Gesture Based UI Development project

"Tilty Table", a Myo Armband controlled Unity game

By Ronan Hanley and Tommey Faherty

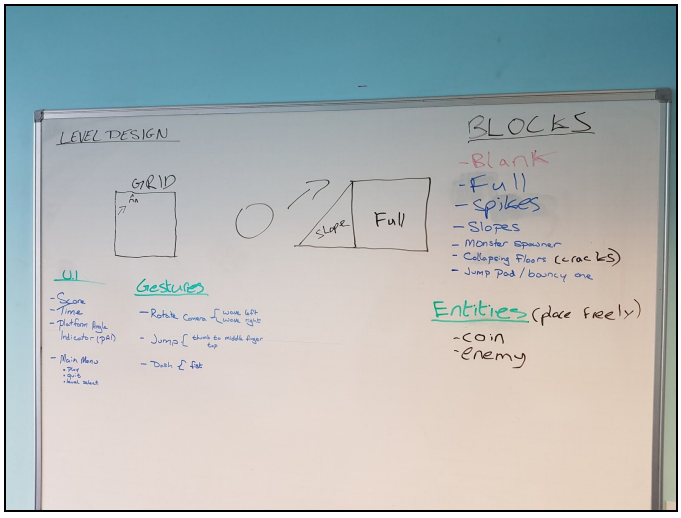
GitHub URL:
https://github.com/TommeyFaherty/gesture-based-ui-project

Purpose of the Application

"Tilty Table" is a game made, in Unity, to be used with the Myo Armband. It was designed with the Myo Armband in mind, which made it easier to implement relevant features. It revolves around the player using the gyroscope to tilt and rotate the floor, moving the ball to the end of the level. The player must avoid several obstacles while navigating through each level. To aid the player in overcoming these hurdles, they can jump and dash. The abilities can be activated by using the Myo Armbands gestures feature. We used the gestures in the Myo Armband to rotate the camera's position and allow the ball to jump and dash.

We also initially programmed the game to be usable with a keyboard. So, if the game does not detect a Myo Armband it will default to the keyboard controls. A display of how the controls function, can be located on the Main Menu screen by clicking the "Tutorial" button.

Gestures Identified as Appropriate



Plan of action

The gestures in our game were one of the initial decisions we made as a group. Since our game was designed around using the Myo, we had an idea of several gestures we could use. We used gestures for mobile abilities and functionality.

For the dash ability we used one of the easier to use gestures. The gesture to tap your thumb on your middle finger. The dash ability is a feature which needed to feel easy to use while tilting your arm to use the gyroscope. While ideally all gestures would fit this idea, we picked the easiest gestures for dash and jump. These abilities needed to be the simplest, so the player doesn't get overwhelmed trying to balance the ball on the level as well as trying to register a complex gesture. Arguably, the simplest gesture was given to the dash as throughout most of our levels we used ramps. This meant that gaining momentum on the ramps could pass a hurdle instead of a jump.

The logic behind the gesture for a jump was similar to that of the dash. It had to be easy to do while tilting the level. For this ability we chose the fist gesture. Keeping the ability easy to use allows for the player to make more impromptu decisions while playing the game.

For resetting the axis of the level, we chose the hand spread gesture. We noticed early on when testing the Myo's, that this was a particularly awkward gesture. It seemed basic however, it caused us to strain our hands when getting the Myo to register the gesture. This meant it would be difficult for the player to use if it was something as frequent as jumping or dashing. We set it to reset the level axis as this was a conditional action. We gave the player several tools to navigate and change perspective so, using this gesture isn't as common as the others. Hence the player is less likely to hurt themselves trying to register the gesture.

To change the camera's perspective, we used the swipe gestures. It made a lot of sense and felt very natural and intuitive to use. If the player wished to rotate the camera to the left, they swiped left and vice versa for the right. The camera moved around the ball at fixed 90-degree angles. So, the player had control to rotate the camera but not too much so that it would cause issues while playing or would be a nuisance to use.

Hardware Used

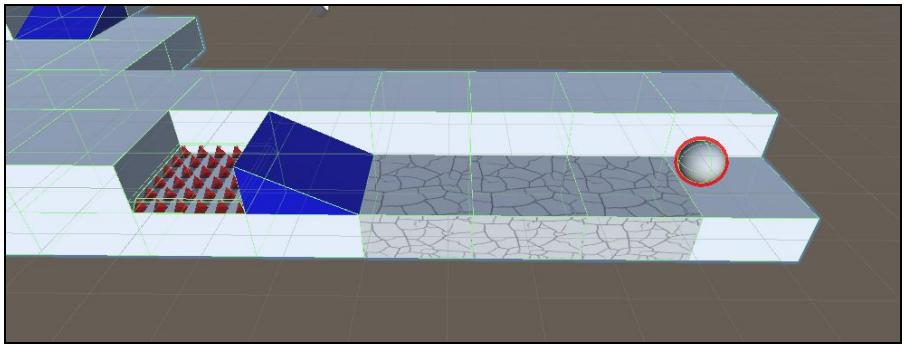
We chose to use the Thalmic Labs Myo Armband for our project. Instead of picking hardware to suit the needs of a project which we wanted to do, we picked a project which could showcase the functionality of the Myo, which we had taken an interest in from the lab. We came up with a few different project ideas for it. Originally, we thought of doing an archery game, where the player wears a myo on both arms, and uses gestures and moves his arms to aim and shoot. Soon after setting out to get started on the project, though, we realised this would be very clunky and awkward to use. It would also require getting hold of *two* Myos every time we wanted to work on the project, which may have not been available. As such, we switched to the *ball on a rotating level* idea, which allowed us to show off the features of the Myo much more easily.

As it became suddenly less clear as to whether or not we would have access to the hardware we needed to complete our project amid the COVID-19 pandemic, we considered using the accelerometer and gyroscope of a smartphone to fulfill the gesture control requirements of this project instead of using a Myo. This would have been a reasonable route to take with the project, since Unity can make builds for Android or iOS without much trouble. However, it would have been a shame to forgo the use of the Myo, as this was quite a unique idea as an input device for a game.

Another alternative idea was to use a webcam, and to use a library such as OpenCV to track the player's hands, arms, or body etc. in order to control the game. Voice commands could be used alongside this to activate the various abilities such as dash and jump. This may have ended up being clunky, inaccurate, and more complicated.Thankfully, we were able to get hold of a Myo to develop our project with, so we were able to continue working towards our original vision of how the game should be controlled.

Architecture

As the game was made in the context of gesture control, we designed it with the use of gestures in mind.The game consists of three main components: the level, the player, and the camera. All three of these components make use of the Myo armband in some way, either through the use of gestures, or movement through the Myo's gyroscope. We designed the levels in a way that forces the player to make use of these gestures, such as having to jump or dash across a gap:



As shown in the above screenshot, we designed levels on a grid system. This made designing levels a lot easier than placing blocks by hand, for example. Instead, we represented each level inside a text file, where each text character represents a block. A Unity script then generates the whole level on the fly every time the scene is loaded. To create the simple models for the blocks, we used Blender. We also applied different materials to them in Unity to make each block stand out. We designed three levels to showcase the game, each in increasing difficulty.

Breakdown of the Scripts

As the project was developed using Unity, all of the classes mentioned here relate to Unity scripts which we developed. They can be divided into five main categories:

1. Level building:
    a. **LevelBuilder** is attached to the *Blocks* child of *Level, and* builds each level based on a text file specifying which blocks go where, using a mapping specified by **BlockCharacter**. Another text file specifies any rotations which should be applied to certain blocks. The game has six main types of block, including ramps, spikes, and falling blocks.
    b. **CrackedBlock, FinishBlock, and SpikesScript** are scripts which are attached to the prefabs of those blocks, giving them behaviour. For example, if the player touches a spike block, the level is restarted.

2. Myo control (using the Myo Unity sample as a starting point), all attached to children of the *Myo* GameObject:
    a. **MyoOrientation** provides a way of getting the positioning of the Myo armband in 3D space using the *GetMyoRotation()* method.
    b. **MyoPose** allows poses to be checked in a single function call, such as *myoPose.ConsumeFistIfDetected().* The pose is "consumed", preventing it from being registered a second time by another script (since poses can be held for more than one frame).
    c. **MyoGUI** is a simple text overlay showing the player the status of the Myo armband, I.e. if there is one detected, if it needs to be synced, etc. This also shows the game's Myo controls when the Myo is ready to use.
3. Level, player, and camera control:
    a. **LevelController** is attached to the *Level*, and allows the level to be rotated using the Myo, or the keyboard controls if the Myo is not available.
    b. **ThirdPersonCamera** is attached to the *Main Camera*, and makes the camera follow the player around. It also allows the camera to be rotated through the keyboard or Myo controls, so that the player gets the best angle possible during gameplay.
    c. **AngleUtils** is a utility class used by the above two scripts. It separates out some of the more complicated Quaternion/Vector code, such as limiting the rotation of a Vector along two axis, and rotating a vector around another vector.
    d. **PlayerController** is attached to the *Player*, and implements the jumping and dashing abilities used by either the Myo or keyboard controls.
4. Menu and scene changes:
    a. **MainMenu** and **YouWin** are simple scripts used to add functionality to menus within the game. They are used in the *MainMenu* and *YouWin* scenes, respectively.
5. Sound:
    a. **AudioManager** and **Sound** are used to play audio in the game, including music and various sound effects. They are accessed using the *AudioManager* found in each scene.

Conclusions and Recommendations

Tommey: Even though our idea was simple, we still learned a lot in relation to using hardware and the possible difficulties that come with it. We had initially given the game keyboard controls so we could get a start on testing the game before we got the Myo armband. It helped a lot in our case since the COVID-19 pandemic shut down all colleges. We had a backup for testing the in-game features. We were lucky enough since Ronan had gotten a Myo armband, but that meant I couldn't test with the Myo. It resulted in more communication to ensure there were no issues with any of my commits.

Developing this in Unity made it much easier than we had initially thought so I feel we made the right choice in choosing to develop it using Unity. Our previous experience using Unity also helped us when coding and organising files. It was interesting having both the keyboard controls and the Myo controls for the game. Even though the different controls would have the same feature output, the code had to be written in a completely different way in some cases.

If I were to recommend anything, one thing I think we could have done better is integrating the Myo's in sooner and providing a better user interface. Using the Myo's earlier might have allowed us to get more features implemented using them. Although I am happy with the work we did get done with them. I feel that if we had more time, we could have done better in terms of the user interface. We purposefully prioritised the functionality of the game with the Myo's as that was the main goal of this project.

Ronan: I enjoyed working with unique hardware, and exploring aspects of Unity which I hadn't done on previous projects. This included manipulating Quaternions and Vectors in complicated ways, in order to give the player the control and perspective of the game that we wanted. Another example is generating block based levels in 3D space, based on a design specified in a text file. After tackling those areas of the project, I am much more prepared for future projects in Unity, or 3D projects in general. Working with the Myo has also shown me the potential benefits of using specialized hardware as an input device for a game or other application.

As for recommendations, I think Tommey and I tackled this project very well. We made plans for the game's design early on, before starting development, and managed to find our own areas to work on without causing issues. Overall, we achieved what we wanted to achieve with the project, and it works about as well as we expected it to. One thing we could have done better was incorporating the hardware into our project at an earlier stage than we did, though. This would give us more time to further integrate the Myo into our project, such as being able to use the Myo to navigate the in-game menus.