Gesture Based UI Project

TetrisTake2

Myo Armband and Voice Controls

Link to Github Repository:

https://github.com/EvanGreaney/GUIProjectRepo

Introduction:

For this project, I was tasked with developing an application with a Natural Interface, there were many options I could have taken to develop this project, from creating a game or application that used the Xbox Kinect to track body movements to control the game or used Nintendo Wii remotes to control an object within a game.

For my game, I decided to primarily focus my project on using the Myo Armband but have aspects of voice control within my project to control the main menu. I first decided to design a clone of the game Tetris but incorporated the Myo armband as the controller for the game and used voice controls to operate the main menu.

Purpose of the application:

For this application, I wanted to blend different gestures within a game and try to create a game that did not rely on a keyboard and mouse or controller but instead tried to rely on the player itself to control the game. Because of this, I decided to incorporate the Myo Armband for controlling the game and voice controls for navigating to the game from the main menu. The game is simple in design but shows how the Myo armband can be implemented to control the game.

The game is based on Tetris and the game aims to move the Tetrimino Blocks from left to right or rotate to fill rows of blocks to clear lines to earn points. It demonstrates how these gestures can be incorporated into the game and how by using simple gestures, it can control the game with some ease.

Gestures Identified as appropriate for this application:

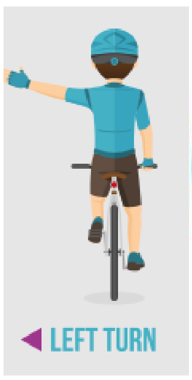
There are two main types within this project, that Myo Armband Gestures and Voice Controlled gestures. These types of gestures candle handle up to 90% of all interactions within the game and allow the player to engage with the game by just using their arm and voice. The game itself is also entirely playable using a keyboard and mouse for those, who wish to not use gestures.

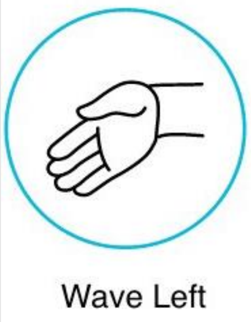
The Gestures used within the application will be explained below:

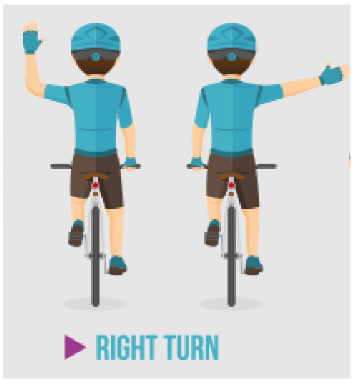
My Armband Gestures:

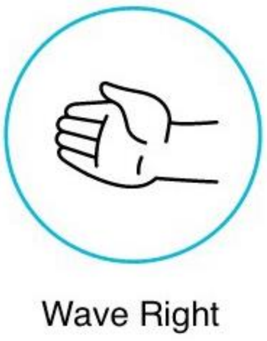
Wave In and Wave Out:

I incorporated these gestures to control the movement of a block to move to the right and the left, I decided to use these gestures as they reminded me of the hand signals used by cyclists when indicating on the direction they are turning, I believed these gestures to be great gestures for moving left and right as people who are capable of driving and cycling are mostly aware of the gestures and understand what they mean so I felt these gestures would be easily transferable to the game and wouldn't be difficult to understand.



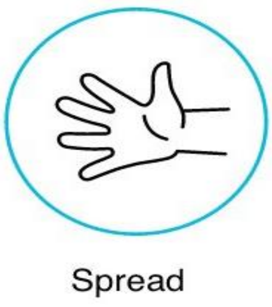






Fingers Spread:

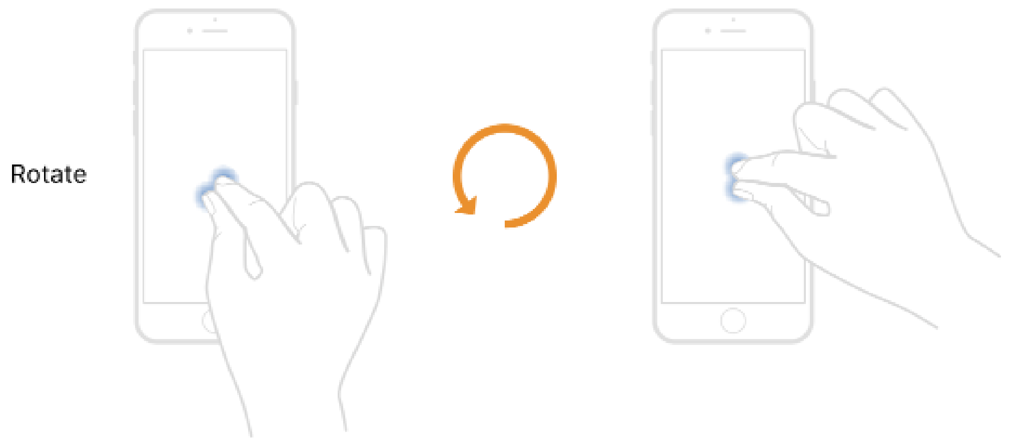
The next Myo Gesture I incorporated, was that of the Fingers Spread gesture, through testing different methods of incorporating the fingers spread gesture, I found that when the user rotates their wrist to almost a 90-degree angle and does the fingers Spread Gesture, it is very similar to the gesture used for a stop around the world. Therefore used this gesture to pause the game simulating stopping the game. I also decided to use this gesture for my resume as well as I found that in many games that I have found use a button to pause a game but also use that same button again to resume the game, I felt this would come as second nature to people who play games and that would do this without even thinking about it.



Double Tap:

For this gesture I decided to use it to rotate the blocks, originally I had planned to use the rotate gesture using the X and Y axis but found that the block would freely rotate and not snap rotate as required for the game itself as each block needed to align precisely with other blocks within the same row otherwise the game would not work. I then searched for similar hand gestures used in other technologies for rotating. After researching I found that Apple uses a similar gesture for rotating an object on the screen.

How it is done is where the user places their index finger and thumb together, then places them on the screen, and then rotates to apply the gesture. The double-tap gesture was the most similar out of all the gestures to this as it used the same motions to apply the double-tap except for rotating the fingers on the screen. After testing the double-tap gesture with the rotation, I found it was a very fast reaction for rotating and allowed the user to snap the block into place even when it is at its most difficult.



Voices Gestures:

For my Main Menu, I use voice recognition to navigate from the menu to the game, to open the options menu, or to exit the game when the project is built. The player can use these as well as the keyboard and mouse to navigate throughout the game.

The game takes common phrases to operate different functions within the main menu like starting the game. It does this by using a series of rules that when a specific phrase is listened to, the game performs a specific action.

Rules:

Start Game – This rule starts the game

Phrases:

- Start game
- Start a new game
- Begin a new game
- New game
- I want to play

All these phrases that when spoken, will start the game for the user.

Quit Game – This rule quits the game

Phrases:

- finish the game
- exit game
- exit
- quit game
- quit
- I give up

When spoken these phrases exit the game entirely.

Options – This Rule opens the Options Pane

Phrases:

- Show me the controls
- Options
- open Options
- Settings
- open Settings

When spoken these phrases open the options pane and deactivate the main menu pane simulating going to another scene.

Main Menu – This Rule opens the Main Menu Pane.

Phrases:

- Return to Main Menu
- Return to the Main Menu
- Main Menu
- Back to Start Screen

- Back to the Start Screen

When spoken these phrases open the Main Menu pane and deactivate the options pane simulating returning to the Main Menu.

Hardware used in creating the application:

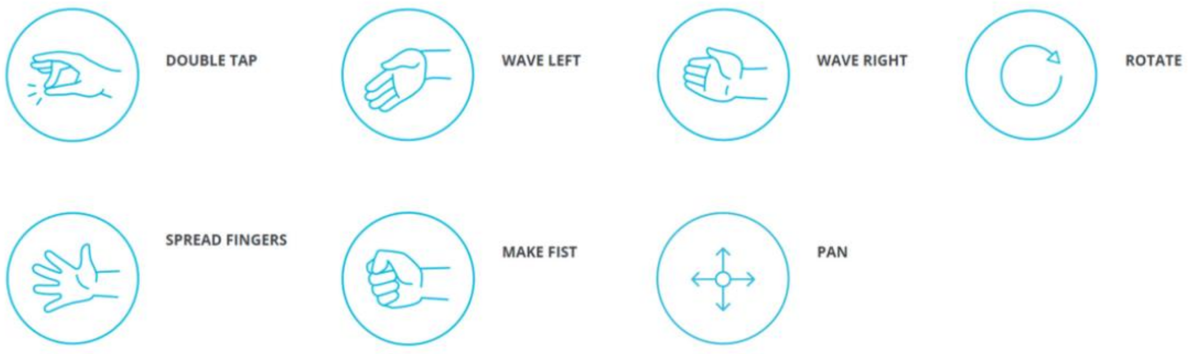
The Game uses two types of gestures within the game, these gestures require specific hardware to make the functionality of the gestures work. The Hardware used in this application is:

- Myo Armband
- Microphone for voice gestures

Myo Armband

What is a Myo Armband?

The Myo Armband was designed by the company by the name of Thalmic Labs, the band was designed to track the wearer's movements and translate them into commands for a computer. It has done this by when the band was wrapped around your forearm and its eight EMG(Electromyogram) sensors would detect the electrical signals in your muscles when a specific action or gesture was performed. There are seven distinct gestures designed for the Myo Armband which can be seen below. These gestures show all the gestures that are available to all Myo Armband users.



Why choose the Myo Armband over other technologies?

I decided to use Myo over other devices such as the Kinect and the Oculus Rift for two reasons, the Myo Armband is relatively accurate when trying to perform a gesture compared to that of the Xbox Kinect where I found that the Xbox Kinect misses more gestures compared to the Myo Armband.

The next reason is because of simplicity, including Myo into a game is a lot easier than developing a hardware-specific game that is designed for a device like an oculus Rift. With Myo, I was able to develop a game that is playable with Mouse and Keyboard and the Myo Armband and can be used either together or separately.

Voice Control

What are voice Controls?

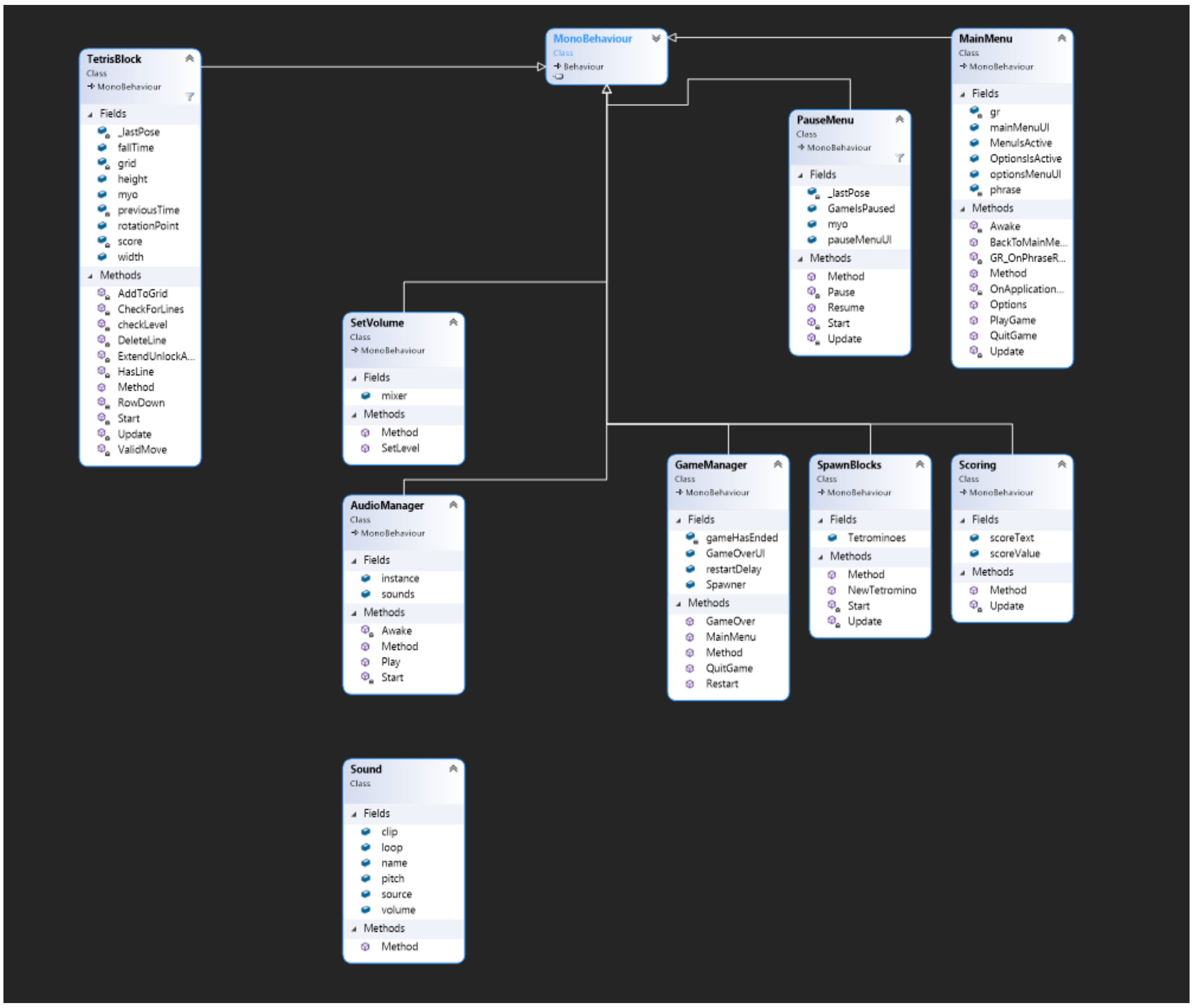
These voice controls are a set of phrases and sentences that when spoken will execute an action within an application, in this case for my game, I used this to control the Main Menu Interaction of the game and allowed the user to move from the main menu to the game with just a single phrase.

Why choose voice controls over other technologies?

The main reason I chose voice control over other technologies is for the simplicity of just using your voice to control the game, the player doesn't have to anything else except for a microphone or headset, they don't necessarily have to have a fancy controller or VR device to use the game, they just need a decent microphone or headset and their voice and they will be more than capable controlling the main menu aspect of the game.

Architecture for the solution:

Class Diagram Illustrating all classes used the game



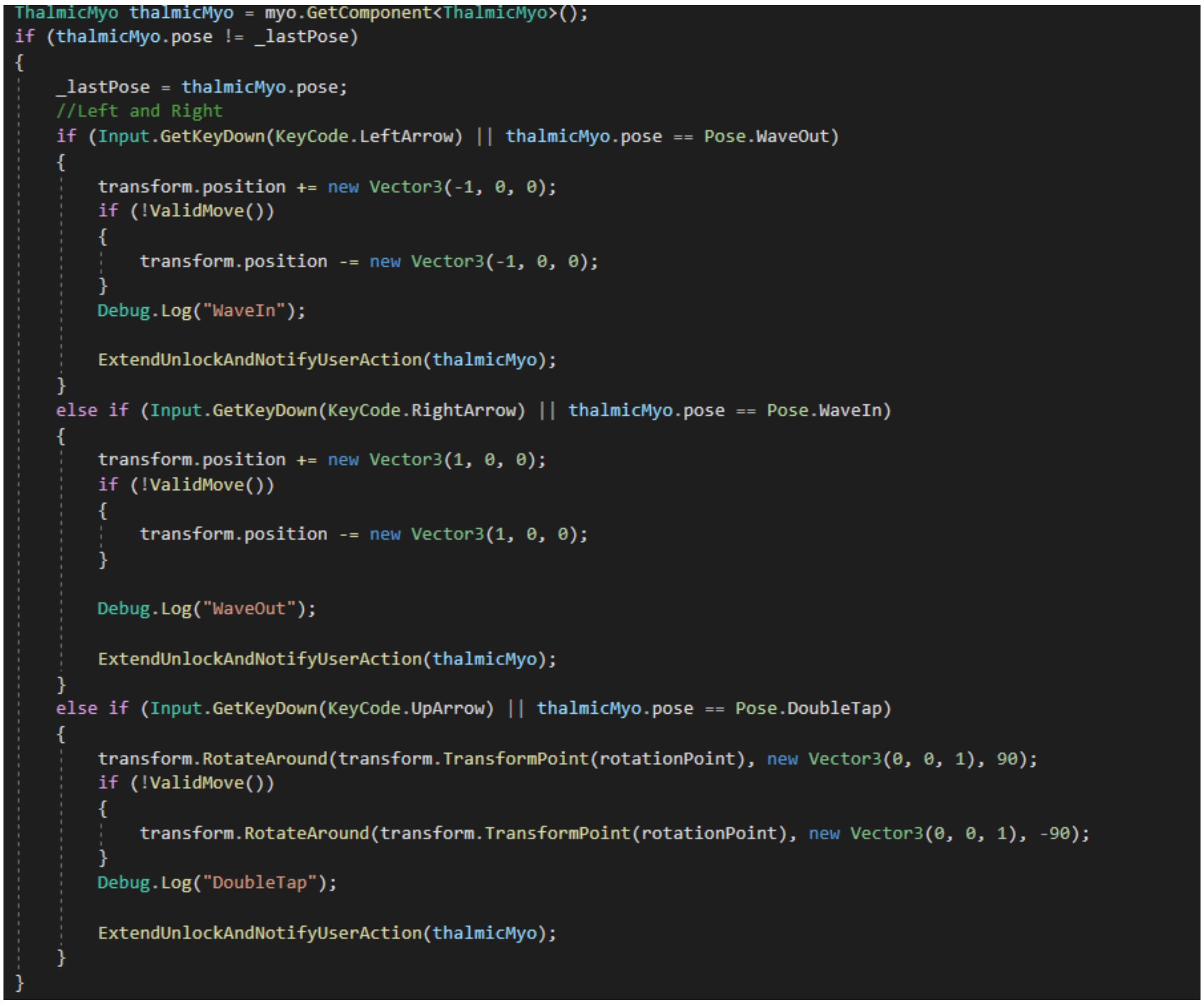
How the Hardware is implemented into the Architecture

The Hardware is implemented in different stages of the game and is used to control different aspects of the game, below will show where they are implemented:

Myo Armband Implementation:

TetrisBlock.cs

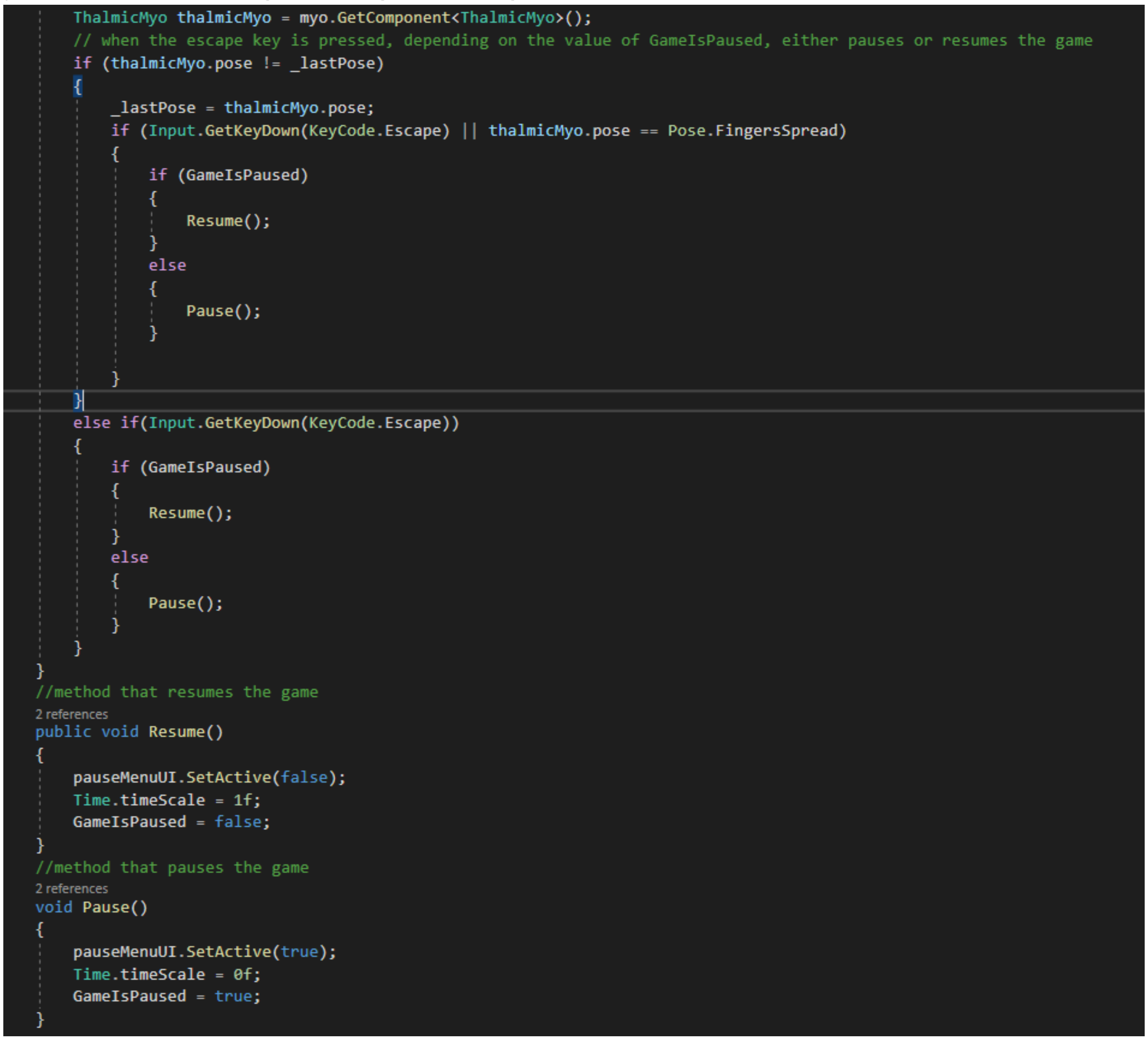
The Myo controls are implemented at the same stage of the update method in TetrisBlock.cs as the left and right arrow keys along with the rotate functionality, this allows for easy movement use and the ability to use both keyboard and mouse and the Myo Armband.



```csharp
ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();
if (thalmicMyo.pose != _lastPose)
{
    _lastPose = thalmicMyo.pose;
    //Left and Right
    if (Input.GetKeyDown(KeyCode.LeftArrow) || thalmicMyo.pose == Pose.WaveOut)
    {
        transform.position += new Vector3(-1, 0, 0);
        if (!ValidMove())
        {
            transform.position -= new Vector3(-1, 0, 0);
        }
        Debug.Log("WaveIn");

        ExtendUnlockAndNotifyUserAction(thalmicMyo);
    }
    else if (Input.GetKeyDown(KeyCode.RightArrow) || thalmicMyo.pose == Pose.WaveIn)
    {
        transform.position += new Vector3(1, 0, 0);
        if (!ValidMove())
        {
            transform.position -= new Vector3(1, 0, 0);
        }

        Debug.Log("WaveOut");

        ExtendUnlockAndNotifyUserAction(thalmicMyo);
    }
    else if (Input.GetKeyDown(KeyCode.UpArrow) || thalmicMyo.pose == Pose.DoubleTap)
    {
        transform.RotateAround(transform.TransformPoint(rotationPoint), new Vector3(0, 0, 1), 90);
        if (!ValidMove())
        {
            transform.RotateAround(transform.TransformPoint(rotationPoint), new Vector3(0, 0, 1), -90);
        }
        Debug.Log("DoubleTap");

        ExtendUnlockAndNotifyUserAction(thalmicMyo);
    }
}
```

Pause.cs

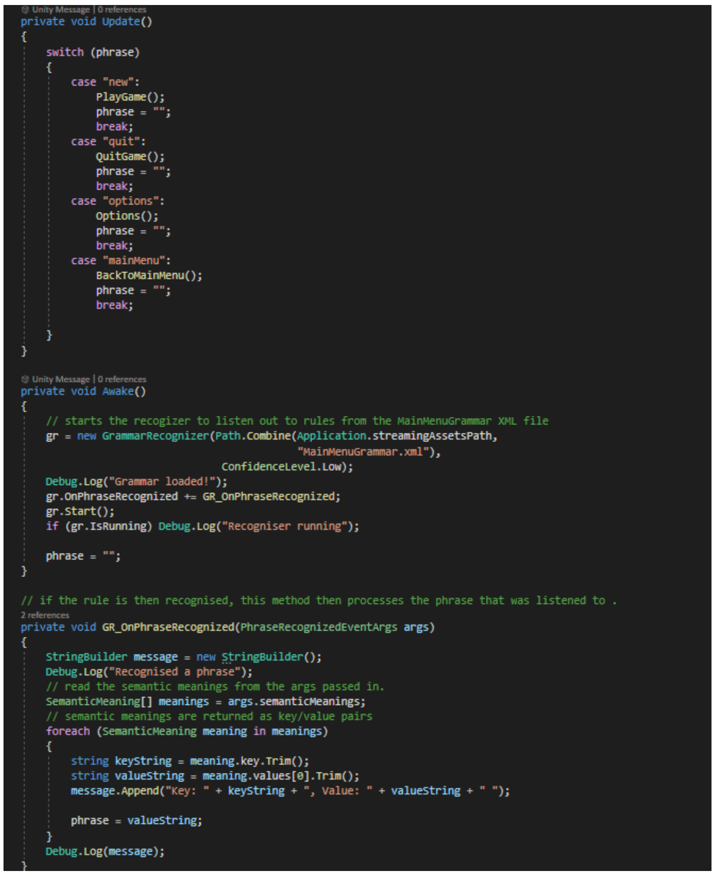
The next Myo Gesture is found in the Pause.cs script where the user uses the Fingers Spread to pause and resume the game using the same gesture.



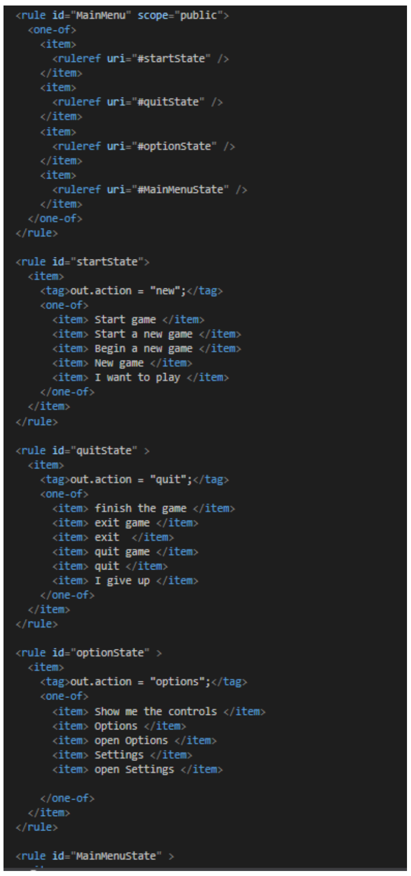
```
    ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();
    // when the escape key is pressed, depending on the value of GameIsPaused, either pauses or resumes the game
    if (thalmicMyo.pose != _lastPose)
    {
        _lastPose = thalmicMyo.pose;
        if (Input.GetKeyDown(KeyCode.Escape) || thalmicMyo.pose == Pose.FingersSpread)
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }

        }
    }
    else if(Input.GetKeyDown(KeyCode.Escape))
    {
        if (GameIsPaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }
    }
}
//method that resumes the game
2 references
public void Resume()
{
    pauseMenuUI.SetActive(false);
    Time.timeScale = 1f;
    GameIsPaused = false;
}
//method that pauses the game
2 references
void Pause()
{
    pauseMenuUI.SetActive(true);
    Time.timeScale = 0f;
    GameIsPaused = true;
}
```

Voice Controls Implementation

The voice controls are being implemented as part of the MainMenu.cs Script along with the Main MenuGrammar.xml file, within the MainMenu.cs is the Grammar recognizer that detects when a phrase is spoken and then determines what that phrase meant and performs a specific action based on the phrases outlined in the MainMenuGrammar.xml.



```
Unity Message | 0 references
private void Update()
{
    switch (phrase)
    {
        case "new":
            PlayGame();
            phrase = "";
            break;
        case "quit":
            QuitGame();
            phrase = "";
            break;
        case "options":
            Options();
            phrase = "";
            break;
        case "mainMenu":
            BackToMainMenu();
            phrase = "";
            break;

    }
}

Unity Message | 0 references
private void Awake()
{
    // starts the recogizer to listen out to rules from the MainMenuGrammar XML file
    gr = new GrammarRecognizer(Path.Combine(Application.streamingAssetsPath,
                                            "MainMenuGrammar.xml"),
                                ConfidenceLevel.Low);
    Debug.Log("Grammar loaded!");
    gr.OnPhraseRecognized += GR_OnPhraseRecognized;
    gr.Start();
    if (gr.IsRunning) Debug.Log("Recogniser running");

    phrase = "";
}

// if the rule is then recognised, this method then processes the phrase that was listened to .
2 references
private void GR_OnPhraseRecognized(PhraseRecognizedEventArgs args)
{
    StringBuilder message = new StringBuilder();
    Debug.Log("Recognised a phrase");
    // read the semantic meanings from the args passed in.
    SemanticMeaning[] meanings = args.semanticMeanings;
    // semantic meanings are returned as key/value pairs
    foreach (SemanticMeaning meaning in meanings)
    {
        string keyString = meaning.key.Trim();
        string valueString = meaning.values[0].Trim();
        message.Append("Key: " + keyString + ", Value: " + valueString + " ");

        phrase = valueString;
    }
    Debug.Log(message);
}
```



```
<rule id="MainMenu" scope="public">
  <one-of>
    <item>
      <ruleref uri="#startState" />
    </item>
    <item>
      <ruleref uri="#quitState" />
    </item>
    <item>
      <ruleref uri="#optionState" />
    </item>
    <item>
      <ruleref uri="#MainMenuState" />
    </item>
  </one-of>
</rule>

<rule id="startState">
  <item>
    <tag>out.action = "new";</tag>
    <one-of>
      <item> Start game </item>
      <item> Start a new game </item>
      <item> Begin a new game </item>
      <item> New game </item>
      <item> I want to play </item>
    </one-of>
  </item>
</rule>

<rule id="quitState" >
  <item>
    <tag>out.action = "quit";</tag>
    <one-of>
      <item> finish the game </item>
      <item> exit game </item>
      <item> exit  </item>
      <item> quit game </item>
      <item> quit </item>
      <item> I give up </item>
    </one-of>
  </item>
</rule>

<rule id="optionState" >
  <item>
    <tag>out.action = "options";</tag>
    <one-of>
      <item> Show me the controls </item>
      <item> Options </item>
      <item> open Options </item>
      <item> Settings </item>
      <item> open Settings </item>

    </one-of>
  </item>
</rule>

<rule id="MainMenuState" >
```

Conclusions & Recommendations

From completing this project, I have learned how to implement two types of hardware into a classic game that works seamlessly with traditional controls or independently from the controls. As a result of this, I have come to appreciate the difficulties of implementing hardware to software and taking input from a piece of hardware, and having them control the application.

What I would've done differently?

If given more time like a year to complete this project I would've like to have incorporated more hardware into the design and potentially not create a game but instead have the Myo Armband and Voice commands be a controller for the likes a car, robot …etc. outputting directions to the likes of a Raspberry Pi or Arduinos built into the car or robot.

Concerning the project I would've liked to try and make the game entirely independent of traditional controls and instead controlled by each device individually or simultaneously, to show that traditional controllers aren't the only way to play games.

After completing the project, I found that like with every hardware and software, there are setbacks to be found like with the Myo Armbands. It is a fantastic piece of technology but the major drawback was that ThalmicLabs discontinued the Myo Armband and currently there is no longer support for the device, hence the lack of gestures that can be implemented and documentation to implement them, because this I may have switched technologies as to work with more supported hardware otherwise for what is available, the technology was brilliant to work with.

In conclusion, I found this project to be rather enjoyable with the experience of being able to implement hardware as part of my game is a rather wonderful experience to have.

References:

1. https://roadcyclinguk.com/how-to/technique/essential-guide-road-cycling-hand-signals-calls.html
2. https://developer.apple.com/documentation/uikit/touches_presses_and_gestures/handling_uikit_gestures/handling_rotation_gestures
3. https://github.com/thalmiclabs/myo-unity