# Creating an Audio Game Platform for the Visually Impaired

An Interactive Qualifying Project Report
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
May 8, 2015

**Written by**

Lucas R. Lebrao

Robert E. McKenna

Justin J. Morrow

**Advised by**

Professor Keith Zizza

## ABSTRACT

Interactive media is a form of entertainment that has become increasingly popular in recent years. However, the entertainment medium is not accessible to people with visual impairments. This project investigates the rise of interactive media with accessibility in mind, focusing on audio-based games. To support this research, this project will involve the development of a game platform using audio and motion-based inputs. This prototype is a proof-of-concept, showing the possibilities of creating games for those who cannot normally play them.

# TABLE OF CONTENTS

**ABSTRACT** .......... **2**

**TABLE OF CONTENTS** .......... **3**

**TABLE OF FIGURES** .......... **4**

**1 INTRODUCTION** .......... **5**

**2 BACKGROUND AND RESEARCH** .......... **6**

2.1 ACCESSIBILITY IN GAMES .......... 6

2.2 EXISTING AUDIO GAMES .......... 7

2.3 VOICE INPUT .......... 10

2.4 ACCELEROMETER INPUT .......... 11

**3 PROJECT** .......... **13**

3.1 PROTOTYPING .......... 13

*3.1.1 Engine Audio Tools* .......... *13*

*3.1.2 Voice Recognition* .......... *14*

*3.1.3 Accelerometer* .......... *16*

3.2 GAME DEVELOPMENT .......... 19

*3.2.1 Early Game Ideas* .......... *20*

*3.2.2 Marco Polo Mini-game* .......... *20*

*3.2.3 Sound Environment* .......... *21*

**4 RESULTS** .......... **23**

4.1 INPUT .......... 23

*4.1.1 Voice Recognition* .......... *23*

*4.1.2 Motion Control* .......... *23*

4.2 RECEPTION .......... 24

*4.2.1 Voice Recognition* .......... *24*

*4.2.2 Motion Control* .......... *24*

*4.2.3 Game Design* .......... *25*

**5 CONCLUSION** .......... **26**

**6 REFERENCES** .......... **28**

## Table of Figures

Figure 1: Unity engine externsion for audio. .....13
Figure 2 : voice receive server .....15
Figure 3: Voice receive object .....15
Figure 4: accelerometer connected to the Arduino .....17
Figure 5: Arduino accelerometer input .....18
Figure 6: Arduino serial input in Unity .....19
Figure 7: Unity editor view of audio system .....22

# 1 INTRODUCTION

The gaming industry has been steadily growing for many years, and gaming has become one of the primary forms of entertainment for young people. In the United States alone, over 150 million people regularly play video games.[1] With this growth, there has also been an emergence in games made with accessibility in mind for those who cannot play standard video games due to some sort of impairment. Visual impairments affect a significant portion of the population, including 14 million people in the U.S.[2] This project focuses on the usage of vocal and motion-based input in games as alternatives to the standard physical inputs, as well as the use of audio output as the primary form of feedback to the player, in order to make games more accessible for the visually impaired.

Voice recognition and motion control are not new conventions: both have been used in games before, and have become increasingly popular in recent years. The existing games that use these technologies serve as a basis for research for this project. By learning from the successes and failures of prior works, a better, more accessible system can be built. This paper discusses the process of designing a prototype gaming platform for those faced with visual impairments, from the initial research to building a prototype with a simple Unity project implementing it as proof of concept.

[1] Campbell, Colin. "Here's How Many People Are Playing Games in America." *Polygon.*
[2] "National Data." *Vision Health Initiative (VHI).*

# 2 Background and Research

This project approaches both accessibility in games and audio-centric game design. Research is required in both of these areas, with the goal of discovering whether anything similar to this project has been done before, what features need to be focused on to create a successful platform for an accessible game, and what needs to be avoided in this process. These points all need to be considered for both the design of a game, and the design of the game platform itself.

Other points that need to be researched include the actual forms of input: accelerometer-based motion tracking and voice-recognition. These two features are the core of the hardware design for this project, and research is required to understand what work has already been done in these fields, and to make sure the idea being developed is unique and useful.

## 2.1 Accessibility in Games

Accessibility is defined as how easy-to-use a tool or object is to people with various disabilities. In video games, there has been a growing number of developers taking accessibility in mind during the design process, as well as more tools and mechanics provided to players so anyone is capable of playing games, despite their impairments. This research highlights the importance of accessibility in games, as well as what has already been done in "accessible games" using the technology available, including voice-recognition and motion-tracking. This research will also provide a good idea of what types of things are necessary, and what should be avoided, in an accessible game.

In an in-depth analysis of the 2002 U.S. census, researchers have found that potentially 11% of the American population is affected by some disability that affects their capacity to play video games. Of that population, 2% of people are unable to play games at all[3]. 11% of the U.S. population is a very large demographic of potential users who would benefit from some sort of accessibility tools. In recent years, awareness of the need for accessibility in games has grown. This has led to the creation of more accessible content and guidelines for game developers to begin creating even more accessible games. Many developers and other people in the gaming industry have become very vocal on the matter, providing both knowledge and tools to allow more people to play their games.

Many games have not been designed specifically with accessibility in mind. Interestingly, some of these games have been picked up by people with impairments, despite the difficulties they face. One

[3] "Yuan, Bei, Eelke Folmer, and Frederick C. Harris, Jr. "Game Accessibility: A Survey." *Springer-Verlag*

great example of this is *Mortal Kombat* (2011), in which a blind player named Carlos Vasquez reached the finals in his pool during a major tournament, Evolution 2014, relying entirely on sound.[4] His performance at this tournament inspired the developers of *Injustice: Gods Among Us*, another fighting game, to add an accessibility mode. Despite *Mortal Kombat* not actually including any accessibility tools of its own, the game design made gameplay possible by sound alone.

Two well-known charities promoting accessibility in game design are *AbleGamers* and *SpecialEffects*, both of which seek fundraising to support development of games for people with disabilities. These charities promote major events, awareness of disabilities, and methods for game designers to provide more support for accessibility. There are also two major online resources for how to make games more accessible: the *Game Accessibility Guidelines*[5] and *Includification*[6].These guidelines provides ways for developers to solve issues within their games, as well as making support for outside tools such as screen readers or key remappers, which allow more people to play their games. All approaches are listed in level of difficulty to implement, where some cheap and quick solutions can be used to make games more accessible with very little cost.

## 2.2 Existing Audio Games

There are a few audio games on the market currently. Most of these games are made by independent developers, and the quality varies greatly between different games. One of the most well-known examples of a good audio game is Papa Sangre, an iOS exclusive game controlled entirely by a touch screen. Some other examples of popular audio games are *BlindSide*, *The Nightjar*, and *Entombed*, all of which have been play-tested and reviewed for this project. *BlindSide* and *The Nightjar* are both survival-horror games, while *Entombed* is a dungeon-crawler. The popularity of horror-based audio games is probably due to the nature of audio games in general: horror games rely on their atmosphere to immerse the player in a world where they truly feel fear, and the gameplay, while important, usually becomes secondary. Therefore, the horror genre lends itself to audio games, in which the focus is usually on the immersive environment built by sound, with gameplay often being less important. Unfortunately,

[4] Tullis, Ryan. "Blind Player Competes at Evolution, Says, 'If You're Playing Me... Don't Hold Back'" *EventHubs*.
[5] "Game Accessibility Guidelines | A Straightforward Reference for Inclusive Game Design." *Game Accessibility Guidelines RSS2*.
[6] "Welcome to Includification - Actionable Game Accessibility." *Includification*.

there are few well-known audio games of different genres; *Entombed* is the only example of a non-horror game that was tested for this project.

*BlindSide* is a survival-horror type game in which your character is tasked with guiding his girlfriend to safety in an apocalyptic setting where everybody has suddenly gone blind. On iOS devices, forward movement is controlled by pressing the top of the screen, backwards movement is controlled by the bottom of the screen, and turning is handled through rotating or tilting the device. Many reviews praise the sound quality of this game, and the gameplay itself is easy to learn. One issue discovered during testing is that it is very difficult to get a mental picture of the area that the player is in. There are only a few ambient sounds in the environment, so most of the world is discovered by walking into objects and having the player character tell you what he walked into. The biggest flaw with this game, however, is the voice acting. In a 30-second scene of walking through a hallway, the same voice line from one character was played at least six times. This can be seen even in the trailer, where the same voice lines are heard over and over again. The quality of the voice lines themselves are not very good, as the voice actors do not seem to fit in a horror setting. For example, the player character is always cheerful sounding and excited, despite the fact that he heard a person being eaten by a monster no more than 15 feet away from him. The unfitting voices, coupled with the incredible repetition of voices, make this game almost unplayable for more than a few minutes.

*The Nightjar* is another survival-horror game built on the same engine as Papa Sangre. This game, however, is set in a sci-fi setting, in which the player is a survivor on a stranded space ship. The controls for this game consist of two buttons on the bottom left and right of the screen for the left and right foot, and a dial at the top of the screen that is used to turn. The play tester complained that this is incredibly difficult to get used to if the screen is not visible (if the player is actually blind or covers their eyes), making the game difficult to play. The voice acting and sound quality in this game are very high, with narration from Benedict Cumberbatch, a professional actor. The ambience and other sound effects, such as the footsteps from the player’s mechanical suit, are all high quality and help immerse the player in the world. However, the gameplay itself is troubling to get used to. In one situation, the player had to navigate towards a large beeping object, but the player could never be sure of its exact location due to limitations in the audio from the game engine. As this game is on iOS devices, it has stereo sound but not surround, so there is no way to tell if a sound is coming from in front of or behind the player, making navigation difficult. Overall, this game is very high quality and tries its best to immerse the player, with only a few major flaws.

*Entombed* is the first and only game tested that was not a survival-horror, instead being an RPG dungeon-crawler. In this type of game, the player chooses a class (fighter, mage, etc.) and a class and begins adventuring through some sort of maze-like dungeon, fighting standard fantasy monsters such as goblins and kobolds. The game begins with a fully-narrated introduction, with high-quality voice and ambience. The game is controlled entirely with a keyboard, which seems like it may be difficult for people who cannot actually see their keyboard. This is almost certainly not a problem for a visually impaired person with a specialized keyboard. Unfortunately, it was very difficult to make any progress in this game. The controls are not very intuitive, as different keys are used to do different things (Q to open the Quest menu, I for inventory, etc), and it does not ever tell you this in the game. As far as actual gameplay, the combat is generally smooth, if not very slow. The narrator reads every character's name and every action during a round of combat, which takes a very long time. Traversing the world, however, is incredibly difficult. Moving in a direction produces one of two sounds: a sound of walking forward, or hitting a wall. The way this system works is not very clear, and at some points the player seems to have gotten myself stuck in a corner and could only escape by mashing the arrow keys. The biggest issue with this game is the narrator, which is a standard text-to-speech program. For a game of this size, it is understandable that text-to-speech would be a viable option, but the lack of a true narrator makes the game very difficult to listen to without difficulty.

Playing through these audio-based games gave a good idea of what has been done correctly and what has been done wrong. One of the most important aspects that most games have gotten right is the environment. In all of the tested games, the ambient sound is high quality and makes the player feel like they are truly in the area. These games also mostly had easy-to-use control schemes, with only *Entombed* requiring the player to actually see the keyboard or controller they are using. This is especially important in a game that is supposed to be accessible to visually-impaired people, as they will not be able to see a controller or keyboard as they play. The biggest flaw in these games is the narration or other human voices. *The Nightjar* has no flaws with the character voices, and the game is clearly well-received due to this. The other games, however, become nearly unplayable entirely due to the character voices. *BlindSide*'s characters repeat themselves constantly, and the voices are not realistic at all. This ruins any sense of immersion the player may have in the world, ultimately making the entire experience unenjoyable. *Entombed* suffers due to its narrator being a text-to-speech program, which itself may not be such a bad thing, but when the narrator is constantly reading text throughout every combat scene, the repetition becomes too much to bear.

## 2.3 Voice Input

Voice recognition technology has been a work-in-progress since the 1950's. Many development projects and teams have attempted to take advantage of the intuitive power that such technology inherently has, with varying levels of success. This power stems from the fact that voice communication has been a development of the human species for thousands of years and therefore has implicitly defined rules that all people begin learning from birth, unlike any other input device. The complexity of these rules is also what makes the development of such an elegant input system extremely difficult for computing systems which must follow very well-defined rules.

One of the earliest speech recognition devices was developed by Bell Laboratories in 1952. This system was titled the “Audrey System.”[7] Not surprisingly, this system could only understand digits, as they are well defined in any context. However, for the first technology of this kind, even such a small step is a huge achievement. It was not until speech recognition began making use of probabilistic models that more dynamic and complicated speech patterns could be recognized. Today, most general-purpose speech recognition systems are based on the Hidden Markov Models (HMM). "The Hidden Markov Model is a powerful statistical tool for modeling generative sequences that can be characterized by an underlying process generating an observable sequence.”[8] The reason such a model is useful is because human language is made up of patterns which, without a probabilistic model, may not be easily deciphered.

Today, implementing a speech recognition service is fairly straight forward through the use of software API’s such as Microsoft’s .NET speech recognition library. Such a library provides many data structures, such as Grammars, that allow developers to define words of interest, accuracies, and strings of words that should be recognized. The use of a speech recognition software for an input module for this project was a clear requirement from the beginning. Such software opens up the product to a large audience of people with disabilities, including not only the visually impaired, but also those who do not have full motor functions in their hands. People with such disabilities would have no problem making use of this input method given the inherent knowledge that most would have of their core language. In addition, voice commands come naturally to anybody who can speak, ensuring that the platform is easy to use for most demographics.

Few games actually managed to implement voice recognition successfully, so there are very few examples to draw on. Most of the early games using this technology were found on consoles such as the

---

[7] Pinola, Melanie. "Speech Recognition Through the Decades: How We Ended Up With Siri." *TechHive*.
[8] Blunsom, Paul. “Hidden Markov Models.”

Nintendo 64; perhaps the most famous example of this is *Hey You, Pikachu!* for the N64. This game came included with a very simple microphone, and the in-game responses were usually okay. This example is particularly popular mostly due to being a Pokémon game, and for the very finicky reception from the microphone. Other examples of games using voice recognition include *Seaman* (Sega Dreamcast, 1999), *Lifeline* (Sony PlayStation 2, 2003) and *Odama* (Nintendo GameCube, 2006). These games all had varying levels of success, though the voice controls in all three of them were mediocre at best.

One of the few very successful examples of voice recognition usage is *Tom Clancy's Endwar* (2008), which implements an intelligent VR system that allows the player to give orders to units under their command. The system will attempt to auto-complete commands once you begin them, making it very easy to use, and the entire system is optional so it will not bother players who do not want to use it. Another recent example is *There Came an Echo* (2015), a game which takes a similar approach as *Endwar.* In this game, the VR system allows dialogue between the player and his squad, and it includes the capacity to create custom orders.[9] These examples prove that using voice recognition in games is feasible, but requires great care to be sure it works well.

## 2.4 Accelerometer Input

Accelerometer input has recently become a key feature in many games, especially those on mobile platforms. Almost every mobile device, including Smartphones and tablets, include an accelerometer for motion-based inputs, which means they can be implemented into games easily. Some games, such as *Bit.Trip Beat*, give players the option to use either motion controls or touch controls, while others, such as *Cube Runner* and *Temple Run*, require motion controls. Either way, motion controls are usually very intuitive; to move their character in a direction, the player tilts their device in that direction. On a mobile device, motion control is especially easy to use since the device is small enough to be moved accurately with little effort.

One of the most well-known non-mobile uses of accelerometer input is the Oculus Rift, a "virtual reality head mounted display." The Oculus Rift uses a combination of accelerometers, gyroscopes, and other devices to track the user's full head movements accurately.[10] Many games that can be played using the Rift were not intended to utilize motion control, unlike many mobile games. Instead, the motion-based inputs from the Rift replace movements that otherwise would be controlled by the mouse. Other

[9] "*There Came an Echo*." *Kickstarter*.
[10] "The All New Oculus Rift Development Kit 2 (DK2) Virtual Reality Headset." *Oculus VR*.

than the Oculus Rift, there are very few headsets with motion tracking, most of which have not been nearly as successful.

There are very few other notable examples of accelerometer input. Besides mobile games and virtual reality, the most successful example is the Nintendo Wii. The controller for the Wii (called a Remote) has an accelerometer for motion controls, alongside an optical sensor to determine where it is pointing. The Wii was hugely successful, selling over 100 million units[11] worldwide, becoming one of the five most successful video game consoles of all time.

Besides the few anomalies such as the Oculus Rift and the Nintendo Wii, few gaming devices have been successful with motion controls as a primary input. This is due to many issues, such as the lack of precision that can be achieved with a standard controller and the physical stress it may cause. Many people still view motion control as a gimmick used to simply be different or attract attention, though its recent rise in popularity suggests that it will continue to see use in video games in the future. As for its usage in gaming for people with disabilities, motion control is a very situational technology. The Nintendo Wii, for example, was actually used as a rehabilitation device for a patient with cerebral palsy.[12] However, other users complained about the motion controls being too physically demanding, which is especially true for users with physical disabilities. For those with visual impairments, motion control can be used to create a truly immersive experience. For example, one group of researchers created an “audio-only sports game” with an array of speakers and a motion-tracking handheld device.[13] This device can simulate a chosen sport almost perfectly to make the player feel like they are truly playing the game.

These examples of other game platforms show how motion controls can used to make the player feel immersed in the world they are playing in much more than any other input system, despite them not having any visual feedback. This is a key feature in this project, as the game platform is intended to be used by people with visual impairments; even without vision, they are able to feel like they are truly interacting with the world inside a game.

[11] "Consolidated Sales Transition by Region" (PDF). Nintendo.
[12] "Research Shows Rehabilitation Benefits of Using Nintendo Wii." *NewsWise*.
[13] Hermann, Thomas. "AcouMotion." *Sonification*

# 3 PROJECT

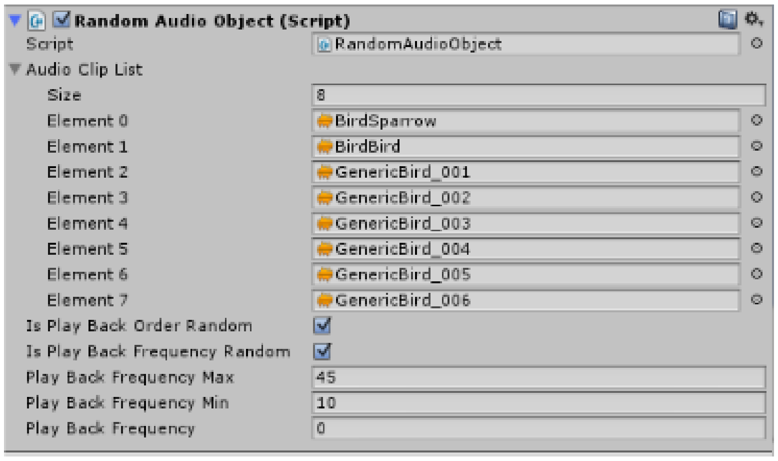
During the planning stage of this project, the prototype was divided into three core systems: the engine, the voice recognition system, and the motion control system. These systems were necessary to implement to ensure rapid prototyping of any possible game ideas. The three chosen systems form the core of the technical work, as they include all of the input and output for the overall system. Once these three modules were created, the project's focus could shift away from technical work and move towards designing a game that would demonstrate everything that has been learned through the previous research.

## 3.1 PROTOTYPING

In order to actually test the viability of the chosen designs, a prototype was needed. For this project, the Unity game engine was chosen as it is relatively easy to work with, and it includes some generic audio manipulation tools. For input, a voice recognition software was required, and for motion tracking, an accelerometer was needed. Together, these three modules formed the first prototype for the project.

### 3.1.1 Engine Audio Tools

Unity, the engine being utilized for this project, provides an extensive set of Audio based tools for designers to work with, although these tools are generic in regards to visual games that often don't place priority on audio tools over 3D manipulation tools. To create an environment ideal for developing audio only games a number of engine extensions were written.



**Random Audio Object (Script)**

| | |
|---|---|
| Script | RandomAudioObject |
| Audio Clip List | |
| Size | 8 |
| Element 0 | BirdSparrow |
| Element 1 | BirdBird |
| Element 2 | GenericBird_001 |
| Element 3 | GenericBird_002 |
| Element 4 | GenericBird_003 |
| Element 5 | GenericBird_004 |
| Element 6 | GenericBird_005 |
| Element 7 | GenericBird_006 |
| Is Play Back Order Random | ✓ |
| Is Play Back Frequency Random | ✓ |
| Play Back Frequency Max | 45 |
| Play Back Frequency Min | 10 |
| Play Back Frequency | 0 |

*Figure 1: An example of a custom engine extension coded for this project, abstracting basic audio features.*

The most important class extension was the AudioObject. The AudioObject class contains functionality for caching multiple audio clips, playing them on triggers, in sequence, randomly, or in loops. These were used in combination with Unity's built in audio manipulation tools that allow designers to change various aspects of audio clips and generating their 3D sound equivalents. The AudioObject class is the parent of all game objects built in this project.

The second core class extended from Unity's engine is the VoiceReceiveObject, an object that inherently has a reference to the system that receives all voice input in the engine. This class allows designers to easily implement voice control, with only the need to define behavior for any received audio. This class is necessary for any controlled objects that require voice control.

Implementing these core features, along with numerous other secondary tools allowed for rapid prototyping of game concepts, allowing for more time testing the most important aspect of the project: ease of use for any user.

### 3.1.2 Voice Recognition

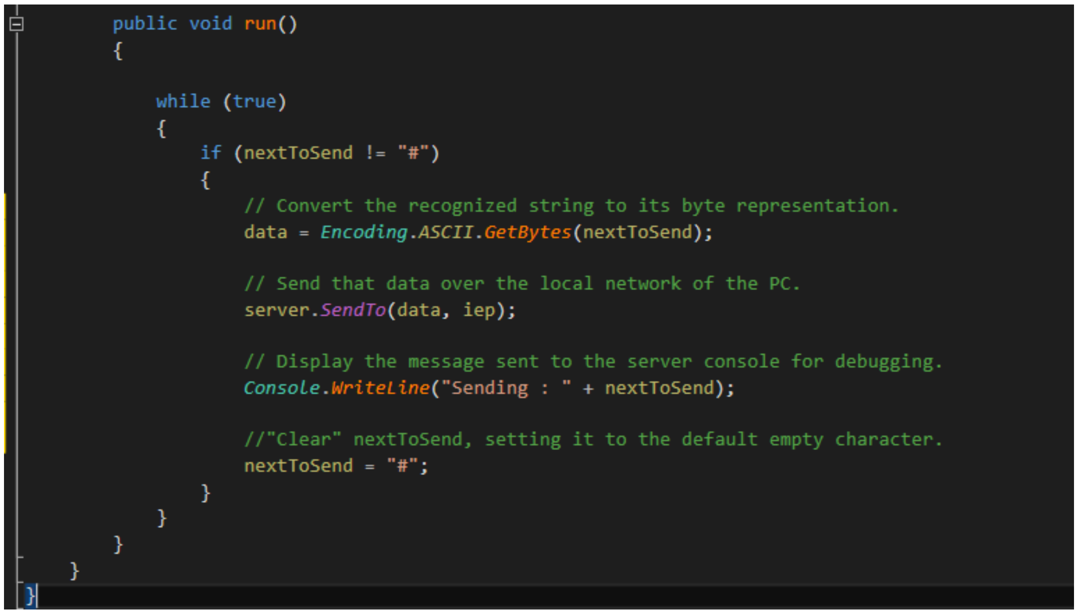
Voice recognition was implemented through the use of Microsoft's .NET Voice Recognition Framework. The choice to use this framework was not the first, or most obvious choice, but became a necessity due to a number of variables.

Firstly, due to no funding, this project needed to make use of free software; this may have been the most influential variable when picking what software to use. The lack of funding limited both the engine and voice recognition software choices. This resulted in first, needing to pick a voice recognition software that could be integrated easily into the engine which provided no such functionality, and second, requiring the voice recognition software itself to be capable of high accuracy recognition as well as ease of extensibility. The .NET Voice Recognition software met both of these needs, but did not provide an easy means of integration into the game engine being used. To solve this issue, a work around was required.

To integrate voice recognition into the game, a separate program modeled after a basic network server was devised. The server would listen for known speech patterns and then send a data packet to the engine, which would receive, parse, and utilize the data packet. This way, all of the .NET framework's power could be utilized, even though the free version of Unity does not actually support its integration.

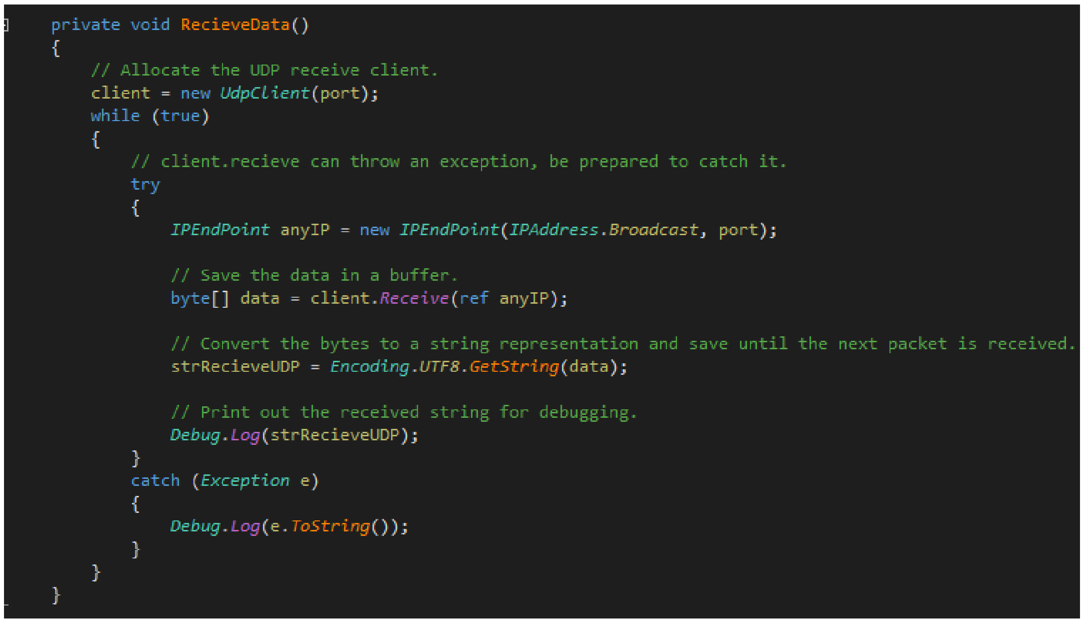
This system was influenced and extended from software other users of the Unity forums had developed[14].

Below are two code snippets that represent the core of the voice input system.



```
        public void run()
        {

            while (true)
            {
                if (nextToSend != "#")
                {
                    // Convert the recognized string to its byte representation.
                    data = Encoding.ASCII.GetBytes(nextToSend);

                    // Send that data over the local network of the PC.
                    server.SendTo(data, iep);

                    // Display the message sent to the server console for debugging.
                    Console.WriteLine("Sending : " + nextToSend);

                    //"Clear" nextToSend, setting it to the default empty character.
                    nextToSend = "#";
                }
            }
        }
    }
}
```

*Figure 2 : The voice receive server runs in a loop, listening for recognized input and sending it when input is received.*



```
    private void RecieveData()
    {
        // Allocate the UDP receive client.
        client = new UdpClient(port);
        while (true)
        {
            // client.recieve can throw an exception, be prepared to catch it.
            try
            {
                IPEndPoint anyIP = new IPEndPoint(IPAddress.Broadcast, port);

                // Save the data in a buffer.
                byte[] data = client.Receive(ref anyIP);

                // Convert the bytes to a string representation and save until the next packet is received.
                strRecieveUDP = Encoding.UTF8.GetString(data);

                // Print out the received string for debugging.
                Debug.Log(strRecieveUDP);
            }
            catch (Exception e)
            {
                Debug.Log(e.ToString());
            }
        }
    }
```

*Figure 3: On the engine side, an object listens to a defined network port and waits for data, when it is received it is converted to a string and saved.*

[14] "ZJP" "Re: [Windows] UDP Voice Recognition Server." *Unity3D Forums.*

Figure 22 displays the core of the server side code which runs with .NETs voice input engine in the background. When the input engine receives a recognized grammar it is sent over the PC's internal network where, as shown in Figure 3, a thread is always listening for the next spoken word. By using these systems in tandem voice input was implemented in a manner that met all the project's needs.

### 3.1.3 Accelerometer

Accelerometer input was much more of a challenge to implement than initially thought. The design consists of an accelerometer (and other required components) mounted to a headset which would track the player's head movements. The chosen design was to use an Arduino microcontroller to read data from the accelerometer and output data through a serial connection to the computer. The Arduino Uno was chosen due it already being owned by a group member, so no purchase was necessary, and programming for an Arduino is relatively simple compared to other microcontrollers.

The programming for the accelerometer itself was actually not difficult. A free-to-use code library was found online which simplified the process greatly. This library was built specifically for the chosen accelerometer, the MMA8452Q,[15] to connect to the Arduino. The connection between the Arduino and the accelerometer is shown in Figure 4.

[15] "MMA8452Q Datasheet." *Freescale Semiconductor Inc.*

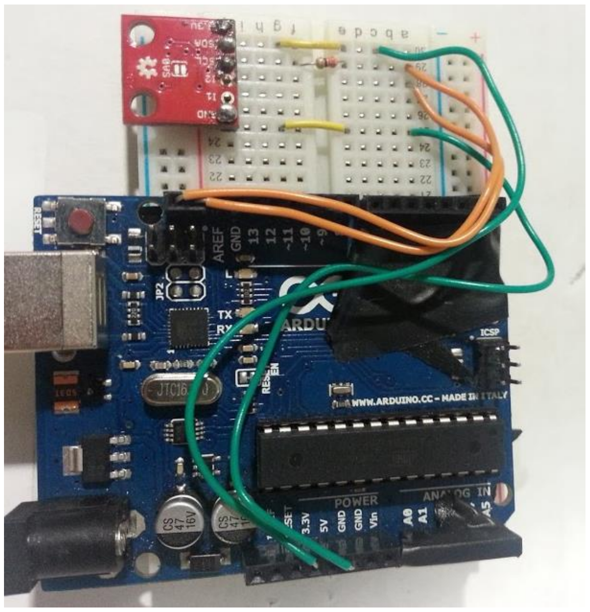
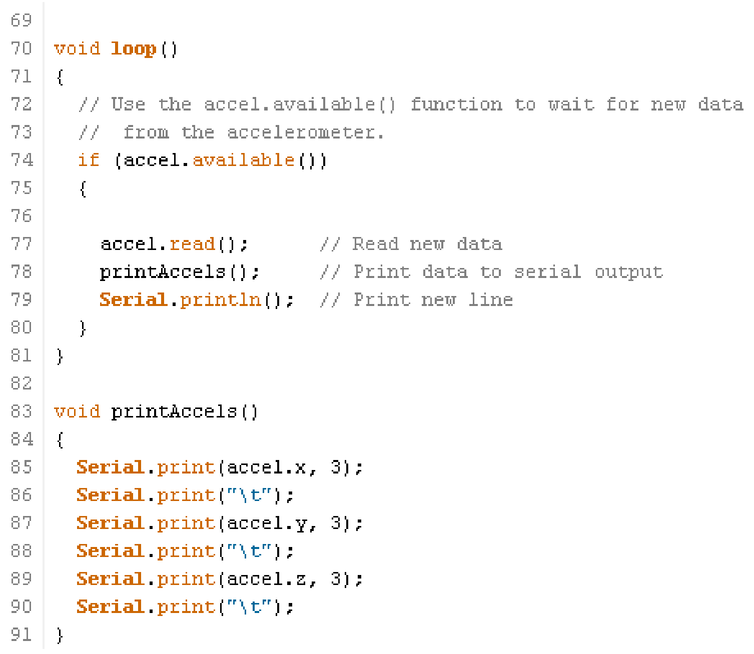


Figure 4: The accelerometer connected to the Arduino

Figure 4 shows the circuit connecting the accelerometer to the Arduino. The accelerometer pins are connected to the corresponding pins on the Arduino board as specified in code that was provided with the accelerometer. This design is terribly inefficient, but it performs its job well for now. In a future prototype, the Arduino chip would be removed from the programming board and all components would be attached to a perforated board or a PCB (Printed Circuit Board), making the entire circuit a lot smaller and easier to mount. The only reason the programming board remains attached is its function as the serial connector; without this board, there would be no way to connect the Arduino chip to the computer, as the team does not have access to the required connectors.

Figure 5 shows the code required to implement the accelerometer. This code is very simple and uses the provided library for the MMA8452Q. The code detects whether the accelerometer is connected to the board or not, and if it is, it reads the data from the accelerometer. It then prints this data as a set of three coordinates (X, Y, and Z), separated by commas, to the serial port. This way, the data can easily be split apart and used in Unity as three separate pieces.

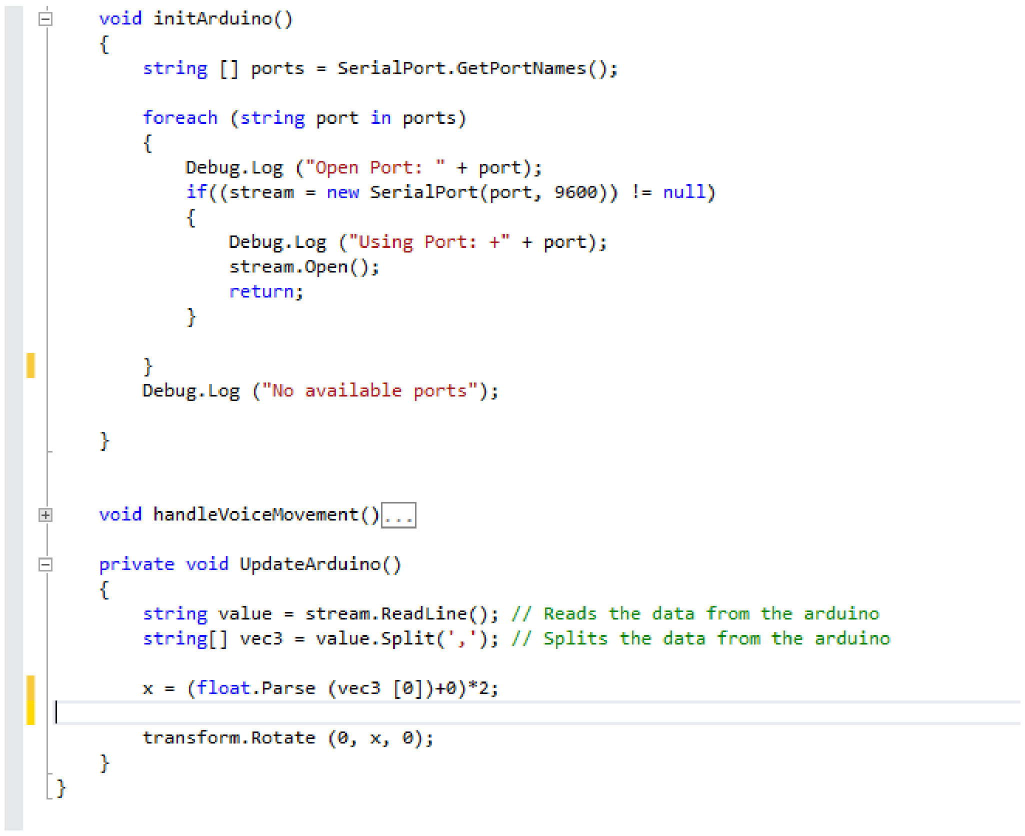


```

void loop()
{
  // Use the accel.available() function to wait for new data
  //  from the accelerometer.
  if (accel.available())
  {

    accel.read();      // Read new data
    printAccels();     // Print data to serial output
    Serial.println();  // Print new line
  }
}

void printAccels()
{
  Serial.print(accel.x, 3);
  Serial.print("\t");
  Serial.print(accel.y, 3);
  Serial.print("\t");
  Serial.print(accel.z, 3);
  Serial.print("\t");
}
```

*Figure 5: The Arduino reads data if the accelerometer is detected and outputs as serial data.*

One of the biggest problems encountered with this process is that the chosen accelerometer will not detect rotations in every direction. The original design was to place the accelerometer module on top of the headset so it could easily be mounted and not bother the user. However, it was discovered that when placed flat, the accelerometer could not detect head rotations to the sides. The work around for this issue for the first prototype was to mount the module on to the side of the headset, on one of the ear-pieces, but this could easily bother the user. The final design places the module on the back of the user's head, connected to the headset by a strap. An ideal solution to this problem would be to use a 6-axis accelerometer instead of a 3-axis, which is what was used. However, this would raise the cost of the project substantially.

The second step of the process was inputting the serial data into Unity. This was relatively simple as well. Unity has built-in support for serial input, so it was simply a matter of adding a few lines of code and using the data. First, the serial port had to be opened. To ensure this would work on every computer, the initialization will scan the computer's communication ports and select an open port to work with the Arduino. Once the game actually begins, the coordinates are input through the serial connection and split into three values (X, Y, and Z). As the game currently only uses rotations on a single plane, only one coordinate is used. Surprisingly, the only modification necessary for the input data was a multiplication to speed up the rotation. The initialization and the update for each frame are shown in Figure 6.



```
void initArduino()
{
    string [] ports = SerialPort.GetPortNames();

    foreach (string port in ports)
    {
        Debug.Log ("Open Port: " + port);
        if((stream = new SerialPort(port, 9600)) != null)
        {
            Debug.Log ("Using Port: +" + port);
            stream.Open();
            return;
        }

    }
    Debug.Log ("No available ports");

}

void handleVoiceMovement()...

private void UpdateArduino()
{
    string value = stream.ReadLine(); // Reads the data from the arduino
    string[] vec3 = value.Split(','); // Splits the data from the arduino

    x = (float.Parse (vec3 [0])+0)*2;

    transform.Rotate (0, x, 0);
}
}
```

*Figure 6: The Player Movement script first initializes the Serial connection, then reads new data each frame.*

As shown in Figure 6, the UpdateArduino() function, which inputs and processes data from the Arduino each frame, is very simple. It reads in a set of three coordinates, splits them based on the locations of commas (which separate the coordinates), and creates a float using the necessary data. The current code allows for an offset, which may be necessary based on the location of the accelerometer relative to the player's head, and it allows a multiplier to speed up or slow down the rotation.

## 3.2 Game Development

In order to actually demonstrate the technologies developed for this project, an actual game is needed. As this project is mostly a proof-of-concept, a full-length game is not required. Throughout the development process, many different ideas for games were discussed which would demonstrate exactly what has been done. In the end, a simple mini-game was selected as it would demonstrate the technology in a way that does not require any tutorial or previous knowledge to understand. With a simple game, more focus can be placed on the technology itself and its applications rather than trying to design a fully-functional game. During the process of creating game ideas, a prototype sound environment was being built to familiarize the team with the tools available. Once a game design was chosen, this environment was expanded greatly to fit the game's needs.

### 3.2.1 Early Game Ideas

Early in the development process, a few different game ideas were created, all of which would still be viable for future games. The first idea the team had was to make the player the captain of a pirate ship. In this game, the player would be the pirate captain, listening to their surroundings and commanding the ship's crew to perform tasks. These tasks would range from steering the ship and speeding up to firing cannons at approaching enemies. Audio cues such as fleeing seagulls, other ships' movements, and the crew yelling would alert the player of what to do. In this game, the sound quality would need to be very high, as the player would want to feel as if they are actually on a ship in the ocean, and the crew members would each need a unique voice to make the crew feel more realistic. Unfortunately, this idea was scrapped for the final project. Though it would have probably been manageable in the time allotted, it was determined that this game would be too difficult for a basic tech demo. The player would need to know all of the possible commands to give their crew, and it would be difficult to differentiate between different objects moving in the water around the ship. It was determined that this game would be better suited for a platform that focuses on audio, but also includes some visual feedback so that the player can perform better.

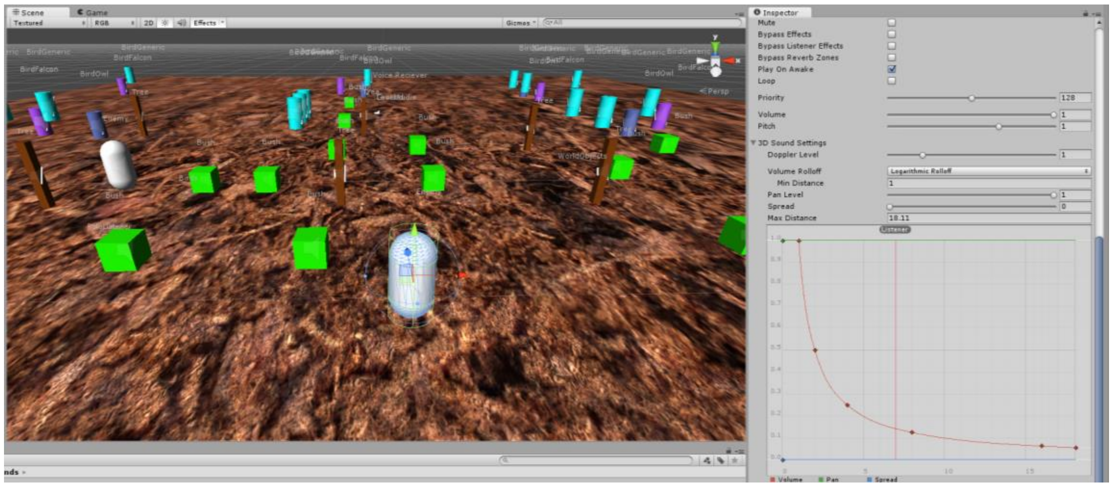
The second idea for a game is something like a standard "text adventure" game, where the player follows a story and makes choices as the game progresses. In this type of game, the player performs a few actions or solves a puzzle, and is then presented with some sort of narrative or story in which they make a choice. For this game, different environments would be used to fit the story, such as a forest or a busy city. This idea was also scrapped due to the scope of the project. For a narrative game, a story would need to be written and all the narration would need to be recorded. As none of the group members have experience with writing a fully interactive narrative, and recording the voice lines would take a great deal of time.  In the future, this type of game would work perfectly on the developed technology, but there was simply not enough time to write a game of this scope.

### 3.2.2 Marco Polo Mini-game

One of the final ideas discussed was simply having the player attempt to navigate a maze. This would be done by having some sort of guide (a beeping sound) in the center to let the player know where to go. However, this idea was deemed too difficult for the player, as they cannot see where they are going or where they have been. However, the idea of a guide led to the development of a mini-game in which the player is playing Marco Polo with a group of NPCs (non-player characters). Marco Polo is a game in which one player is blinded and must find and tag other players. This player must call out to other players (by saying "Marco") and the other players respond (by saying "Polo"). By listening to the voices of the others, the blinded player can try to reach the others, and whoever they tag takes their place instead. Since this is a single-player demo, the other players contain simple AI that makes them wander around the map and avoid the player, and they will always respond when the player calls to them.

### 3.2.3 Sound Environment

The development of the prototype sound environment focused primarily on providing the player with a vivid environment in which the player could orient solely through sound. Care was taken to make sure the sound effects were not overwhelming, so finding the NPCs based on their voice responses is a feasible task. To do this, the sound environment was gradually built up, beginning with an ambient forest background track. Each element (birds, trees, etc.) was then added as an AudioObject, its sound was tuned to blend in with the current environment, and its audio range was adjusted. For example, the sound from wind blowing through trees is loud only when the player is in close proximity to the tree, and the fade distance is very short. Birds, on the other hand, remain relatively loud for a longer distance. Figure 6 shows a screenshot of development in Unity. On the left side of the screen, all of the game's objects can be seen, each of which have an assigned audio track. On the right side, the basic audio editing tools are shown. The graph at the bottom shows the drop-off of the sound over distance; in this example, the selected object is one of the NPCs, and the drop-off is sudden so that the "Polo" responses are very quiet if the player is far away, but very loud if the player is close.



Mute ☐
Bypass Effects ☐
Bypass Listener Effects ☐
Bypass Reverb Zones ☐
Play On Awake ☑
Loop ☐
Priority 128
Volume 1
Pitch 1
3D Sound Settings
Doppler Level 1
Volume Rolloff Logarithmic Rolloff
Min Distance 1
Pan Level 1
Spread 0
Max Distance 18.11

*Figure 7: Unity Editor View of the Sound Levels setup within Unity's 3D Audio System*

Figure 7 shows the editor view of Unity when working with audio. The left side of the screen shows the objects in the game, each of which has their own AudioObject. The right side of the screen shows the drop-off distance for the audio tracks. For example, the selected object, an NPC, drops off quickly as their voice should be quiet from far away.

Due to engine limitations, sound options are severely lacking; it was difficult to provide a variety of sound without adding multiple different audio tracks to the same object using scripts. Unity is not particularly great for creating a sound environment without a great deal of modification. Despite the limitations, the sound environment does sound semi-realistic, though Unity would not be the top choice for a future project of this kind.

# 4 Results

Since this project involved constructing a physical model and developing a working game, testing to ensure that everything actually worked was crucial. The different parts that had to be tested were the input modules and hardware, the audio environment, and the actual usability of the final product. Based on the results from these tests, it would be determined if this idea would be practical for future development.

## 4.1 Input

The two input modules, the voice recognition and the motion control, needed to be tested both independently and together. As both parts were developed separately, there was no guarantee that they would work together without causing any problems. Fortunately, as both parts were separate, they did not interfere with each other at all.

### 4.1.1 Voice Recognition

The voice recognition system used for this project proved useful and effective. A number of user tests were conducted to build an optimal design model for ensuring an intuitive voice recognition system. In other words, multiple users were tested to be sure the system could recognize words despite accents or different pronunciations. It was discovered that the software itself functioned perfectly well on its own, and the choice of recognizable words was the deciding factor in its success. Often, even with a tutorial, users would try to use different words (often synonyms) that the system would not recognize, i.e. replacing "forward" as a move command with the word "go." In an ideal and polished system, a multitude of similar words would be mapped to the same action to ensure that users could make use of their own unique ways of speaking.

### 4.1.2 Motion Control

The motion control system works as intended, though there are a few small issues. As discussed previously, there was not an easy way to mount the accelerometer unit onto the headset, so it would need to be strapped on to the back of a headset. Once mounted and enclosed, this system would not have any issues. In the game, the player character rotates as long as the user turns their head in the corresponding direction. Currently, as there is no need for vertical rotation, the accelerometer only detects rotations to the left and right, but this would change for any game requiring more directions.. Unfortunately, there is still a small problem with the accelerometer itself: If the user turns too suddenly, or if the module is shaken, the accelerometer will stop responding, which will cause the game to crash. This issue is most likely due to a loose connection between the Arduino and the computer, as the cable connection on the Arduino is weak. This issue would be fixed by enclosing the module in a more stable package, but for this prototype, this was not done.

## 4.2 Reception

After the prototype was finished, user testing was required to see if it was successful or not. Testers gave feedback on the voice recognition system, the motion controls, and the overall game design. For a future prototype, user criticism would be taken into consideration to fix any problems there might be.

### 4.2.1 Voice Recognition

The voice recognition system fared well during the user testing stages of this project. Users often found it naturally intuitive, as they were able to jump right into gameplay given very little instruction, which is an excellent sign for general usability. This being said, the system was also a source of confusion and frustration for some individuals. This confusion stemmed from the inherent input lag that the system has, which often resulted in users being unsure of what the last registered command was. In addition, the lack of an expansive library of suitable voice commands greatly constrain some users, often forcing them into using speech they were not necessarily accustomed to.

### 4.2.2 Motion Control

The motion control system performed well during testing in most cases. The turning mechanics are not intuitive to all players, unfortunately; many testers hoped that turning their head would cause a 1-to-1 turn for the in-game character. However, this would not work due to the limitations of using a wired device, as the user would eventually end up tangled in the wires from the headset and the Arduino. Currently, the in-game character will keep turning as long as the user keeps their head turned in that direction, and will turn faster if the player's head is turned farther. Once they understand how it works, most users adjusted well.

### 4.2.3 Game Design

Users enjoyed the concept of the Marco Polo prototype and were generally successful in finding the NPCs spread throughout the forest. However, some users became frustrated with the initial difficulty in becoming immersed in the world, which is to be expected. This is due in part to Unity's inability to truly replicate actual 3D sound physics; instead, it estimates the expected results. These estimated results are then reproduced in stereo, a 2-point audio system which provides very little information of where a sound is coming from other than left and right. That being said, after given enough time to familiarize themselves with the soundscape, users appreciated the environment and felt the forest was immersive. The motion control system supported the user's sense of localization by allowing them to quickly pivot in order to face sounds that they hear. This helped to alleviate the limitations of stereo sound. Another factor that helped bypass the engine's limitations was the usage of high-quality stereo headphones, which provide a better sound stage and less confusing audio.

# 5 Conclusion

The goal of this project was to create a gaming platform prototype for the visually impaired, and to create a game to demonstrate this platform. The team's purpose is to show that it is feasible to make video games for people with impairments without sacrificing any ease-of-use or enjoyment. This prototype was supplemented by research into what exactly makes a great accessible game, and what is required in a successful audio-based game.

The research performed for this project has shown what exactly is required for an accessible audio game in order to be successful. Based on this research, it has been determined that the key criteria for a good audio game are high-quality sounds and ambience, and intuitive gameplay; if the sound quality is low, users will not be immersed in the world, and if the gameplay is not enjoyable and easy-to-learn, there will be no reason for the player to want to continue playing. This research also showed what needs to be avoided in a game, such as highly-repetitive sounds and confusing controls.

The development of the prototype followed the lessons learned from the research and adapted the Unity engine to solve any problems that arose. The creation of the AudioObject class proved invaluable, as this formed the core of all of the engine's audio design. The voice recognition system as a central focus of the platform ended up being more successful than originally believed; almost anybody could easily control themselves without a physical controller, making this platform easily accessible to many people. The VR system is also easily expandable, allowing the game developer to easily create a Grammar that can be recognized. The motion control system, while not as intuitive, also served its purpose of making the game easy to play without a standard controller. Despite some initial confusion and frustration from players, both of these input methods are easy to learn and understand, making them perfect for a game platform built for accessibility once the player knows how it works.

Creating a soundscape that sounds realistic was one of the most challenging aspects of this project. This is mostly due to the limitations of the Unity engine (and most 3D game engines), as it was not built for sound design. The other challenge was making things sound like they do in nature, as sounds in the real world are usually often random or unpredictable. Overcoming these challenges proved that it is possible to make a sound environment that sounds realistic despite limitations from the engine's stereo sound system.

The final part of this project was actually building a game that would demonstrate the technology used. The Marco Polo mini-game ended up being perfect for this purpose, as it requires the player to use voice controls and locate sounds via head movements. Due to the aforementioned engine limitations, actually locating the NPCs proved difficult for some players, though there is nothing that can be done about this problem in the current implementation. Overall, however, this game proved enjoyable to players, and it showed off the technologies well.

In the future, many things would be changed for this project. The biggest change would be choosing or creating a new game engine. Unity is great for creating small games and it is free to use, but it is not specialized enough in making 3D soundscapes. A new engine would need to be used with support for surround sound and actual 3D audio tools. Other future improvements would be the addition of a larger library of available voice commands, which was one of the biggest complaints. Additionally, a new accelerometer circuit would be needed to easily mount onto a headset, as the current model is unwieldy and aesthetically awful. However, despite the problems faced and the improvements that could be used, this project successfully showed that it is possible to build a game platform entirely based on audio input/output and motion. For the future of accessibility gaming, this project will serve as a step forward, showing the possibilities of already existing technologies and the advancements that can be made.

# 6 REFERENCES

"The All New Oculus Rift Development Kit 2 (DK2) Virtual Reality Headset." *Oculus VR*. N.p., n.d. Web.

Blunsom, Paul. "Hidden Markov Models." Thesis. Utah State University, 2014. Print.

Campbell, Colin. "Here's How Many People Are Playing Games in America." *Polygon*. N.p., 14 Apr. 2015. Web.

"Consolidated Sales Transition by Region" (PDF). *Nintendo*. n.d. Web.

"MMA8452Q Datasheet." *Freescale Semiconductor Inc.* Oct. 2013. Web. <https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/MMA8452Q-rev8.1.pdf>.

"Game Accessibility Guidelines | A Straightforward Reference for Inclusive Game Design." *Game Accessibility Guidelines RSS2*. N.p., n.d. Web.

Hermann, Thomas. "AcouMotion." *Sonification*. N.p., 19 Apr. 2006. Web.

"National Data." *Vision Health Initiative (VHI)*. Centers for Disease Control and Prevention, 28 Sept. 2009. Web.

Pinola, Melanie. "Speech Recognition Through the Decades: How We Ended Up With Siri." *TechHive*. N.p., 2 Nov. 2011. Web.

"Research Shows Rehabilitation Benefits of Using Nintendo Wii." *NewsWise*. Rutgers University, 23 Sept. 2008. Web.

"There Came an Echo." *Kickstarter*. N.p., n.d. Web. <https://www.kickstarter.com/projects/iridiumstudios/there-came-an-echo-0/description>.

Tullis, Ryan. "Blind Player Competes at Evolution, Says, 'If You're Playing Me... Don't Hold Back'" *EventHubs*. N.p., 11 Aug. 2014. Web.

"Welcome to Includification - Actionable Game Accessibility." *Includification*. N.p., n.d. Web.

Yuan, Bei, Eelke Folmer, and Frederick C. Harris, Jr. "Game Accessibility: A Survey." *Springer-Verlag*, 01 Mar. 2011. Web.

"ZJP" "Re: [Windows] UDP Voice Recognition Server." *Unity3D Forums.* N.p., 4 Mar. 2013. Web. <http://forum.unity3d.com/threads/windows-udp-voice-recognition-server.172758/>.