if (recognizer != null)
        {
            await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
            recognizer.Recognizing -= RecognizingHandler;
            recognizer.Recognized -= RecognizedHandler;
            recognizer.Canceled -= CanceledHandler;
            recognizer.Dispose();
            recognizer = null;
            recognizedString = "Speech recognition stopped.";
        }
    }

    // Speech events
    // "Recognizing" events are fired every time interim results are returned during
recognition
    private void RecognizingHandler(object sender, SpeechRecognitionEventArgs e)
    {
        //UnityEngine.Debug.Log("RecognizingHandler() start");
        if (!recognizeText)
            return;
        //UnityEngine.Debug.Log("RecognizingHandler() pass");

        if (e.Result.Reason == ResultReason.RecognizingSpeech)
        {
            lock (threadLock)
            {
                recognizedString = $"GATHERING: {e.Result.Text}";
            }
        }
    }

    // "Recognized" events are fired when the utterance end was detected by the server
    private void RecognizedHandler(object sender, SpeechRecognitionEventArgs e)
    {
        if (recognizedText)
            return;
        else
            recognizedText = true;

        if (e.Result.Reason == ResultReason.RecognizedSpeech)
        {
            lock (threadLock)
            {
                recognizedString = e.Result.Text;
                voiceControl.CheckKeywordsAndFireEvents(recognizedString);

                textClearTimer = 5.0f;
            }
        }
        else if (e.Result.Reason == ResultReason.NoMatch)
        {
            UnityEngine.Debug.LogFormat($"RESULT: Could not recognize speech.");
        }
    }
```

```
    // "Canceled" events are fired if the server encounters some kind of error.
    private void CanceledHandler(object sender, SpeechRecognitionCanceledEventArgs e)
    {
        UnityEngine.Debug.Log("CanceledHandler()");
        errorString = e.ToString();
        UnityEngine.Debug.Log("CanceledHandler() finished");
    }
}
```

### Appendix 8 – VoiceControl.cs

https://github.com/dejwkubikson/Speech-Recognition-in-FPS-Games/blob/main/Honours%20Project/Assets/Scripts/VoiceControl.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.Linq;

using System.Threading.Tasks;

public class VoiceControl : MonoBehaviour
{
    public GameObject allyObject;
    public AllyScript allyScript;
    GameControl gameControl;

    Text commandsText;

    public bool waitForKillConfirmation = false;
    public bool waitForTargetSpecification = false;

    List<int> matchList = new List<int>();
    Dictionary<string, Action.actionType> actionsDict;
    static string[] actionKeywords = { "left", "right", "closer", "further", "near" };

    public class Action
    {
        public enum actionType{
            TellAboutEnemies,
            TellIfEnemyDown,
            TellWhereToGoWhatToDo,
            DetermineEnemyToKill,
            ConfirmTarget,
            KillEnemyOrder,
            ForceMissionFour
        }
    }

    // Start is called before the first frame update
    void Start()
    {
        SetUpActions();
        /*TestRecognitionLevenshteinDistance("What do I need to do");
        TestRecognitionMatchingKeywordsPercentage("What do I need to do");
        TestRecognitionLevenshteinDistance("What should I do");
        TestRecognitionMatchingKeywordsPercentage("What should I do");
        TestRecognitionLevenshteinDistance("I like enemies");
        TestRecognitionMatchingKeywordsPercentage("I like enemies");
        TestRecognitionLevenshteinDistance("Kill the enemy that is closer to me");
```

```
        TestRecognitionMatchingKeywordsPercentage("Kill the enemy that is closer to
me");
        TestRecognitionLevenshteinDistance("Kill the enemy that is on my left side");
        TestRecognitionMatchingKeywordsPercentage("Kill the enemy that is on my left
side");*/

        allyObject = GameObject.FindGameObjectWithTag("Ally");
        allyScript = allyObject.GetComponent<AllyScript>();
        gameControl =
GameObject.FindGameObjectWithTag("GameController").GetComponent<GameControl>();
        commandsText = GameObject.Find("CommandsText").GetComponent<Text>();

        if (gameControl.speechRecognition)
            DisplayPossibleCommands(new string[] { "Where do I go", "Where is the
checkpoint", "What do I do" });
        else
            DisplayPossibleCommands(new string[] { "Press [1]: Ask where are the
enemies.", "Press [2]: Ask what to do", "Press [3]: Confirm enemy status", "Press [4]:
Roger" });
    }

    public void TestRecognitionLevenshteinDistance(string textToCheck)
    {
        foreach (string key in actionsDict.Keys)
        {
            matchList.Add(LevenshteinDistance(key, textToCheck));
        }

        int pos = matchList.IndexOf(matchList.Min());
        string actionText = actionsDict.ElementAt(pos).Key;
        Debug.Log($"LevenshteinDistance: textToCheck: {textToCheck}
({textToCheck.Length}). Recognized {actionText} ({actionText.Length})." +
            $"with matchList.Min() equal to {matchList.Min()}. Matched in
{(actionText.Length - matchList.Min()) * 100 / actionText.Length}%");


        matchList.Clear();
    }

    public void TestRecognitionMatchingKeywordsPercentage(string textToCheck)
    {
        foreach (string key in actionsDict.Keys)
        {
            matchList.Add(MatchingKeywordsPercentage(key, textToCheck));
        }

        int pos = matchList.IndexOf(matchList.Max());
        string actionText = actionsDict.ElementAt(pos).Key;
        Debug.Log($"MatchingKeywords: textToCheck: {textToCheck}
({textToCheck.Split().Length}). Recognized {actionText}
({actionText.Split().Length})." +
            $"with matchList.Max() equal to {matchList.Max()}. Matched in
{matchList.Max()}% ");

        matchList.Clear();
    }

    void SetUpActions()
    {
        actionsDict = new Dictionary<string, Action.actionType>();

        // TellAboutEnemies();
```

```
        string[] enemyNames = { "target", "tango", "enemy", "soldier" };
        for (int i = 0; i < enemyNames.Length; i++)
        {
            string pluralName = enemyNames[i] == "enemy" ? "enemies" : enemyNames[i] +
"s";
            actionsDict.Add($"Where is {enemyNames[i]}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"Where is the {enemyNames[i]}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"Where are {pluralName}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"Where are the {pluralName}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"Are there any {pluralName}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"Are there any {pluralName} nearby",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"Can you see any {pluralName}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"Positions of {pluralName}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"What are the positions of {pluralName}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"Do you see any {pluralName}",
Action.actionType.TellAboutEnemies);
            actionsDict.Add($"How many {pluralName} do you see",
Action.actionType.TellAboutEnemies);
        }

        // TellIfEnemyDown();
        actionsDict.Add("", Action.actionType.TellIfEnemyDown);
        for (int i = 0; i < enemyNames.Length; i++)
        {
            string pluralName = enemyNames[i] == "enemy" ? "enemies" : enemyNames[i] +
"s";
            actionsDict.Add($"Confirm {enemyNames[i]} is killed",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Confirm {enemyNames[i]} is down",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Confirm {enemyNames[i]} is dead",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Confirm {enemyNames[i]} status",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Is {enemyNames[i]} dead",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Is {enemyNames[i]} down",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Is {enemyNames[i]} alive",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Are {pluralName} dead",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Are {pluralName} down",
Action.actionType.TellIfEnemyDown);
            actionsDict.Add($"Are {pluralName} alive",
Action.actionType.TellIfEnemyDown);
        }

        // TellWhereToGoWhatToDo();
        actionsDict.Add("Where do I go", Action.actionType.TellWhereToGoWhatToDo);
        actionsDict.Add("Where do I need to go",
Action.actionType.TellWhereToGoWhatToDo);
        actionsDict.Add("What do I do", Action.actionType.TellWhereToGoWhatToDo);
```

```
        actionsDict.Add("What do I need to do",
Action.actionType.TellWhereToGoWhatToDo);
        actionsDict.Add("What to do", Action.actionType.TellWhereToGoWhatToDo);
        actionsDict.Add("Where is checkpoint",
Action.actionType.TellWhereToGoWhatToDo);
        actionsDict.Add("Where is the checkpoint",
Action.actionType.TellWhereToGoWhatToDo);
        actionsDict.Add("What is mission", Action.actionType.TellWhereToGoWhatToDo);
        actionsDict.Add("What is the mission",
Action.actionType.TellWhereToGoWhatToDo);

        // DetermineEnemyToKill();
        for (int i = 0; i < enemyNames.Length; i++)
        {
            actionsDict.Add($"Kill the {enemyNames[i]} on left",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Kill the {enemyNames[i]} on my left",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Kill the {enemyNames[i]} on right",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Kill the {enemyNames[i]} on my right",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Kill the {enemyNames[i]} closer",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Kill the {enemyNames[i]} closer to me",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Kill the {enemyNames[i]} further",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Kill the {enemyNames[i]} further to me",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Kill the {enemyNames[i]} near me",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} on left",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} on my left",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} on right",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} on my right",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} closer",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} closer to me",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} further",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} further to me",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"I kill the {enemyNames[i]} near me",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Left {enemyNames[i]} is yours",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Right {enemyNames[i]} is yours",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"{enemyNames[i]} on the left is yours",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"{enemyNames[i]} on the right is yours",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Closer {enemyNames[i]} is yours",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Further {enemyNames[i]} is yours",
Action.actionType.DetermineEnemyToKill);
```

```
            actionsDict.Add($"{enemyNames[i]} closer to me is yours",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"{enemyNames[i]} further to me is yours",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Left {enemyNames[i]} is mine",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Right {enemyNames[i]} is mine",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"{enemyNames[i]} on the left is mine",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"{enemyNames[i]} on the right is mine",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Closer {enemyNames[i]} to me is mine",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"Further {enemyNames[i]} to me is mine",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"{enemyNames[i]} closer to me is mine",
Action.actionType.DetermineEnemyToKill);
            actionsDict.Add($"{enemyNames[i]} further to me is mine",
Action.actionType.DetermineEnemyToKill);
        }

        // ConfirmTarget();
        actionsDict.Add($"Who you aim at", Action.actionType.ConfirmTarget);
        for (int i = 0; i < enemyNames.Length; i++)
        {
            actionsDict.Add($"Confirm {enemyNames[i]}",
Action.actionType.ConfirmTarget);
            actionsDict.Add($"Confirm {enemyNames[i]} to kill",
Action.actionType.ConfirmTarget);
            actionsDict.Add($"Confirm {enemyNames[i]} to shoot",
Action.actionType.ConfirmTarget);
            actionsDict.Add($"Confirm your {enemyNames[i]}",
Action.actionType.ConfirmTarget);
            actionsDict.Add($"Which {enemyNames[i]} is your",
Action.actionType.ConfirmTarget);
            actionsDict.Add($"Which {enemyNames[i]} is yours",
Action.actionType.ConfirmTarget);
            actionsDict.Add($"What is your {enemyNames[i]}",
Action.actionType.ConfirmTarget);
            actionsDict.Add($"Which {enemyNames[i]} you kill",
Action.actionType.ConfirmTarget);
            actionsDict.Add($"Which {enemyNames[i]} you shoot",
Action.actionType.ConfirmTarget);
        }

        // KillEnemyOrder();
        actionsDict.Add("Shoot", Action.actionType.KillEnemyOrder);
        actionsDict.Add("Kill", Action.actionType.KillEnemyOrder);
        actionsDict.Add("Shoot to kill", Action.actionType.KillEnemyOrder);
        actionsDict.Add("Now", Action.actionType.KillEnemyOrder);
        actionsDict.Add("Fire", Action.actionType.KillEnemyOrder);
        for (int i = 0; i < enemyNames.Length; i++)
        {
            actionsDict.Add($"Kill {enemyNames[i]}",
Action.actionType.KillEnemyOrder);
            actionsDict.Add($"Shoot {enemyNames[i]}",
Action.actionType.KillEnemyOrder);
        }

        // ForceMissionFour();
        actionsDict.Add("Campfire first", Action.actionType.ForceMissionFour);
```

```
        actionsDict.Add("Clear campfire", Action.actionType.ForceMissionFour);
        for (int i = 0; i < enemyNames.Length; i++)
        {
            string pluralName = enemyNames[i] == "enemy" ? "enemies" : enemyNames[i] +
"s";
            actionsDict.Add($"Deal with {pluralName} near campfire",
Action.actionType.ForceMissionFour);
            actionsDict.Add($"Deal with {enemyNames[i]} near campfire",
Action.actionType.ForceMissionFour);
            actionsDict.Add($"Kill {pluralName} near campfire",
Action.actionType.ForceMissionFour);
            actionsDict.Add($"Kill {enemyNames[i]} near campfire",
Action.actionType.ForceMissionFour);
        }
    }


    public static int MatchingKeywordsPercentage(string source, string target)
    {
        if (string.IsNullOrEmpty(source) || string.IsNullOrEmpty(target))
            return 0;

        string[] sourceWords = source.Split();
        string[] targetWords = target.Split();

        int sLen = sourceWords.Length;

        int matchingKeywords = 0;

        for(int i = 0; i < sourceWords.Length; i++)
        {
            for(int x = 0; x < targetWords.Length; x++)
            {
                if (sourceWords[i].ToLower() == targetWords[x].ToLower())
                {
                    matchingKeywords++;
                    break;
                }
            }
        }

        return (int)(matchingKeywords * 100 / sLen);
    }

    public static int LevenshteinDistance(string source, string target)
    {
        source = source.ToLower();
        target = target.ToLower();

        if(string.IsNullOrEmpty(source))
        {
            if (string.IsNullOrEmpty(target))
                return 0;
            else
                return target.Length;
        }

        if (string.IsNullOrEmpty(target))
            return source.Length;

        if(source.Length > target.Length)
        {
            string temp = target;
```

```
            target = source;
            source = temp;
        }

        var m = target.Length;
        var n = source.Length;
        var distance = new int[2, m + 1];

        // Initializing the distance 'matrix'
        for(var j = 1; j <= m; j++)
        {
            distance[0, j] = j;
        }

        var currRow = 0;
        for(var i = 1; i <= n; ++i)
        {
            currRow = i & 1;
            distance[currRow, 0] = i;
            var prevRow = currRow ^ 1;
            for(var j = 1; j <= m; j++)
            {
                var cost = (target[j - 1] == source[i - 1] ? 0 : 1);
                distance[currRow, j] = Mathf.Min(Mathf.Min(
                    distance[prevRow, j] + 1,
                    distance[currRow, j - 1] + 1),
                    distance[prevRow, j - 1] + cost);
            }
        }

        return distance[currRow, m];
    }

public void CheckKeywordsAndFireEvents(string recognizedString)
    {
        Debug.Log("CheckKeywordsAndFireEvents(" + recognizedString + ")");
        //allyScript.TellAboutEnemies();
        string textToCheck = recognizedString.ToLower();
        textToCheck = textToCheck.Replace(",", "");
        textToCheck = textToCheck.Replace(".", "");
        textToCheck = textToCheck.Replace("?", "");
        textToCheck = textToCheck.Replace("!", "");
        textToCheck = textToCheck.Replace("Charlie", "");

        // Checking will be done word by word
        string[] expressions = textToCheck.Split(' ');

        bool stopNextAction = false;

        // Checking if the action is to be stopped
        foreach (string expression in expressions)
        {
            string compare = expression.Trim();
            if (stopNextAction == false)
            {
                if (compare.Contains("no") && compare.Length == 2)
                {
                    stopNextAction = true;
                }
                else if (compare.Contains("not") && compare.Length == 3)
                {
                    stopNextAction = true;
                }
```

```
                else if (compare.Contains("do not"))
                {
                    stopNextAction = true;
                }
                else if (compare.Contains("don't"))
                {
                    stopNextAction = true;
                }
            }
        }

        if (stopNextAction)
            return;

        int currMission = gameControl.GetCurrentMission();
        bool foundMatch = false;
        string matchText = "";
        string keyword = "";
        int matchPos = 0;

        if (textToCheck.Contains("roger") || textToCheck.Contains("understood") ||
textToCheck == "got it" || textToCheck == "thanks")
        {
            allyScript.rogerThat = true;
            allyScript.rogerThatText = textToCheck.Substring(0, 1).ToUpper() +
textToCheck.Substring(1);
            return;
        }

        matchList.Clear();

        // Check what type of speech is expected depending on current gameplay status
        if (waitForTargetSpecification)
        {
            bool foundKeyword = false;
            // Looking only for determining the target
            // If current mission is 2 or 4 - action keywords must be present
            if (currMission == 2 || currMission == 4)
            {
                for(int i = 0; i < actionKeywords.Length; i++)
                {
                    if(textToCheck.Contains(actionKeywords[i]))
                    {
                        foundKeyword = true;
                        keyword = actionKeywords[i];
                        break;
                    }
                }

                if(foundKeyword)
                {
                    foreach (string key in actionsDict.Keys)
                    {
                        if (actionsDict[key] ==
Action.actionType.DetermineEnemyToKill)
                        {
                            matchList.Add(MatchingKeywordsPercentage(key,
textToCheck));
                        }
                        else
                        {
                            matchList.Add(0);
```

```
                        }
                    }

                    matchPos = matchList.IndexOf(matchList.Max());
                    matchText = actionsDict.ElementAt(matchPos).Key;
                    // Determinig the % of match
                    if (matchList.Max() > 60)
                    {
                        foundMatch = true;
                    }
                }
                else
                {
                    // Keyword is not specified so looking for other possible text
actions
                    foreach (string key in actionsDict.Keys)
                    {
                        // Looking only tell what to do, tell about the enemies, tell
if enemy down
                        if (actionsDict[key] ==
Action.actionType.TellWhereToGoWhatToDo || actionsDict[key] ==
Action.actionType.TellAboutEnemies ||
                            actionsDict[key] == Action.actionType.TellIfEnemyDown)
                        {
                            matchList.Add(MatchingKeywordsPercentage(key,
textToCheck));
                        }
                        else
                        {
                            matchList.Add(0);
                        }

                    }

                    matchPos = matchList.IndexOf(matchList.Max());
                    // Determinig the % of match
                    if (matchList.Max() > 60)
                    {
                        CallAllyAction(actionsDict.ElementAt(matchPos).Value);
                    }
                }
            }

            // If successfully found a possible match
            if(foundKeyword && foundMatch)
            {
                bool determineBasedOnPlayerDecision = false;
                string[] split = matchText.Split();

                foreach(string word in split)
                {
                    if(word == "I" || word == "mine")
                    {
                        determineBasedOnPlayerDecision = true;
                        break;
                    }
                }

                if(determineBasedOnPlayerDecision)
                {
                    // Find the opposite word
                    string temp = "";
```

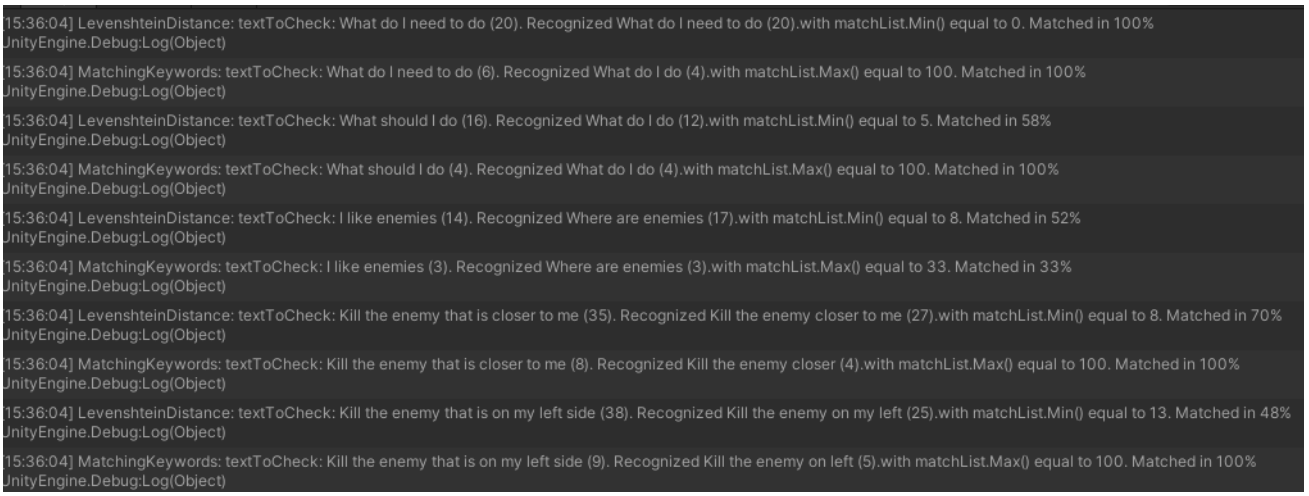
```
                    switch (keyword)
                    {
                        case "left":
                            temp = "right";
                            break;
                        case "right":
                            temp = "left";
                            break;
                        case "further":
                            temp = "closer";
                            break;
                        case "near":
                            temp = "closer";
                            break;
                        case "closer":
                            temp = "further";
                            break;
                        default:
                            temp = "left";
                            break;
                    }

                    allyScript.determineEnemyToKillPos = temp;
                    allyScript.determineEnemyToKill = true;
                }
                else
                {
                    allyScript.determineEnemyToKillPos = keyword;
                    allyScript.determineEnemyToKill = true;
                }
            }
            // Else if the keyword was used but no match was found.
            else if(foundKeyword)
            {
                allyScript.notUnderstood = true;
            }
        }
        else if(waitForKillConfirmation)
        {
            // Looking only for kill confirmation, enemy kill order, tell me what to
do, tell about enemies, tell if enemy down
            foreach (string key in actionsDict.Keys)
            {
                // Looking only tell what to do, tell about the enemies, tell if enemy
down
                if (actionsDict[key] == Action.actionType.ConfirmTarget ||
actionsDict[key] == Action.actionType.KillEnemyOrder ||
                    actionsDict[key] == Action.actionType.TellWhereToGoWhatToDo ||
actionsDict[key] == Action.actionType.TellAboutEnemies ||
                    actionsDict[key] == Action.actionType.TellIfEnemyDown)
                {
                    matchList.Add(MatchingKeywordsPercentage(key, textToCheck));
                }
                else
                {
                    matchList.Add(0);
                }
            }

            matchPos = matchList.IndexOf(matchList.Max());
            // Determinig the % of match
            if (matchList.Max() > 60)
```

```
            {
                CallAllyAction(actionsDict.ElementAt(matchPos).Value);
                // If executed the kill order
                if (actionsDict.ElementAt(matchPos).Value ==
Action.actionType.KillEnemyOrder)
                {
                    waitForKillConfirmation = false;
                    DisplayPossibleCommands(new string[] { "Where do I go", "Where is
the checkpoint", "What do I do", "Are targets down" });
                }
            }
        }
        else if(currMission == 3)
        {
            // Looking only for standard questions or if the player wants to deal with
campfire as well
            // Deal with enemies at campfire - call AllyScript.ForceMissionFour()
            foreach (string key in actionsDict.Keys)
            {
                // Looking only tell what to do, tell about the enemies, tell if enemy
down
                if (actionsDict[key] == Action.actionType.TellWhereToGoWhatToDo ||
actionsDict[key] == Action.actionType.TellAboutEnemies ||
                    actionsDict[key] == Action.actionType.ForceMissionFour)
                {
                    matchList.Add(MatchingKeywordsPercentage(key, textToCheck));
                }
                else
                {
                    matchList.Add(0);
                }
            }

            matchPos = matchList.IndexOf(matchList.Max());
            // Determinig the % of match
            if (matchList.Max() > 60)
            {
                CallAllyAction(actionsDict.ElementAt(matchPos).Value);
            }
        }
        else
        {
            // Looking only for standard questions
            foreach (string key in actionsDict.Keys)
            {
                // Looking only tell what to do, tell about the enemies, tell if enemy
down
                if (actionsDict[key] == Action.actionType.TellWhereToGoWhatToDo ||
actionsDict[key] == Action.actionType.TellAboutEnemies ||
                    actionsDict[key] == Action.actionType.TellIfEnemyDown)
                {
                    matchList.Add(MatchingKeywordsPercentage(key, textToCheck));
                }
                else
                {
                    matchList.Add(0);
                }
            }

            matchPos = matchList.IndexOf(matchList.Max());

            // Determinig the % of match
```

```
            if (matchList.Max() > 60)
            {
                Debug.Log("HERE 5");
                CallAllyAction(actionsDict.ElementAt(matchPos).Value);
            }
        }

        Debug.Log($"Matchpos: {matchPos}, matchlist.Max {matchList.Max()}, len
{matchList.Count}");

        if(matchList.Max() <= 60)
        {
            allyScript.notUnderstood = true;
        }
    }
    private void CallAllyAction(Action.actionType typeOfAction)
    {
        Debug.Log($"CallAllyAction({typeOfAction})");

        switch(typeOfAction)
        {
            case Action.actionType.ConfirmTarget:
                //allyScript.ConfirmTarget();
                allyScript.confirmTarget = true;
                break;
            /*case Action.actionType.DetermineEnemyToKill:
                allyScript.DetermineEnemyToKill(null);
                break;*/
            case Action.actionType.ForceMissionFour:
                //allyScript.ForceMissionFour();
                allyScript.forceMissionFour = true;
                break;
            case Action.actionType.KillEnemyOrder:
                //allyScript.KillEnemyOrder();
                allyScript.killEnemyOrder = true;
                break;
            case Action.actionType.TellAboutEnemies:
                //allyScript.TellAboutEnemies();
                allyScript.tellAboutEnemies = true;
                break;
            case Action.actionType.TellIfEnemyDown:
                //allyScript.TellIfEnemyDown();
                allyScript.tellIfEnemyDown = true;
                break;
            case Action.actionType.TellWhereToGoWhatToDo:
                //allyScript.TellWhereToGoWhatToDo();
                allyScript.tellWhereToGoWhatToDo = true;
                break;
            default:
                break;
        }
    }

    public void DisplayPossibleCommands(string[] commandsArray)
    {
        if (gameControl.speechRecognition)
            commandsText.text = "Try these commands [T]:\nRoger / Understood / Got
it\n";
        else
            commandsText.text = "Commands:\n";

        for (int i = 0; i < commandsArray.Length; i++)
```

```
        {            commandsText.text += commandsArray[i] + "\n";
        }}}
```

## Appendix 9 – Result comparison between Levenshtein distance and Keyword matching




## Appendix 10 – AllyScript.cs

https://github.com/dejwkubikson/Speech-Recognition-in-FPS-Games/blob/main/Honours%20Project/Assets/Scripts/AllyScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class AllyScript : MonoBehaviour
{
    GameObject playerObject;
    public GameObject enemyToKill = null;
    public Text allyText;
    public Text playerText;

    VoiceControl voiceControl;
    GameControl gameControl;

    public GameObject[] missionOneEnemies;
    public GameObject[] missionTwoEnemies;
    public GameObject[] enemies;

    private AudioSource audioSource;

    private bool answered = false;
    private float answerTimer = 3.0f;
    private float answerCurrTimer = 0.0f;

    public bool notUnderstood = false;
    public bool rogerThat = false;
    public string rogerThatText = "";
    public bool tellAboutEnemies = false;
    public bool tellIfEnemyDown = false;
    public bool tellWhereToGoWhatToDo = false;
    public bool determineEnemyToKill = false;
    public string determineEnemyToKillPos = "";
    public bool killEnemyOrder = false;
    public bool confirmTarget = false;
    public bool forceMissionFour = false;
```

```
    private string targetPos = "";

    // Start is called before the first frame update
    void Start()
    {
        playerObject = GameObject.FindGameObjectWithTag("Player");
        voiceControl =
GameObject.FindGameObjectWithTag("VoiceControl").GetComponent<VoiceControl>();
        gameControl =
GameObject.FindGameObjectWithTag("GameController").GetComponent<GameControl>();
        audioSource = GetComponent<AudioSource>();

        enemies = GameObject.FindGameObjectsWithTag("Soldier");
    }

    void Update()
    {
        enemies = GameObject.FindGameObjectsWithTag("Soldier");

        if (answered)
        {
            answerCurrTimer += Time.deltaTime;
            if(answerCurrTimer > answerTimer)
            {
                answered = false;
                answerCurrTimer = 0.0f;
                allyText.text = "";
                playerText.text = "";
            }
        }

        if(rogerThat)
        {
            Roger();
            rogerThat = false;
            rogerThatText = "";
        }

        if(notUnderstood)
        {
            NotUnderstood();
            notUnderstood = false;
        }

        if(tellAboutEnemies)
        {
            tellAboutEnemies = false;
            TellAboutEnemies();
        }

        if(tellIfEnemyDown)
        {
            tellIfEnemyDown = false;
            TellIfEnemyDown();
        }

        if(tellWhereToGoWhatToDo)
        {
            tellWhereToGoWhatToDo = false;
            TellWhereToGoWhatToDo();
        }
```

```
        if(determineEnemyToKill)
        {
            if (DetermineEnemyToKill(determineEnemyToKillPos))
            {
                voiceControl.waitForTargetSpecification = false;
                voiceControl.waitForKillConfirmation = true;
            }
            determineEnemyToKill = false;
            determineEnemyToKillPos = "";
        }

        if(killEnemyOrder)
        {
            killEnemyOrder = false;
            KillEnemyOrder();
        }

        if(confirmTarget)
        {
            confirmTarget = false;
            ConfirmTarget();
        }

        if(forceMissionFour)
        {
            forceMissionFour = false;
            ForceMissionFour();
        }

        if(!gameControl.speechRecognition)
        {
            if (Input.GetKeyDown(KeyCode.Alpha1))
            {
                Debug.Log("TellAboutEnemies()");
                TellAboutEnemies();
            }
            if (Input.GetKeyDown(KeyCode.Alpha2))
            {
                Debug.Log("TellWhereToGoWhatToDo()");
                TellWhereToGoWhatToDo();
            }
            if (Input.GetKeyDown(KeyCode.Alpha3))
            {
                Debug.Log("TellIfEnemyDown()");
                TellIfEnemyDown();
            }
            if (Input.GetKeyDown(KeyCode.Alpha4))
            {
                Debug.Log("Roger");
                Roger();
            }
        }
    }

    public void NotUnderstood()
    {
        int random = Random.Range(1, 4);
        if (random == 1)
            RadioCall("I didn't get that Alpha.");
        else if (random == 2)
            RadioCall("I don't understand you Alpha.");
```

```
        else if (random == 3)
            RadioCall("Negative, be clear with your commands Alpha.");
        else
            RadioCall("Alpha, what are you talking about?");
    }

    public void RadioCall(string text)
    {
        Debug.Log(text);
        answered = false;
        allyText.text = "Charlie: " + text;
    }

    public void Roger()
    {
        if (allyText.text == "" || answered)
            return;

        if(gameControl.speechRecognition)
        {
            //allyText.text = "Alpha: " + rogerThatText;
        }
        else
        {
            int random = Random.Range(1, 3);
            if (random == 1)
                playerText.text = "Alpha: Roger";
            else if (random == 2)
                playerText.text = "Alpha: Understood";
            else
                playerText.text = "Alpha: Got it";
        }

        answered = true;
    }

    public void TellWhereToGoWhatToDo()
    {
        int currentMission = gameControl.GetCurrentMission();

        switch (currentMission)
        {
            case 0:

                RadioCall("Alpha you need to get to the small village. " +
                    "There's a bridge that will get you through the river.");
                break;
            case 1:

                RadioCall("Alpha we need to wait for the group of enemies to pass.");
                break;
            case 2:

                if (missionOneEnemies[0].GetComponent<EnemyBehaviour>().dead &&
missionOneEnemies[1].GetComponent<EnemyBehaviour>().dead)
                {
                    if (gameControl.checkPointReached == false)
                    {
                        RadioCall("Alpha you need to get over the bridge and get to
the main village where Mohamed is.");
                    }
                    else
```

```
                    {
                        RadioCall("Alpha you need to get to the village.");
                    }
                }
                else
                {
                    if(enemyToKill == null)
                    {
                        RadioCall("Alpha we need to kill the two tangos near the
campfire. Tell me which one is yours.");
                        if (gameControl.speechRecognition)
                            voiceControl.DisplayPossibleCommands(new string[] { "I'll
take down the one on the left", "Right one is yours", "Right one is mine" });
                    }
                    else
                    {
                        RadioCall("Alpha waiting for your command to shoot.");
                        if (gameControl.speechRecognition)
                            voiceControl.DisplayPossibleCommands(new string[] {
"Shoot", "Kill", "Confirm your target" });
                    }
                }
                break;
            case 3:

                RadioCall("Alpha you need to eliminate or get past the multiple tangos
to the main building.");
                break;
            case 4:

                if (missionTwoEnemies[0].GetComponent<EnemyBehaviour>().dead &&
missionTwoEnemies[1].GetComponent<EnemyBehaviour>().dead)
                {
                    RadioCall("Alpha get into the main building and eliminate
Mohamed.");
                }
                else
                {
                    if (enemyToKill == null)
                    {
                        RadioCall("Alpha we need to kill the two tangos near the
campfire. Tell me which one is yours.");
                        if (gameControl.speechRecognition)
                            voiceControl.DisplayPossibleCommands(new string[] { "I
will take down the one closer to me", "Kill the one further to me", "Closer enemy to
me is yours" });
                    }
                    else
                    {
                        RadioCall("Alpha, waiting for your command to shoot.");
                        if (gameControl.speechRecognition)
                            voiceControl.DisplayPossibleCommands(new string[] {
"Shoot", "Kill", "Confirm your target" });
                    }
                }
                break;
            default:
                break;
        }
    }

    public void TellIfEnemyDown()
```

```
    {
        int currentMission = gameControl.GetCurrentMission();
        if(currentMission == 2)
        {
            if (missionOneEnemies[0].GetComponent<EnemyBehaviour>().dead &&
missionOneEnemies[1].GetComponent<EnemyBehaviour>().dead)
            {
                RadioCall("Alpha both targets down. Nice shot.");
            }
            else
            {
                if (gameControl.speechRecognition)
                {
                    if (voiceControl.waitForKillConfirmation)
                    {
                        RadioCall("Alpha tangos alive, waiting for your call");
                        voiceControl.DisplayPossibleCommands(new string[] { "Shoot",
"Kill", "Confirm your target" });
                    }
                    else
                    {
                        RadioCall("Alpha tangos alive, waiting for your target
confirmation.");
                        voiceControl.DisplayPossibleCommands(new string[] { "I'll take
down the one on the left", "Right one is yours", "Right one is mine" });
                    }
                }
            }
        }
        else if(currentMission == 4)
        {
            if (missionTwoEnemies[0].GetComponent<EnemyBehaviour>().dead &&
missionTwoEnemies[1].GetComponent<EnemyBehaviour>().dead)
            {
                RadioCall("Both tangos down. Good shot Alpha.");
            }
            else
            {
                if (gameControl.speechRecognition)
                {
                    if (voiceControl.waitForKillConfirmation)
                    {
                        RadioCall("Alpha, tangos alive, waiting for your call");
                        voiceControl.DisplayPossibleCommands(new string[] { "Shoot",
"Kill", "Confirm your target" });
                    }
                    else
                    {
                        RadioCall("Alpha, tangos alive, waiting for your target
confirmation.");
                        voiceControl.DisplayPossibleCommands(new string[] { "I'll take
down the one near me", "Kill the one near the bulding", "My one is near the building"
});
                    }
                }
            }
        }
        else
        {
            RadioCall("Alpha, nothing to confirm.");
        }
    }
```

```
    public void TellAboutEnemies()
    {
        int[] nsewEnemyCount = { 0, 0, 0, 0 };
        int totalEnemies = 0;

        for (int i = 0; i < enemies.Length; i++)
        {
            GameObject enemy = enemies[i];

            if (enemy.GetComponent<EnemyBehaviour>() != null)
            {
                if (enemy.GetComponent<EnemyBehaviour>().dead)
                    continue;
            }

            playerObject = GameObject.FindGameObjectWithTag("Player");

            float distance = Vector3.Distance(enemy.transform.position,
playerObject.transform.position);

            // Find enemies in radious
            if (distance <= 4)
            {
                // Forward vector will always be the same. Prevents from giving
'directional' direction of enemies.
                Vector3 fwd = new Vector3(0, 0, 1); //playerObject.transform.forward;
                Vector3 targetDir = enemy.transform.position -
playerObject.transform.position;

                // Getting angle to object
                float angleToEnemy = Vector3.Angle(targetDir, fwd);

                // Checking if the object is on the right or left of the object - used
to get the correct Compass location
                Vector3 perp = Vector3.Cross(fwd, targetDir);
                float dir = Vector3.Dot(perp, playerObject.transform.up);

                // N
                if(angleToEnemy <= 45)
                {
                    //Debug.Log($"{enemy.name}: Enemy at North, angle {angleToEnemy},
dir {dir}");
                    nsewEnemyCount[0] += 1;
                }
                // E
                else if(angleToEnemy > 45 && angleToEnemy <= 135 && dir > 0)
                {
                    //Debug.Log($"{enemy.name}: Enemy at East, angle {angleToEnemy},
dir {dir}");
                    nsewEnemyCount[2] += 1;
                }
                // W
                else if(angleToEnemy > 45 && angleToEnemy <= 135 && dir < 0)
                {
                    //Debug.Log($"{enemy.name}: Enemy to West, angle {angleToEnemy},
dir {dir}");
                    nsewEnemyCount[3] += 1;
                }
                // S
                else
                {
```

```
                    //Debug.Log($"{enemy.name}: Enemy at South, angle {angleToEnemy},
dir {dir}");
                    nsewEnemyCount[1] += 1;
                }

                totalEnemies++;
            }
        }

        // List enemies to the player
        string radioText = "";
        if (nsewEnemyCount[0] > 0)
        {
            if (nsewEnemyCount[0] > 1)
            {
                radioText += $"{nsewEnemyCount[0]} enemies at your North.";
            }
            else
            {
                radioText += "One enemy at your North. ";
            }
        }

        if (nsewEnemyCount[2] > 0)
        {
            if (nsewEnemyCount[2] > 1)
            {
                radioText += $"{nsewEnemyCount[2]} enemies at your East. ";
            }
            else
            {
                radioText += "One enemy at your East. ";
            }
        }

        if (nsewEnemyCount[3] > 0)
        {
            if (nsewEnemyCount[3] > 1)
            {
                radioText += $"{nsewEnemyCount[3]} enemies at your West. ";
            }
            else
            {
                radioText += "One enemy at your West. ";
            }
        }

        if (nsewEnemyCount[1] > 0)
        {
            if (nsewEnemyCount[1] > 1)
            {
                radioText += $"{nsewEnemyCount[1]} enemies at your South. ";
            }
            else
            {
                radioText += "One enemy at your South. ";
            }
        }

        if(totalEnemies == 0)
        {
            radioText = "I don't see anyone nearby.";
```

```
        }

        RadioCall("Alpha, " + radioText);
    }

    private static Vector3 getRelativePosition(Transform origin, Vector3 position)
    {
        Vector3 distance = position - origin.position;
        Vector3 relativePosition = Vector3.zero;

        relativePosition.x = Vector3.Dot(distance, origin.right.normalized);
        relativePosition.y = Vector3.Dot(distance, origin.up.normalized);
        relativePosition.z = Vector3.Dot(distance, origin.forward.normalized);

        return relativePosition;
    }

    public bool DetermineEnemyToKill(string enemyPosName)
    {
        Debug.Log($"DetermineEnemyToKill({enemyPosName})");
        bool found = false;
        int currentMission = gameControl.GetCurrentMission();

        Vector3 enemy1Pos = Vector3.zero;
        Vector3 enemy2Pos = Vector3.zero;
        GameObject enemy1 = null;
        GameObject enemy2 = null;

        // Getting relative position from player to enemy
        if (currentMission == 2)
        {
            enemy1 = missionOneEnemies[0];
            enemy2 = missionOneEnemies[1];
            Debug.Log($"Relative 1: {enemy1Pos}, 2: {enemy2Pos}");
        }
        else if(currentMission == 4)
        {
            enemy1 = missionTwoEnemies[0];
            enemy2 = missionTwoEnemies[1];
        }

        switch(enemyPosName)
        {
            case "left":
                enemy1Pos = getRelativePosition(playerObject.transform,
enemy1.transform.position);
                enemy2Pos = getRelativePosition(playerObject.transform,
enemy2.transform.position);
                if (enemy1Pos.x <= enemy2Pos.x)
                    EnemyToKill(enemy1);
                else
                    EnemyToKill(enemy2);
                found = true;
                break;
            case "right":
                enemy1Pos = getRelativePosition(playerObject.transform,
enemy1.transform.position);
                enemy2Pos = getRelativePosition(playerObject.transform,
enemy2.transform.position);
                if (enemy1Pos.x >= enemy2Pos.x)
                    EnemyToKill(enemy1);
                else
```

```
                    EnemyToKill(enemy2);
                found = true;
                break;
            case "closer":
                enemy1Pos = enemy1.transform.position;
                enemy2Pos = enemy2.transform.position;
                if (Vector3.Distance(playerObject.transform.position, enemy1Pos) <=
Vector3.Distance(playerObject.transform.position, enemy2Pos))
                    EnemyToKill(enemy1);
                else
                    EnemyToKill(enemy2);
                found = true;
                break;
            case "near":
                enemy1Pos = enemy1.transform.position;
                enemy2Pos = enemy2.transform.position;
                if (Vector3.Distance(playerObject.transform.position, enemy1Pos) <=
Vector3.Distance(playerObject.transform.position, enemy2Pos))
                    EnemyToKill(enemy1);
                else
                    EnemyToKill(enemy2);
                found = true;
                break;
            case "further":
                enemy1Pos = enemy1.transform.position;
                enemy2Pos = enemy2.transform.position;
                if (Vector3.Distance(playerObject.transform.position, enemy1Pos) >=
Vector3.Distance(playerObject.transform.position, enemy2Pos))
                    EnemyToKill(enemy1);
                else
                    EnemyToKill(enemy2);
                found = true;
                break;
            default:
                found = false;
                break;
        }

        if(found == false)
        {
            int random = Random.Range(1, 3);
            if(random == 1)
                RadioCall("Alpha, I didn't understand that, I repeat, I didn't
understand that.");
            else if(random == 2)
                RadioCall("Alpha, I don't know which tango you want me to kill, I
repeat, I don't know which tango you want me to kill.");
            else
                RadioCall("Alpha, be more specific with the target you want me to
shoot, I repeat, be more specific with the target you want me to shoot.");
        }
        else
        {
            targetPos = enemyPosName;

            if (enemyPosName == "further" || enemyPosName == "closer")
                RadioCall("Roger that Alpha, I'll kill the target " + enemyPosName + "
to you.");
            else if (enemyPosName == "near")
                RadioCall("Roger that Alpha, I'll kill the target near you");
            else
```

```
                RadioCall("Roger that Alpha, I'll kill the target on your " +
enemyPosName + ".");
        }

        return found;
    }

    public void ConfirmTarget()
    {
        if(enemyToKill == null)
        {
            RadioCall("Negative, specify the target you want me to kill.");
        }

        if (targetPos == "further" || targetPos == "closer")
            RadioCall("Alpha, I'm aimed in the target " + targetPos + " to you.");
        else if (targetPos == "near")
            RadioCall("Alpha, I'm aimed in the target near you.");
        else
            RadioCall("Alpha, I'm aimed in the target on your " + targetPos + ".");
    }

    public void EnemyToKill(GameObject enemyObject)
    {
        enemyToKill = enemyObject;

        RadioCall("Alpha, target in sight. On your command.");

        if (gameControl.speechRecognition)
        {
            voiceControl.DisplayPossibleCommands(new string[] { "Shoot", "Kill",
"Confirm your target" });
            voiceControl.waitForKillConfirmation = true;
        }
    }

    public void ForceMissionFour()
    {
        gameControl.forceMission4 = true;
    }
    public void KillEnemyOrder()
    {
        if (enemyToKill == null)
            return;

        EnemyBehaviour enemyScript = enemyToKill.GetComponent<EnemyBehaviour>();

        enemyScript.ReceiveDamage(100);

        audioSource.Play();

        int random = Random.Range(1, 4);
        if (random == 1)
            RadioCall("Alpha, enemy down");
        else if (random == 2)
            RadioCall("Alpha, tango down");
        else if (random == 3)
            RadioCall("Alpha, target dead");
        else
            RadioCall("Alpha, enemy killed");
    }
}
```