**Kwii-Mote NEA Project**

**By Kweku Ackom-Mensah**

|  |  |
| --- | --- |
| **analysis** | **1** |
| **design** | **18** |
| **Technical solution** |  |
|  |  |
|  |  |

**Contents**

* 1. **Analysis**

**1.1**

For my Nea I will be creating a series of games which will be controlled various methods of motion capture.

The project would be an investigation into motion capture and the different methods in which you can implement it into video games as well as the different methods used to obtain the raw data that is used in the video games (IMU, LED tracking etc.).

*(playstation move controller)**(nintendo wii controller)*

***IMU(inertial movement units)***

An IMU is a unit that uses both a gyroscope and an accelerometer to measure linear movement and rotation. It does this by getting the raw data from the accelerometer and integrating it twice (from acceleration to velocity, then from velocity to displacement) in order to find the distance that it has travelled. To measure rotation, the unit takes all the raw data from the gyroscope in a certain period of time and sums it all up creating a value for the rotation of the unit.

***LED Tracking***

LED tracking works by using a camera to identify led lights within an image. The image will then be split into a bit map or grid and the location of the lights will be assigned a value based on where it is within the image. In order to track movement within the user you would look for changes within the location of the LED within the cameras image.

Supervisor

My supervisor will be my computer science teacher, I will have regular meetings with him to discuss and review the progress of my project.

Target Audience

The target audience for my project will be children and teenagers as these were the age groups that the wii and playstation move, as well as video games in general, are most popular amongst so I believe that they would be able to give me the best feedback and response to my project.

When the project is completed, I will test it amongst both young children, 7-12 and teenagers, 13-17. I will also test my project with users using both motion capture and then replaying using a keyboard or a controller to see if there is a noticeable change in the gameplay and/or enjoyment of the game.

**1.2 - Investigation**

Inspiration for my project

***The Wii***

The Nintendo Wii was released in 2006 and quickly grew popular amongst children and teens as it gave them access to a new way to play games which had already been done before and make them feel completely different through the use of motion capture. It also implemented local multiplayer function which allowed users to play with their friends cooperatively or against each other. It used both an IMU and an infrared camera that tracked to location of a two LED lights that stayed in the same location in order to assign itself a position in relation to the LEDs.

|  |  |
| --- | --- |
| Pros | Cons |
| * Very accurate tracking of movements through IMU and data smoothing algorithms * Easy to use UI * Large selection of games | * Problematic LED tracking due to the small field of view of the infrared camera within the controller |

***Wii Motion Plus***

Wii Motion Plus was released three years after the Nintendo Wii and was an add on to the Wii mote controller. It housed extra accelerometers and gyroscopes which could be used in tandem with the Wii mote. This meant that the Wii mote could capture and measure player movement more accurately allowing the user to have a smoother gameplay experience.

|  |  |
| --- | --- |
| Pros | Cons |
| * Even more accurate movement and orientation tracking due to the extra hardware * Even larger selection of games | * LED tracking was still inaccurate at times due to the field of view of the infrared camera within the controller |

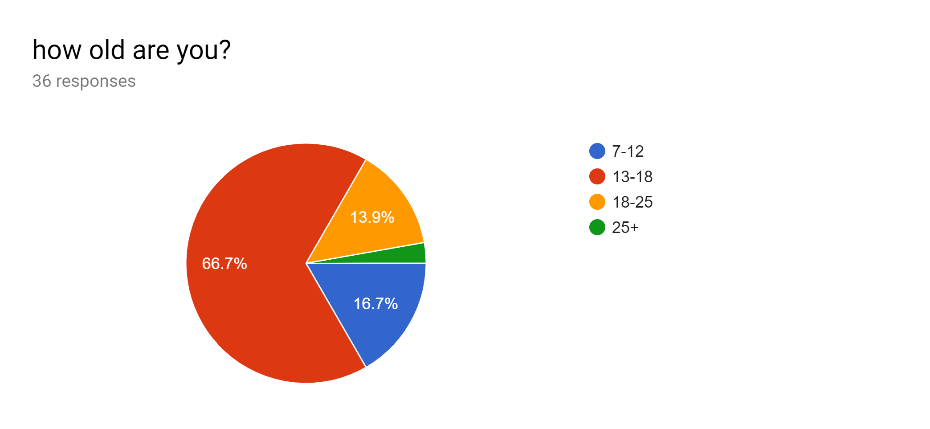
***PlayStation Move***

The PlayStation move was released in 2010 and was in a way Sony’s response to the Nintendo Wii. It was similar to the Wii in the way that it used a gyroscope and accelerometer to measure movement and rotation however it sported a large orb at the top of the controller that housed a set of RGB LED lights which could be identified and tracked by an external camera that would identify each led light in the frame and would give it a virtual location based on the location light in the camera frame. The orb on the controller was also quite large making it difficult for its view of the camera to be obstructed by other objects. As well as this, the lights could also change colour within the orb and this could be used to easily identify different players by both the camera and the users

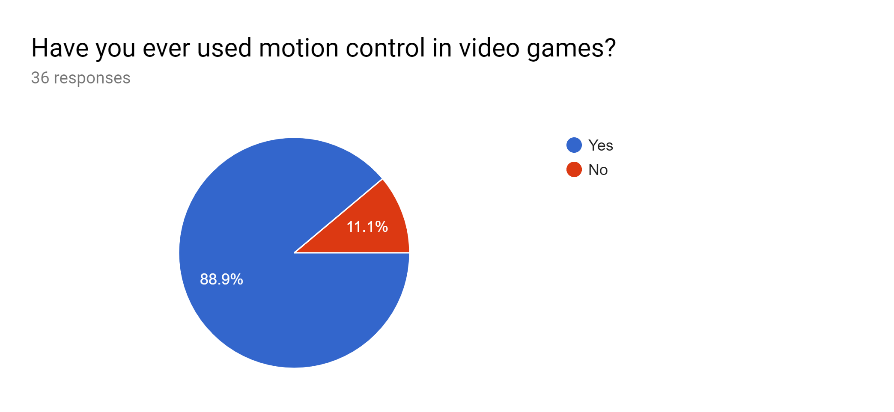
|  |  |
| --- | --- |
| Pros | Cons |
| * Very accurate tracking of movements through LED tracking as the light on the controller was quite large making it easy to locate * Large orb making the controller easy to track | * Fragile design (joint between the orb and controller could easily break). This would be especially problematic as it would be made for younger children who may not always be the most careful * Small selection of games |

After reviewing these consoles and the pros and cons of each I have decided that I will use the orb design of the PlayStation move and the RGB lights to distinguish between different players. However, I will make sure that the orb is secure and sturdy on the controller. I will also use the design of the Wii motion plus in the sense that I will use multiple accelerometers and gyroscopes and take an average of all the data I order to get more accurate readings of the raw data.

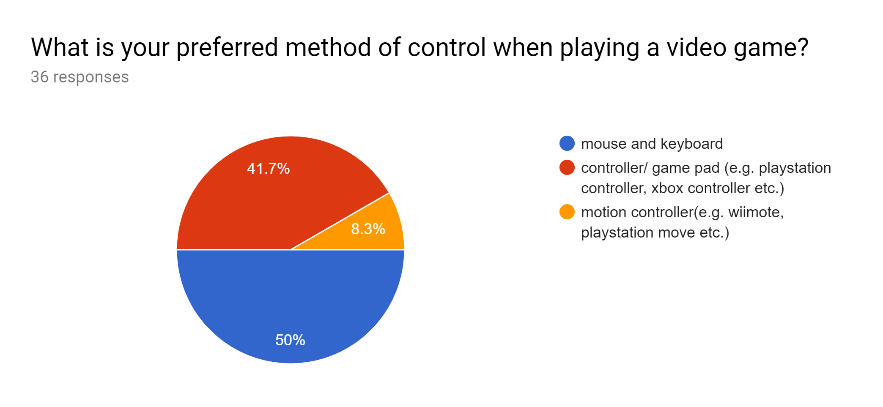
My Questionnaire



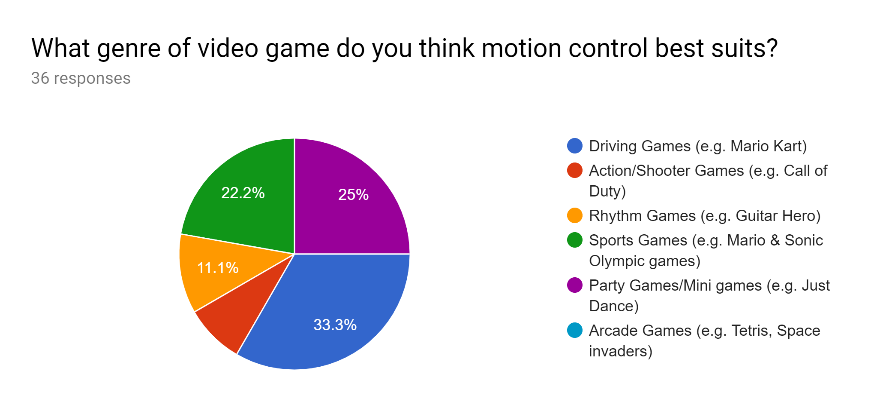
As most of the users will be either teenagers or children, the questionnaire was answered by these ages (with most of the 18-25 responses being people who are aged 18).



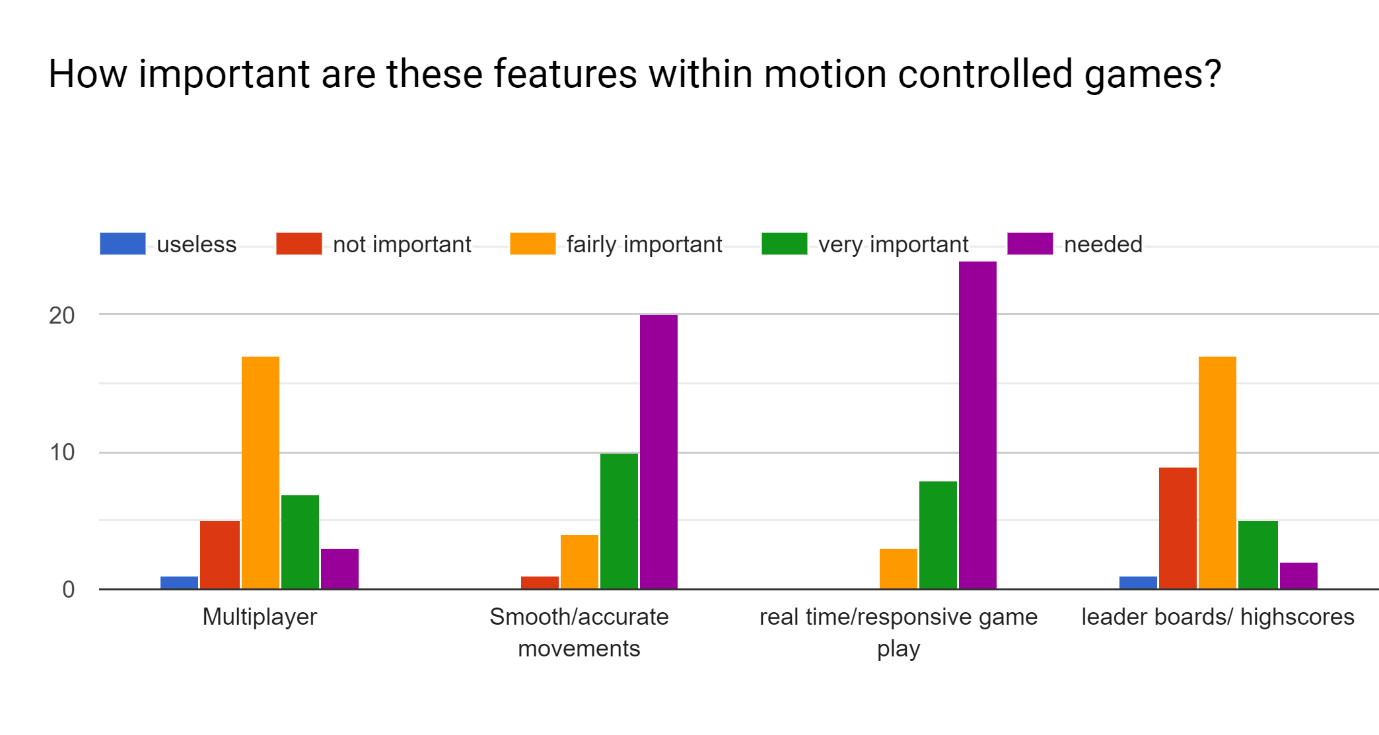
I then asked whether they had used motion control in video games before, to see if they were familiar with how it would and the different ways it could be integrated into a game. most of the responses were yes however 4 people responded with a no, meaning that they may not be as familiar with motion control in video games and/or how to use it. As a result of these responses, I will make sure that my project is easy to use and that there are clear instructions to for the user.



The users were then asked what their preferred method of control is when playing a video game. only 3 out of the 36 responded with motion control and there was a plethora of reasons for this, however the recurring answer was that it was not accurate enough to be used competitively (especially in action/shooter games). As a result of this, I will be making sure that the data that I collect for the motion capture is as accurate as possible. To make sure the data is accurate and smooth I will work with a bigger bitmap grid and a higher pixel density so that the LED can be assigned a more precise location. I will also be using multiple gyroscopes and accelerometers within the IMU and take an average of all the values. I with then use this data to navigate and control through the video game as if it was the raw data.



I then asked the users what genre of game they thought would be best suit a motion-controlled game. The majority of users said that a driving game similar in style to Mario kart would be best suited for a project like mine. Sports & party games where also shown to be popular amongst the different genres of a game.

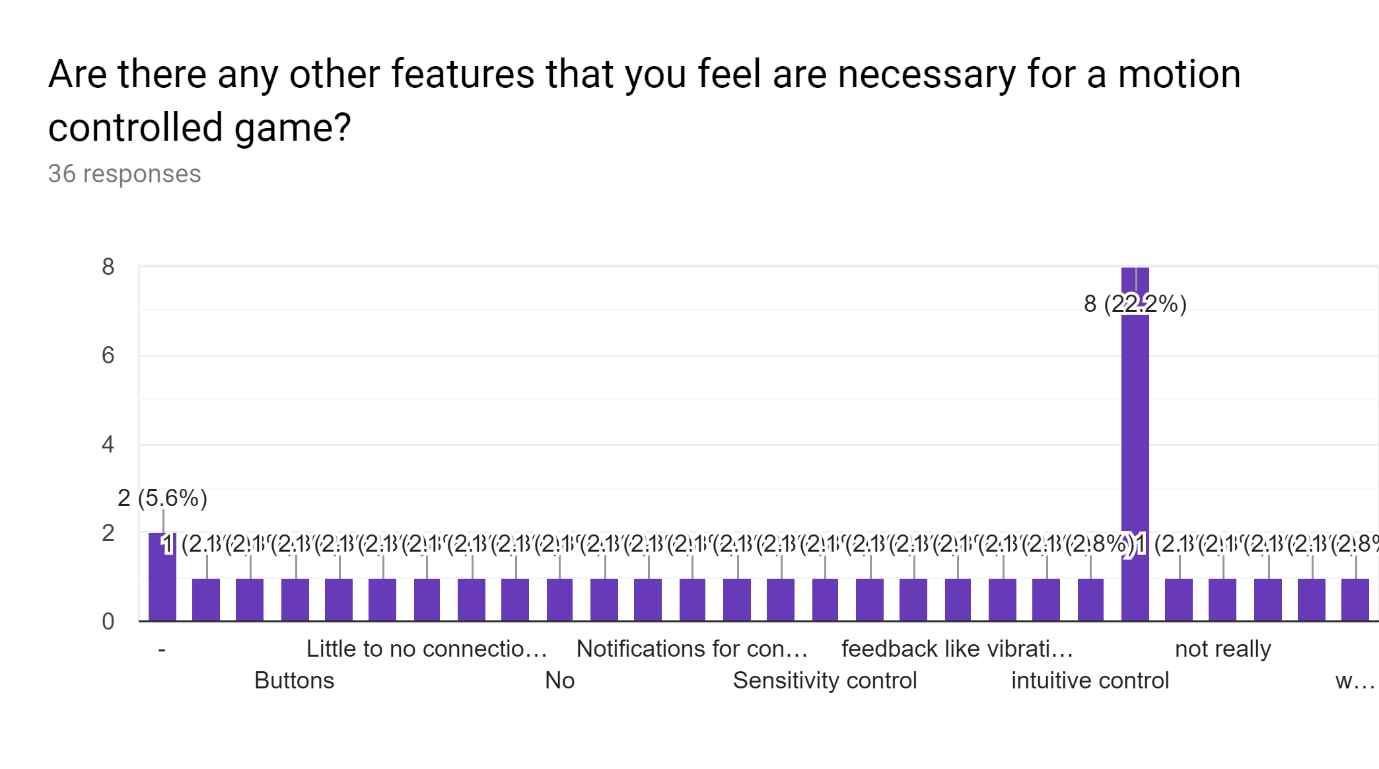


I then gave the users a choice of replies they could respond with based on how important they felt a certain gameplay feature was.

The first feature mentioned was the option to have a local multiplayer mode that would allow two or more users to play at once either against each other or as a team. On the survey multiplayer was not the most popular feature listed, with it only receiving 7 votes for very important and 3 for needed. However, I will still be implementing an option for local multiplayer especially in the party/mini game. To help me with this, I will be using a design similar to the playstation move in which I will use different coloured LED lights in order to distinguish between users.

Smooth and accurate movements where amongst the most popular features in the chart, with nearly 2/3 of the responses saying that it was needed to create a good/enjoyable user experience. So, I will be using multiple gyro scopes and accelerometer in or the get more accurate data. I will also be using the LED motion capture in tandem with the IMU to calculate movement or speed to a more accurate degree.

Real time responsive gameplay was the most popular feature amongst the users, with 24 people voting that it was need in motion control video games. When asked why they chose needed as a response one person said that ‘if there was a big lag between the input and what was happening on screen it would make the game less fun and enjoyable to play as you don’t feel like you are actually even plating’. I agree with this statement and think that real time gameplay and quick responsive gameplay should be a priority when creating these games.

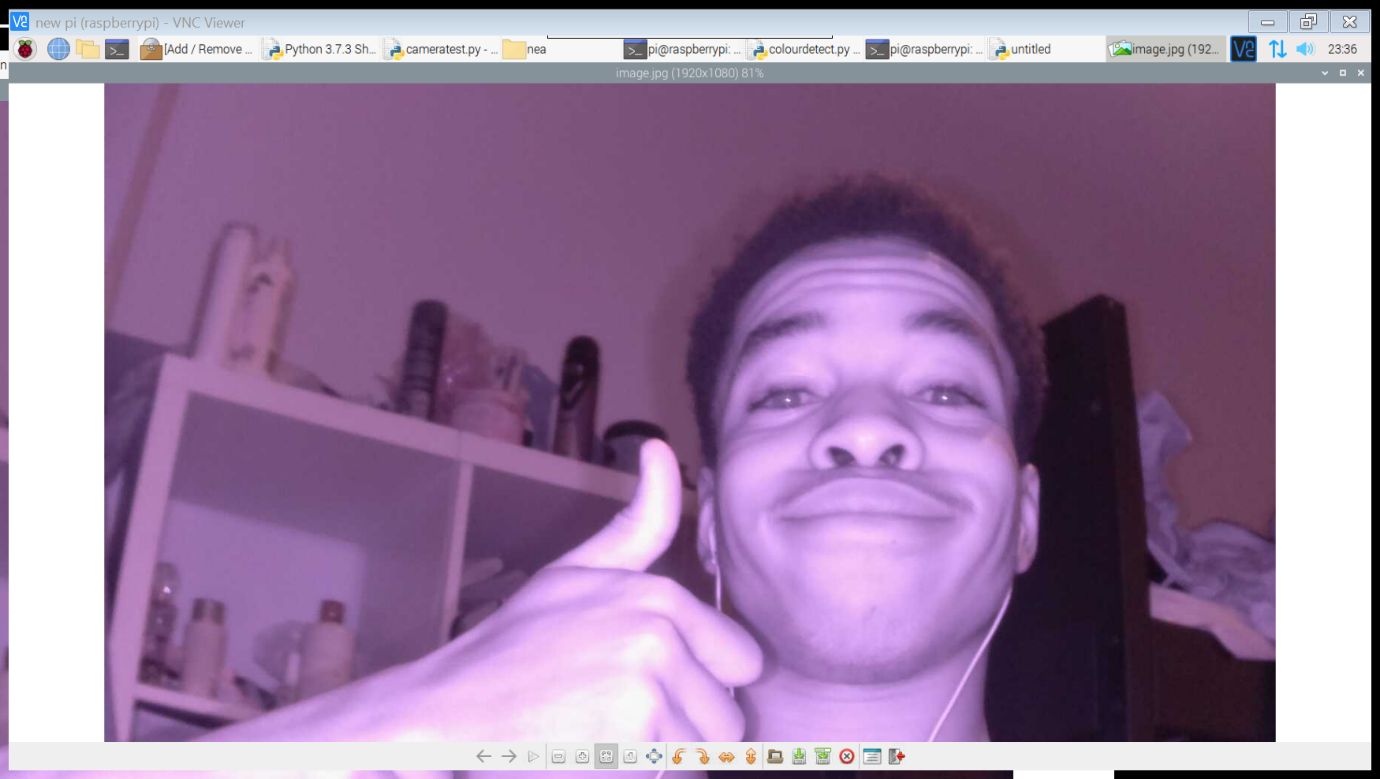
Leader boards and high scores were one of the least popular features on the list and while I will still be attempting to implement some form of basic leader boards or high scores in the future. 

At the end of the survey I asked if there was any feature that could be added to the project to enhance the gameplay experience. There were a number of different responses, some of the main/recurring responses were to have notifications based on when a controller either connects of disconnects and an option to adjust the sensitivity of the device. I will be adding both of these features to my project as the both make the make it easier for all users to play.

There was also a request for there to be a way for users to be able to easily distinguish between which controller is player 1 or player 2. In response to this I will be changing the colours of the LEDs that are attached to the controller dependent on which user is playing e.g. the LED of the controller for player 1 will be red and the LED of the controller for player 2 will be blue.

Someone also asked for me to include customisable characters in the games. So, I will be giving the option for users to use an image of their face that would be taken by the character instead of the default game characters. Users will also be able to earn alternative avatars that they will be able to use in the games (these avatars will most likely just be images and gifs from google that I find funny). Users will also have the option to create an account to which all their avatars and point would be saved. The usernames and passwords and the avatars and points associated with each account for each account would be saved to a csv file.

There was also a request for high scores to be saved so I am going to implement the option for a user to save their highest scores to a leader board that will automatically be sorted from high to low by using a bubble sort function that I will create.

*(image of user taken with infrared camera)*

Another feature that was requested was the option to 1v1 or play against another player competitively in the driving game. I will be attempting to implement this feature into the driving game alongside with the feature for special items/ avatars to be earnt from winning enough 1v1s.

There was also a request for the user to be able to choose between a 1 handed or 2 handed mode in the dancing game. So, I will be implementing this feature as well as a points multiplier based on which game mode you choose.

Interview

After reading the results from the survey I decided to conduct an interview with my girlfriend’s little sister, May, who is 10. I did this in order to get a better understanding of what would be needed in my project in order to make it as enjoyable as possible.

The first question I asked was what age she was how old she was as I wanted to interview someone under the age of 13. Because this is the age group that I want my project to appeal to the most.

I then asked, ‘whether or not she had used motion control before?’, to which she said ‘yes’. The next question I asked was ‘what her favourite method of control was when playing a video game’ to which she replied, ‘a game controller because it easier to control things’. As a result of this answer I am going to make sure that my project is easy to use by all ages.

The next question I asked was how hard it was to use motion control the first time she used it in a video game. her response was that it was ‘pretty easy’. This would probably due to the clear instructions and intuitive design of the Wii (this was the first motion-controlled gaming device that she used).

I then asked how often she used motion control in videos, to which she responded, ‘about a few times a month’, when I asked why her response was that it was ‘only fun when playing games against your friends or when you can play against them’.

I also asked if she had eve used motion control outside of video games and if so where and she said that she had used it when ‘doing a vr tour of famous landmarks.’ As a follow up question to this I asked if there was anything in specific that she liked about that. Her response was that ‘it was easy to use’.

After conducting the interview, the general message that I received was that motion control is most enjoyable when it is easy to use and when using it with others. As a result of this I will strive to make my project easily usable to all age groups especially children as they are the target audience. I will also attempt to implement local multiplayer functionality so that the users will be able to interact with their friends.

project decision.

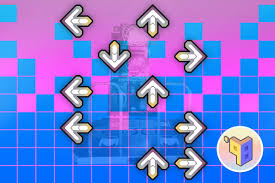
After researching and receiving all this knowledge about motion control in video games I have decided to split my project into three sperate games that would be accessed through a menu/UI that would be controlled by a motion controller.

The first game that I will be creating would be a driving game (similar to *Super Mario Kart)*. This would be to demonstrate the use of IMU and how I can use it to measure orientation to control the steering of a character’s car within a driving game (tilting left and right to steer). Points would be allocated to the user based on the time taken to complete the track and the position that they finish in. I will also create a dancing game, similar to *just dance or dance dance revolution*. The game would require the user the controller in a certain direction, shown on the screen, in time with a music track. I will be making this game to demonstrate how I can track LED lights on a camera in order to track a user’s movements within a certain amount of space. The final game that I create will be a baseball game which would use LED tracking to identify where the controller is and use this information to draw a baseball bat on the screen at a location based on the position of the controller. The game would require the user to move the controller to the correct location in order to hit the ball back. The user would also be required to swing the controller at a specific time in order to return the ball on the game. The distance that the ball travels would be based on how close the users timing of the swing is to the time that they were supposed to swing and the speed that they swing the controller. I would measure the swing by using the IMU to measure sudden changes in movement and determine whether or not the controller has been swung or not.

If there is time at the end of the project, I will recreate the driving game for iOS devices using swift and I will be using the inbuilt gyroscope and accelerometer in the device to receive the raw data for the orientation which will again be used to control the steering of the driver in the game.

The will also be a series of random small games e.g. a running game which requires the user to shake the motion control unit as fast as they can in order to make the on-screen character run or a rhythm game that would require the user to swing the unit in time with a song that is being played.

*Super Mario Kart.   Wii sports: Baseball*

*Dance dance revolution.                                                        Source: google images*

**Prototyping and how I will be making the game.**

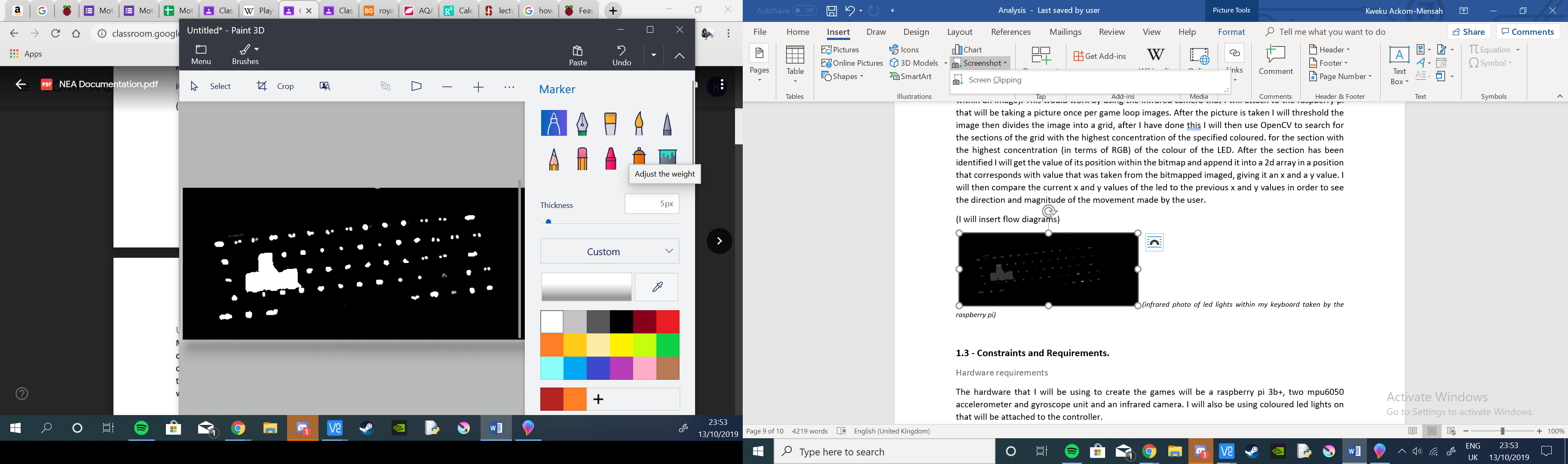
Motion tracking

Over the summer I created some programs in python and pygame in order to test the led tracking and the IMU unit and familiarise myself with them.

For the IMU I used an mpu6050 module and attached it to a raspberry pi through GPIO pins and then using an i2c connection to allow communication between the pi and the module. I then created a program that detected linear movement within from the accelerometer and moved an image around a blank screen according to the direction that the unit was moving.

For the led tracking I used attached an infrared camera to a raspberry pi to take a picture of the led lights within my keyboard. Using OpenCV, I then converted into a binary image based on whether or not the was a LED light present. After doing this I then bitmapped the binary image and took the values of the sections with the LEDs present.

*(photo of the led lights from my keyboard)*

 *(binary photo of the led lights created using OpenCV)*

The games

For the driving game I will be using pygames draw function to draw the image of the driver in the middle of the screen, I will then redraw the track and other players to make it seem as if they are moving backwards in order to simulate the player moving forward. I will append the values of the time when the race starts and the time that they cross the finish line and the characters that passed the finish line in order of the when they finished, so I can work out how long each character took to complete the race and the position that they came in. At the end of the race the use will be allocated points based on the position that they came in and the time that they took to complete the race.

(insert data flow diagram for driving game)

As for the dance game that will be done by drawing arrows to travel up the screen and when the reach the they reach the top of the screen the time will be noted and the direction that the arrow was pointing. As this is happening the user’s movements will be also be monitored and each time that there is a significant change in the position of the controller the time and the direction of the movement will be noted. These values will then be compared to the time and direction of the arrow reaching the top and depending on how close the times are and whether it is the same direction the user will be allocated points.

(insert data flow diagram for the dance game)

Finally, the baseball game will also use pygame’s draw function to redraw the position of the bat according to the position of the controller in the frame of the camera. I will then draw a moving ball coming towards the player and take a note of the time when it reaches the earliest place that the player can hit it from. I will again be monitoring the movement of the unit however this time I will be using the IMU to determine whether the is a significant change in the movement of the unit. If there is, I will append the values of the location of the unit from the led tracking system and the time of the swing and compare it to the location of the ball within the game and time to see whether the ball has been hit. If the ball has been hit, I will then use the orientation of unit and the speed of the swing in order to determine how far the ball will travel and whether it will curve while it is in the air. The user will be allocated points based on how far they can hit the ball.

(insert data flow diagram for the baseball game)

All the points that the users achieve from each game will then be append to a list of personal high scores and global high scores dependent on how many points they have. The user will be able to view these high scores and they will be automatically updated and sorted as each.

While all these games are running, I will be constantly checking to see if there is a sudden stop in the data from wither the IMU or the LED tracking system and If there is the game loop would halt and the user would be notified that the controller of disconnected.

(insert data flow diagram for the main menu/ user interface)

How the motion tracking will work.

For the IMU tracking I will be using two mpu6050 modules which use an accelerometer and gyroscope to measure in order to measure linear acceleration, rotation and tilt. To measure the acceleration, I will take an average of the values that are recorded by the two units and append them into list. As the list is being made, I will divide the values of the averages into sets of 3’s. I will then take a sum of the 3 values to see if there is a positive or a negative change in the accelerometer values, this will allow me to see the direction in which the unit has moved. I will be making the values into sets of 3 in order to counteract the negative values that are given at the end of user’s movement as a result of unit being stopped. After I take each sum, I will append the sums into lists of 5s in order to determine the magnitude of the movement that the user has made. I will again be doing this by taking sums of the lists. As for the tilt and rotation I will be again taking averages of the data and making them into sets of 3s. Then I will be taking a sum of the sets of data each game loop and adding it to variable called orientation that will dictate the orientation and tilt of the unit. I will also be implementing an orientation reset that will allow the user to reset the orientation of the module back to 0, this is encase there is a significant offset in the data.

(I will insert screenshots and data flow diagrams of the imu data in the console here)

As for the led tracking, I will be using an infrared camera (as it will allow me to block out all colour other than the led lights) and a library called OpenCV (that would allow me to detect specific colour within an image). This would work by using the infrared camera that I will attach to the raspberry pi that will be taking a picture once per game loop images. After the picture is taken I will threshold the image, convert it into a binary image and then divide it into a bit-mapped grid. After I have done this, I will then use OpenCV to search for the sections of the grid with the highest concentration of positive values within the binary image. for the section with the highest concentration (in terms of RGB) of the colour of the LED. After the section has been identified I will get the value of its position within the bitmap and append it into a 2d array in a position that corresponds with value that was taken from the bitmapped imaged, giving it an x and a y value. I will then compare the current x and y values of the led to the previous x and y values in order to see the direction and magnitude of the movement made by the user.

(I will insert flow diagrams)

Project Scope / To do list

I am aiming to finish my project within a few months. So, I have made a table in order to track my projects progress and how long each task will take in order to help to keep me on track (I will be filling it out as I go through the project)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Task | Duration(in days) | Due date |
| 1 | **Analysis** | **13** | 14th October |
| 1.1 | Introduction | 1 |  |
| 1.2 | Investigation | 8 |  |
| 1.3 | Constraints, requirements and limitations | 2 |  |
| 1.4 | Objectives | 1 |  |
| 1.5 | Proposed Solution | 1 |  |
|  |  |  |  |
| 2 | **Documented design** |  | 12th November |
| 2.1 |  |  |  |
|  |  |  |  |

**1.3 - Constraints and Requirements.**

Hardware requirements

The hardware that I will be using to create the games will be a raspberry pi 3b+, two mpu6050 accelerometer and gyroscope unit and an infrared camera. I will also be using coloured led lights on that will be attached to the controller.

Other than that, my project should be able to be run on most computers as it not a hardware straining program.

As for other devices such as phones and tablets (if I get around to it), it will again be a relatively accessible program and will probably be available to use on any device running IOS 12 or higher.

Software requirements

The only software that my project will require is a working version of Raspbian with OpenCV already on it, these are both free programs that are available for anyone to download and use.

As for the mobile driving game the only software that will be required is IOS 12 or higher, this is supported by most iPhone/iPad devices.

In order to create my project, I will be using Raspbian and python (with pygame) to program the led tracking and IMU along with the baseball game, dance game and driving game. However, for the mobile version of the driving game I will be using XCode with swift to create the program.

User’s Knowledge of information technology

As I will be testing my game amongst children and teenagers, I will assume that the users will have a fair understanding of technology and how to use it. However, there will be some people who have had no previous experience with motion-controlled games before as shown in the survey. Therefore, I am aiming to make my project as easily accessible as possible. I will be doing this by making the controls and gameplay intuitive meaning that the user will be able to understand and figure out what is happening within the game and how to play without really having to look at the instructions or controls. I will also think carefully about the design of the UI and placement of the buttons, so it is easy to navigate through using the motion controller. I will also include a ‘help me’ page that will include the instructions of the games on how to use the motion controller in the games/UI.

Access rights

All of the users will have the same permissions and will be able to access and use all the features in the program other that avatars or character items that they have not earnt yet. However, I may add the option for special mini games that would only be available to the top 3 people on the leader board e.g. a smaller additional rhythm game as this was the next most popular game genre according to the survey.

**1.4 – Limitations**

Areas not touched / areas to consider for the future

There are some features that I could add in the future such as online multiplayer or even multiplayer between two devices that are connected to the same network. I will not be doing this now as I do not have access to enough hardware and I do not yet have enough knowledge about networking to be able to connect multiple devices to the same game without it having a significant negative effect on the gameplay/responsiveness of the game. So, I will only be implementing the option for single and local multiplayer on the same device for now.

I could also further develop the games and add more features to each game for example adding power ups to the driving game such as a speed boost or projectiles that you can shoot at other racers in order to slow them down. There are also other features like being able to upload and drive against a ghost of your previous or best lap/race. I will not be implementing this feature now as it would require me to save a recording of the race/lap and save it to a players account then replay it as the player is doing the race and I do not think I have enough time or the knowledge to create an efficient method of storing all the ghost replays. In the dancing game there are features like being able to upload a custom song and have the program automatically create a routine for the song in time with the beat of the music. I will not be implementing this feature as it would require a further understanding of reading from mp3 and mp4 files. There is also a multitude of other features that I can include in the baseball game. However, I will not be including them in my project at the moment as they would not provide much benefit to the motion-controlled aspects of the gameplay so I would rather be focusing on the more essential aspects of each game at the moment.

I could also make my project available to more devices, especially ones with in built gyroscopes and accelerometers. However, this would require me to have further knowledge on additional programming languages that I do not already know, such as java for some android devices.

**1.5 – Objectives**

General objectives

My general objectives involve making sure that the user is able to play the games and navigate through the UI / the main menu.

* The main program and games all function as planned; the menu loads and allows the user to choose whether they want to go play one of the games, view their account or navigate to the leader boards.
* When a new user creates an account the credentials (and picture if they choose to take one is saved to a csv file that can be accessed later
* Users can view and edit the leader boards (by achieving a high score).
* The motion control units will be able to track its movement to an accurate enough degree that the user is able to play and interact with my project with ease.
* The infrared camera is able to identify the controllers and allocate them a locational value based on their location.
* The user is notified if one of the units gets disconnected or stops working.
* The user is able to recalibrate the orientation and/or the position of the unit

Specific objectives

When the user opens the program, they should be taken to a menu that they can use the motion controller to navigate through.

* The main menu options should be:
  + Login or play as guest
    - Select a game
      * Driving game
        + Single player

Time trial

Race vs CDPU

* + - * + Local multiplayer

1v1

Race vs user & CPU

* + - * Dance game
        + Single player

1 handed mode

2 handed mode

* + - * + Multiplayer

1 handed mode

2 handed mode

* + - * Baseball game (if the motion control unit gets disconnected from the computer at any point during any of the games the game should pause automatically)
        + Single player
        + Multiplayer
      * Minigame (only accessible by top 3 users on the global all-round leader board, a random mini game is chosen)
        + Single player
        + multiplayer
    - View account (only available for login users)
      * Change username
      * Change avatar or cosmetic
      * Purchase avatar or cosmetic
        + Random choice
        + Specific purchase
    - View leader boards (only available for login users)
      * Personal high scores
        + Driving game
        + Dance game
        + Baseball game
      * Global high scores
        + Driving game
        + Dance game
        + Baseball game
    - Help
      * Recalibrate
        + Recalibrate controller position
        + Recalibrate controller orientation
      * Game instructions
        + Driving game
        + Dance game
        + Baseball game
      * Controls
  + Create an account (users’ credentials will be saved to a csv file)
  + Help
    - Recalibrate
      * Recalibrate controller position
      * Recalibrate controller orientation
    - Game instructions
      * Driving game
      * Dance game
      * Baseball game
    - Controls
  + Exit

Additional objectives

* Users will have the ability to play as guest

**1.6 – Proposed solution**

Conclusion

The hardware I will be using is are 2 mpu6050 units and an infrared camera. I will be using 2 modules in order to minimise the discrepancies within the data and I will be using an infrared camera as it will make it easier to block out anything that is not the LED.

I will be programming in pygame which is an extension of the programming language Python because I am already familiar with it and it is the default language for a Raspberry Pi (this would be beneficial to me as it will allow me to access the mpu6050 module and camera relatively easily). I will be using pygame’s draw function to create moving images within my game.

I will be using a library called OpenCV (<https://opencv.org/>). This library is mainly aimed at allowing people to use real-time camera inputs within code. I will be using it to identify the colours in of the LEDs on the controllers and then converting the image into a binary image with the coloured LED being identified as white and everything else in the image as black. I will then divide it into a bit-mapped grid. After I have done this, I will then use OpenCV to search for the sections of the grid with the highest concentration of positive values within the binary image. for the section with the highest concentration (in terms of RGB) of the colour of the LED. After the section has been identified I will get the value of its position within the bitmap and append it into a 2d array in a position that corresponds with value that was taken from the bitmapped imaged, giving it an x and a y value. I will then compare the current x and y values of the led to the previous x and y values in order to see the direction and magnitude of the movement made by the user.

As for the IMU tracking I will be using two mpu6050 modules which use an accelerometer and gyroscope to measure in order to measure linear acceleration, rotation and tilt. To measure the acceleration, I will take an average of the values that are recorded by the two units and append them into list. As the list is being made, I will divide the values of the averages into sets of 3’s. I will then take a sum of the 3 values to see if there is a positive or a negative change in the accelerometer values, this will allow me to see the direction in which the unit has moved. I will be making the values into sets of 3 in order to counteract the negative values that are given at the end of user’s movement as a result of unit being stopped. After I take each sum, I will append the sums into lists of 5s in order to determine the magnitude of the movement that the user has made. I will again be doing this by taking sums of the lists. As for the tilt and rotation I will be again taking averages of the data and making them into sets of 3s. Then I will be taking a sum of the sets of data each game loop and adding it to variable called orientation that will dictate the orientation and tilt of the unit.

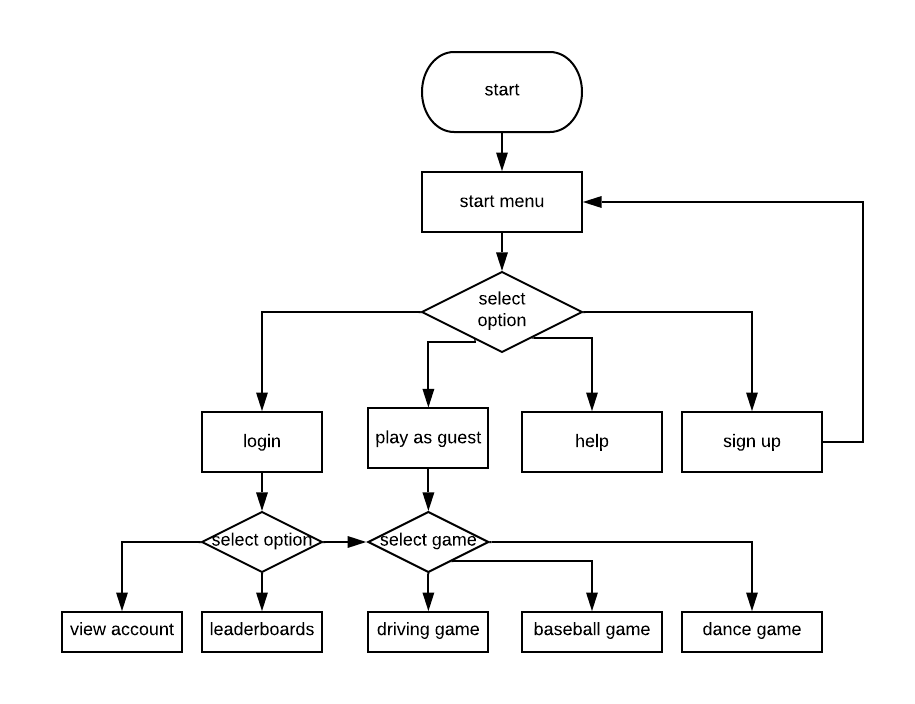
1. **Design**

**2.1 – High level overview**

The flowcharts below describe how my project will work. When the user starts the program, they will be sent to the start menu where there will be several options that the user can use the motion tracking to interact with.

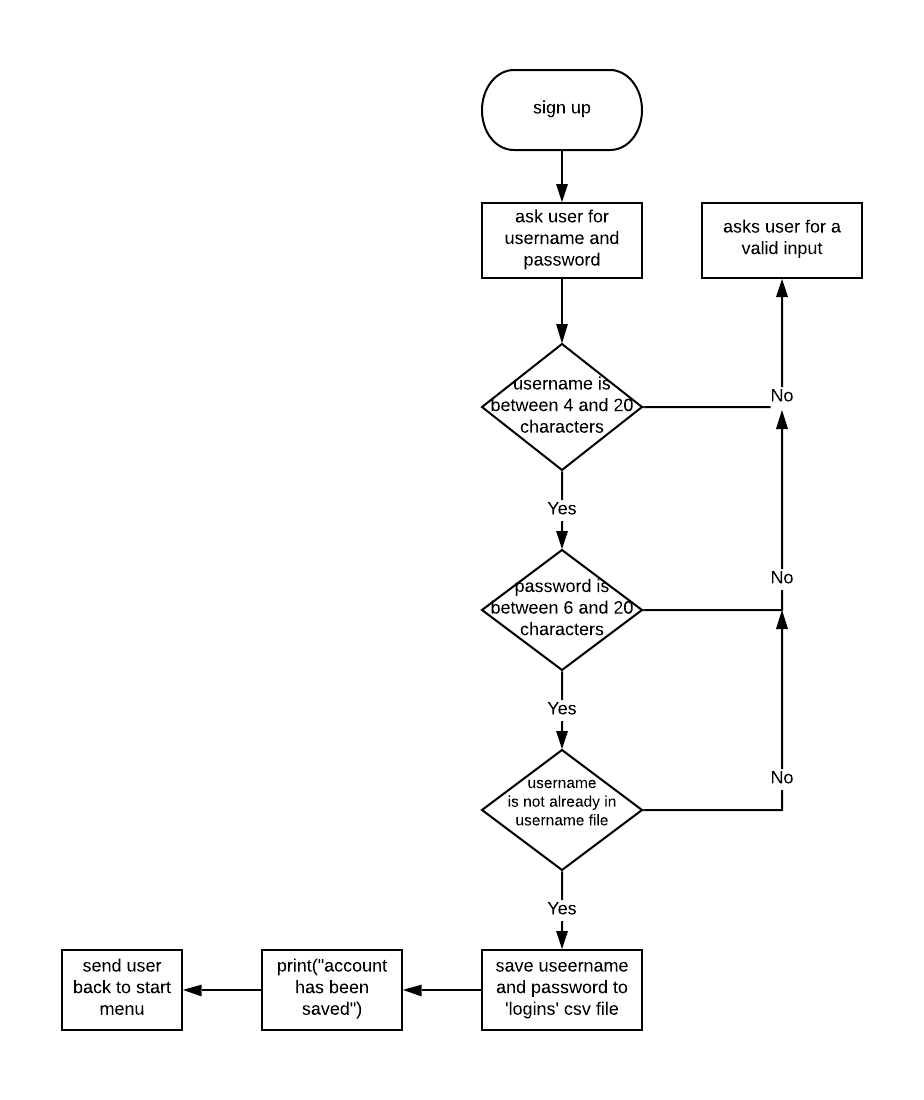
(with each step in the program there will always be a back button that would take the user to the page that they were on before, I will do this by appending the name of each page into a stack and popping the top value when the user wants to go back)

**Starting menu**



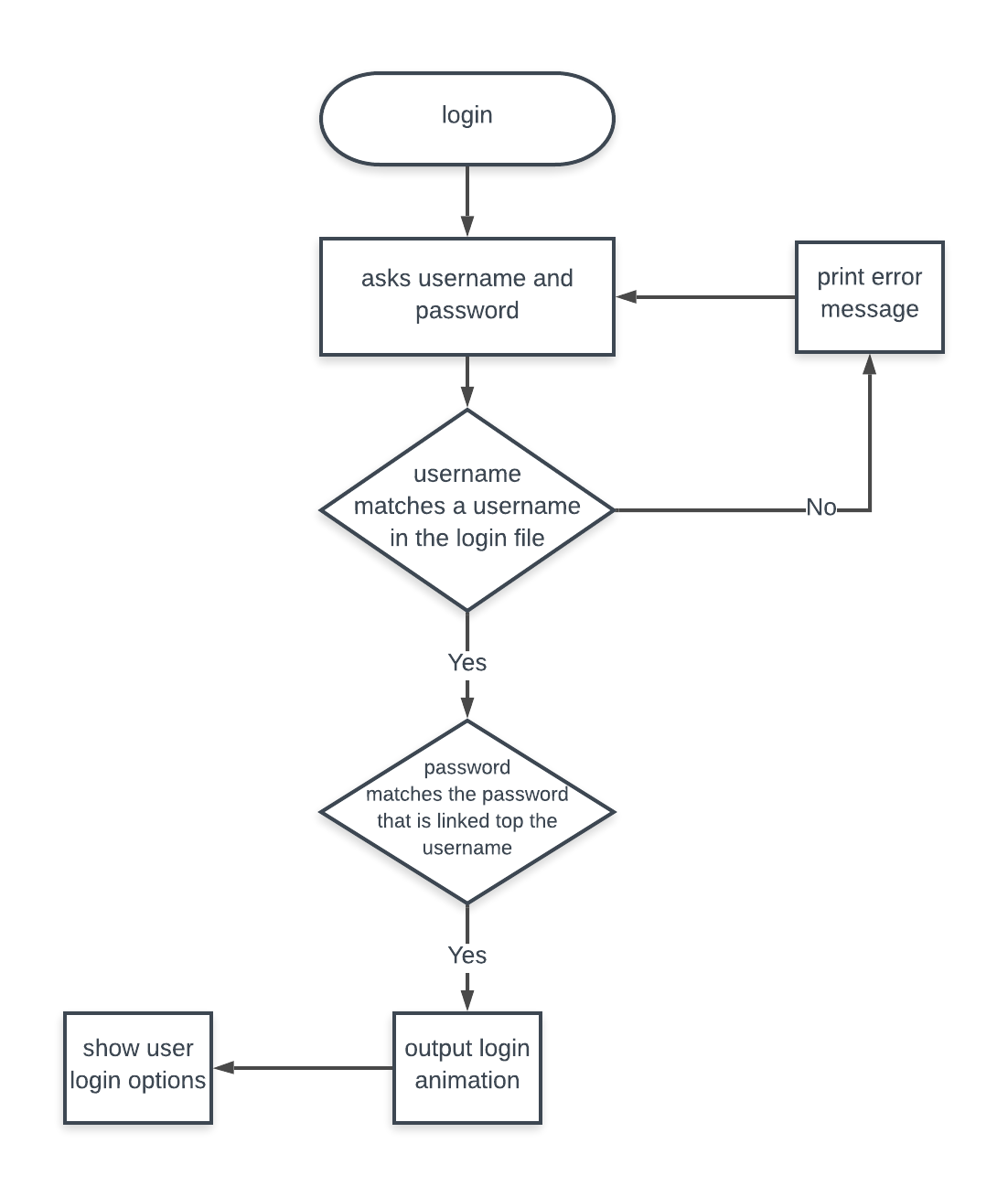
**Sign Up**

When the user selects ‘sign up’ they will be taken to a page where they can create their own account by entering a username and a password. These input will be checked to ensure that they are valid before they are saved to the login file.



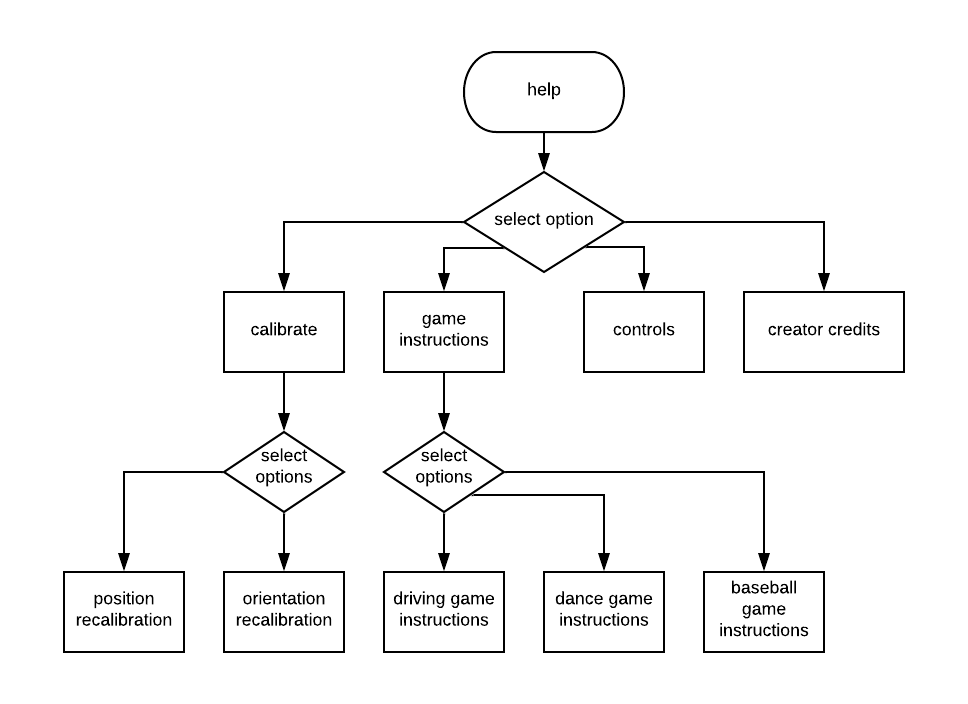
**Login**

When the user selects login, they will be prompted to enter their user name and password. These inputs will then be compared to the usernames and passwords in the ‘logins’ csv file to see whether they are valid.



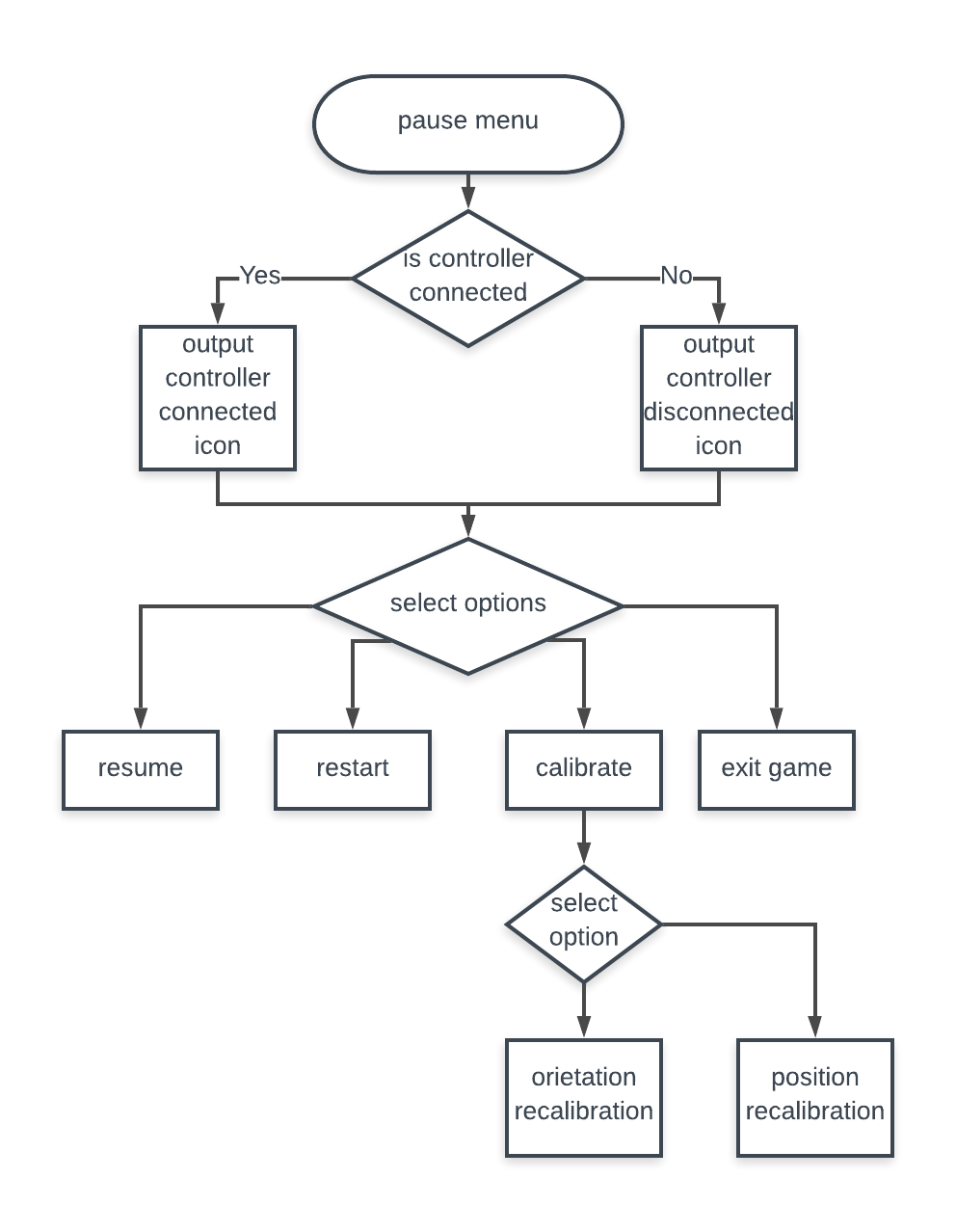
**Help**

When the user selects help, they will be taken to a page where they will be able to edit the settings and some of the user interface features of the program. They will also be able to view instructions of each game and recalibrate both the orientation and the position of the controller. There will also be an option to view creator credits of the game where all the people that where included as gif or skins for games will be mentioned.



**Pause Menu**

If the user is in a game and the motion controller is disconnected the user will be notified and the game will pause automatically and the pause menu will pop up. The pause menu will also be available to access manually mid game. It will give the user access to several in game features such as resume, restart, calibrate and exit. It will also show the user whether the controller has successfully reconnected.



**2.2 – Modular systems**

IPSO Table

Drawn below is an IPSO table for the game loops of my project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Inputs** | **Processes** | **Storage** | **Outputs** |
| - user input and raw data from the motion control unit  - user detail (username and password) | - collision detection of player and other objects (car and grass/curb in driving game etc.)  - processing and smooth raw data from the camera and the IMU unit in order to make it into usable inputs for the games  - calculating and sorting high scores within the game. | - csv files will be used as data bases in order to store most of the data in my project e.g. logins, skins and avatars and high scores. | - output player sprites to the screen  - output the game map or course to the screen  - sorted leader boards and high scores  - player skins and avatars that a player has achieved. |

**2.3 - Data Dictionary**

I will be storing most of the data in csv files and using them as data bases. The things that I will be storing in the csv files will be the logins, game scores and user skins and avatars

Login file

Every time that a user signs up on my program their login credentials will be saved to this file in order to be used to login

Example:

|  |  |
| --- | --- |
| **username** | **password** |
| Player 1 | PASsworD123 |
| KwekerOats | Qwerty123 |

Game scores

Each time that a registered user finishes a game they will be asked if they want to save their score. If they select yes, then their score will be saved to this file along with the game and the mode that they were playing.

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Game** | **Username** | **Score** | **Mode** |
| Driving | Player1 | 2345 | Time trial |
| Dance | KwekerOats | 3830 | One handed |
| Driving | Player1 | 4534 | Vs cpu |
| Baseball | KwekerOats | 3452 | Single player |

Character skins

Each time that a user is created a unique csv fil will also be created that contains the names of the png files that the user has access to as skins or avatars. It will also contain the type of cosmetic item that it is (either background skin, character skin or avatar)

Example:

|  |  |  |
| --- | --- | --- |
| **Image** | **Achieved Y/N** | **Type** |
| Kwekuface.png | Yes | Avatar |
| Johfaces.png | No | Dance game character skin |
| Trollface.png | Yes | Avatar |
| Champion.png | Yes | Driving game background skin |

**2.4 – Data structures**

the variables that will be used most often in each of the games are

driving game:

- racetime

- carposition

- trackspeed

- orientation

Dance game

- movedirection

- movetime

- userdirection

- usertime

- songspeed

Baseball game

- ballposition

- ballspeed

- ballhittime

- batposition

- batorientation

- batswing

I will be using object orientated programming to create most of the games. Some examples of the classes that will be made are:

- user\_car (in the driving game)

- player\_right\_hand (in the dancing game)

- user\_bat (in the baseball game)

I will be appending the name of the menu screens that the user goes to a stack, so I can allow them to go back to the last page by popping the most recent value.

**2.5 – Validation**

I will be validating the inputs that the user enters for the username and password to ensure that it would not impact my program negatively or cause it to stop working.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Validation check** | **Description** | **Error message** |
| Username | 4 <= length <= 20  Username is not the same as a previously saved username | Ensures that the username is between 4 and 20 characters long (so the whole name can be outputted in the leader board page if need)  Ensures that there are not two people that share the same name (so that there are no errors when trying to match the usernames and password) | Please choose a username between 4 and 20 characters long.  Usernames is already taken please choose a different one. |
| Password | 6 <= length <= 20 | Ensures that the password is a sensible length that the user will remember. | Please choose a password between 4 and 20 characters long |

…(documentation is not done, I will upload the rest by the end of the week)

Technical solution

Driving game:

The driving game is made by drawing a moving road and background and keeping the car in the middle to give the illusion that it is moving. The mpu6050 class is what allows the pi to communicate with the accelerometer and gyroscope. It works by first defining the ranges of the accelerometer and gyroscope and the registers that it will be using. It then defines the function that reads the raw values from the mpu6050 module via an i2c connection. It then takes to two values and combines them together. After this the functions to read either the gyroscope or the accelerometer is defined so that it can be used to get numerical values.

Smoothing the data

After the mpu6050 module and the pygame is set up an intial gyroscope value is taken. After this an average of the next 5 values are taken in order to ensure that there are no outliers and then the avrerage is appended to a list. This is done 3 times then another average of these 3 values are taken and then subtracted from the initial value in order to see the total distance that the module has turned/moved.

Drawing the road

The road is drawn by using 400 rectangles that are 1 pixel high. The location of these pixels are dependent on the direction that the road is turning, the magnitude of the turn and the y value of the point of the screen that it is located. This is because in order to create the illusion of a 3d road and perspective the road must slowly decrease in size and get further away from the middle of the screen during a turn as it gets further away as the field of view of the user would be increasing. As well as this the background of the game would have to be turning with the user as they turned in order to show that their direction was changing.

import pygame

from pygame.locals import \*

import os

import time

from mpu6050 import mpu6050

import math

import smbus

class mpu6050:

GRAVITIY\_MS2 = 9.80665

address = None

bus = None

ACCEL\_SCALE\_MODIFIER\_2G = 16384.0

ACCEL\_SCALE\_MODIFIER\_4G = 8192.0

ACCEL\_SCALE\_MODIFIER\_8G = 4096.0

ACCEL\_SCALE\_MODIFIER\_16G = 2048.0

GYRO\_SCALE\_MODIFIER\_250DEG = 131.0

GYRO\_SCALE\_MODIFIER\_500DEG = 65.5

GYRO\_SCALE\_MODIFIER\_1000DEG = 32.8

GYRO\_SCALE\_MODIFIER\_2000DEG = 16.4

ACCEL\_RANGE\_2G = 0x00

ACCEL\_RANGE\_4G = 0x08

ACCEL\_RANGE\_8G = 0x10

ACCEL\_RANGE\_16G = 0x18

GYRO\_RANGE\_250DEG = 0x00

GYRO\_RANGE\_500DEG = 0x08

GYRO\_RANGE\_1000DEG = 0x10

GYRO\_RANGE\_2000DEG = 0x18

PWR\_MGMT\_1 = 0x6B

PWR\_MGMT\_2 = 0x6C

ACCEL\_XOUT0 = 0x3B

ACCEL\_YOUT0 = 0x3D

ACCEL\_ZOUT0 = 0x3F

TEMP\_OUT0 = 0x41

GYRO\_XOUT0 = 0x43

GYRO\_YOUT0 = 0x45

GYRO\_ZOUT0 = 0x47

ACCEL\_CONFIG = 0x1C

GYRO\_CONFIG = 0x1B

def \_\_init\_\_(self, address, bus=1):

self.address = address

self.bus = smbus.SMBus(bus)

self.bus.write\_byte\_data(self.address, self.PWR\_MGMT\_1, 0x00)

def read\_i2c\_word(self, register):

high = self.bus.read\_byte\_data(self.address, register)

low = self.bus.read\_byte\_data(self.address, register + 1)

value = (high << 8) + low

if (value >= 0x8000):

return -((65535 - value) + 1)

else:

return value

def get\_accel\_data(self, g = False):

x = self.read\_i2c\_word(self.ACCEL\_XOUT0)

y = self.read\_i2c\_word(self.ACCEL\_YOUT0)

z = self.read\_i2c\_word(self.ACCEL\_ZOUT0)

accel\_scale\_modifier = None

accel\_range = self.read\_accel\_range(True)

if accel\_range == self.ACCEL\_RANGE\_2G:

accel\_scale\_modifier = self.ACCEL\_SCALE\_MODIFIER\_2G

elif accel\_range == self.ACCEL\_RANGE\_4G:

accel\_scale\_modifier = self.ACCEL\_SCALE\_MODIFIER\_4G

elif accel\_range == self.ACCEL\_RANGE\_8G:

accel\_scale\_modifier = self.ACCEL\_SCALE\_MODIFIER\_8G

elif accel\_range == self.ACCEL\_RANGE\_16G:

accel\_scale\_modifier = self.ACCEL\_SCALE\_MODIFIER\_16G

else:

accel\_scale\_modifier = self.ACCEL\_SCALE\_MODIFIER\_2G

x = x / accel\_scale\_modifier

y = y / accel\_scale\_modifier

z = z / accel\_scale\_modifier

if g is True:

return {'x': x, 'y': y, 'z': z}

elif g is False:

x = x \* self.GRAVITIY\_MS2

y = y \* self.GRAVITIY\_MS2

z = z \* self.GRAVITIY\_MS2

return {'x': x, 'y': y, 'z': z}

def set\_gyro\_range(self, gyro\_range):

self.bus.write\_byte\_data(self.address, self.GYRO\_CONFIG, 0x00)

self.bus.write\_byte\_data(self.address, self.GYRO\_CONFIG, gyro\_range)

def read\_gyro\_range(self, raw = False):

raw\_data = self.bus.read\_byte\_data(self.address, self.GYRO\_CONFIG)

if raw is True:

return raw\_data

elif raw is False:

if raw\_data == self.GYRO\_RANGE\_250DEG:

return 250

elif raw\_data == self.GYRO\_RANGE\_500DEG:

return 500

elif raw\_data == self.GYRO\_RANGE\_1000DEG:

return 1000

elif raw\_data == self.GYRO\_RANGE\_2000DEG:

return 2000

else:

return -1

def get\_gyro\_data(self):

x = self.read\_i2c\_word(self.GYRO\_XOUT0)

y = self.read\_i2c\_word(self.GYRO\_YOUT0)

z = self.read\_i2c\_word(self.GYRO\_ZOUT0)

gyro\_scale\_modifier = None

gyro\_range = self.read\_gyro\_range(True)

if gyro\_range == self.GYRO\_RANGE\_250DEG:

gyro\_scale\_modifier = self.GYRO\_SCALE\_MODIFIER\_250DEG

elif gyro\_range == self.GYRO\_RANGE\_500DEG:

gyro\_scale\_modifier = self.GYRO\_SCALE\_MODIFIER\_500DEG

elif gyro\_range == self.GYRO\_RANGE\_1000DEG:

gyro\_scale\_modifier = self.GYRO\_SCALE\_MODIFIER\_1000DEG

elif gyro\_range == self.GYRO\_RANGE\_2000DEG:

gyro\_scale\_modifier = self.GYRO\_SCALE\_MODIFIER\_2000DEG

else:

gyro\_scale\_modifier = self.GYRO\_SCALE\_MODIFIER\_250DEG

x = x / gyro\_scale\_modifier

y = y / gyro\_scale\_modifier

z = z / gyro\_scale\_modifier

return {'x': x, 'y': y, 'z': z}

def get\_all\_data(self):

temp = self.get\_temp()

accel = self.get\_accel\_data()

gyro = self.get\_gyro\_data()

return [accel, gyro, temp]

if \_\_name\_\_ == "\_\_main\_\_":

mpu = mpu6050(0x68)

print(mpu.get\_temp())

accel\_data = mpu.get\_accel\_data()

print(accel\_data['x'])

print(accel\_data['y'])

print(accel\_data['z'])

gyro\_data = mpu.get\_gyro\_data()

print(gyro\_data['x'])

print(gyro\_data['y'])

print(gyro\_data['z'])

os.environ['SDL\_VIDEO\_CENTERED'] = "True"

pygame.init()

width, height = 1200, 600

screen=pygame.display.set\_mode((width, height))

backdrop = pygame.image.load('mountain range.png')

backdrop = pygame.transform.scale(backdrop, (10000,600))

player = pygame.image.load('mario back.png')

player = pygame.transform.scale(player, (2000,1000))

playerpos =[550,400]

roadwidth = 100

x = 100

view = 0

pview = 0

road = []

sensor = mpu6050(0x68)

for \_ in range(100):

for x in range(100):

road.append(0)

for x in range(0,250):

road.append(x)

for x in range(50):

road.append(250)

for x in range(250,0,-1):

road.append(x)

for x in range(50):

road.append(0)

for x in range(0,-250,-1):

road.append(x)

for x in range(50):

road.append(-250)

for x in range(-250,0):

road.append(x)

y = 50

iy =0

speed = 0

keys = [False,False,False,False]

vibrate = 2

speedchange = -4

p = pygame.time.get\_ticks()

while True:

x\_gyro1 = []

x\_gyro2 = 0

    old\_val = sensor.get\_gyro\_data(x)

    for \_ in range(3):

        for x in range(5):

            x\_gyro2 += sensor.get\_gyro\_data(x)

        x\_gyro2 = x\_gyro2/5

        x\_gyro1.append(x\_gyro2 - old\_val)

    for x in x\_gyro:

        x\_change += x

    x\_change = x\_change/3

    pview += x\_change/3

    view += x\_change

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

exit(0)

if event.type == pygame.KEYDOWN:

if event.key==K\_a and keys[3] == False:

player = pygame.image.load('mario left.png')

player = pygame.transform.scale(player, (200,200))

keys[1]=True

elif event.key==K\_d and keys[1] == False:

keys[3]=True

player = pygame.image.load('mario left.png')

player = pygame.transform.flip(player, True, False)

player = pygame.transform.scale(player, (200,200))

if event.type == pygame.KEYUP:

if event.key == K\_a and keys[3] == False:

player = pygame.image.load('mario back.png')

player = pygame.transform.scale(player, (2000,1000))

keys[1]=False

elif event.key == K\_d and keys[1] == False:

player = pygame.image.load('mario back.png')

player = pygame.transform.scale(player, (2000,1000))

keys[3]=False

'''if keys[1]:

view += 7

pview += 2

if keys[3]:

view -= 7

pview -= 2'''

if y\_orientation >= 1:

keys[1] = True

elif y\_orientation <= -1:

keys[3] = True

else:

keys[1],keys[3] = False,False

vibrate = -vibrate

playerpos[1] += vibrate

screen.fill(0)

y = road[iy]

iy += 1

speed += speedchange

view += road[iy]/25

rwidth = 25

screen.blit(backdrop, (-(y/2) - 1000 + (pview\*2),0 - 135))

for x in range(1,400):

turn = (x - 400)\*-1

speed += 1

pygame.draw.rect(screen, (40,255,120), (0,200 + x,view + width/2 - rwidth/2 + ((width/2 - rwidth / 2)\* ((turn/100000) \* y)),1),0)

pygame.draw.rect(screen, (255,200,120), (view + width/2 + rwidth/2 + ((width/2 - rwidth / 2)\* ((turn/100000) \* y)),200 + x,10000,1),0)

pygame.draw.rect(screen, (50,50,50), (view + width/2 - rwidth/2 + ((width/2 - rwidth / 2)\* ((turn/100000) \* y)),200 + x,rwidth + turn/3,1),0)

pygame.draw.rect(screen, (150,150,150), (view + width/2 - rwidth/2 + ((width/2 - rwidth / 2)\* ((turn/100000) \* y)),200 + x,(rwidth + turn/3)/20,1),0)

pygame.draw.rect(screen, (150,150,150), (view + width/2 - rwidth/2 + ((width/2 - rwidth / 2)\* ((turn/100000) \* y)) + (rwidth + turn/3) - (rwidth + turn/3)/20,200 + x,(rwidth + turn/3)/20,1),0)

if speed%20 != 0 and speed%20 != 1 and speed%20 != 2 and speed%20 != 3 and speed%20 != 4 and speed%20 != 5 and speed%20 != 6:

pygame.draw.rect(screen, (200,200,200), (view + width/2 - rwidth/2 + ((width/2 - rwidth / 2)\* ((turn/100000) \* y)) + (rwidth + turn/3)/2 - (rwidth + turn/3)/40,200 + x,(rwidth + turn/3)/20,1),0)

if speed%40 == 0:

pygame.draw.rect(screen, (50,50,50), (view + width/2 - rwidth/2 + ((width/2 - rwidth / 2)\* ((turn/100000) \* y)),200 + x,(rwidth + turn/3)/20,100),0)

pygame.draw.rect(screen, (50,50,50), (view + width/2 - rwidth/2 + ((width/2 - rwidth / 2)\* ((turn/100000) \* y)) + (rwidth + turn/3) - (rwidth + turn/3)/20,200 + x,(rwidth + turn/3)/20,10),0)

rwidth += 3

speedchange = -4

if view >= 550 or view <= -550:

time.sleep(.01)

'''add timer on'''

screen.blit(player,playerpos)

roadwidth = 100

x = 100

font = pygame.font.Font('Road\_Rage.ttf',50)

text = font.render(str((pygame.time.get\_ticks() - p)/1000), True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2, 100)

screen.blit(text,textRect)

pygame.display.update()

Driving game menu:

The driving game menu is where the user can navigate to either the driving game the help page or the leader boards.(it will be controlled by the led tracking when that is finished rather than a mouse). The animations for the text and background start when the user hovers over the location of the text with the mouse cursor and when it is selected by clicking on the program spawns a command prompt in order to call the file that the user selected.

import pygame

from pygame.locals import \*

import os

import time

pygame.init()

os.environ['SDL\_VIDEO\_CENTERED'] = "True"

width, height = 1200,600

screen = pygame.display.set\_mode((width,height))

background = pygame.image.load('background.PNG')

background = pygame.transform.scale(background, (1200,600))

menu\_car = pygame.image.load('poster car.PNG')

menu\_car\_pos = [-280,450]

menu\_car = pygame.transform.scale(menu\_car, (280,140))

lcount = 0

lcount2 = 4

def effects(lcount,lcount2):

lframe\_size = [50,350]

lframe = ['1','2','3','2','3','2','1','1','2','1','1','2','1','2','1']

if lcount < len(lframe) - 1:

lcount += 1

else:

lcount = 0

if lcount2 < len(lframe) - 1:

lcount2 += 1

else:

lcount2 = 0

lightning = pygame.image.load('lightning' + lframe[lcount] + '.png')

lightning = pygame.transform.scale(lightning,lframe\_size)

lightning2 = pygame.image.load('lightning' + lframe[lcount2] + '.png')

lightning2 = pygame.transform.scale(lightning2,lframe\_size)

screen.blit(lightning, (350,-20))

screen.blit(lightning2, (800,-100))

screen.blit(lightning, (1000,0))

screen.blit(lightning2, (100,-50))

return lcount,lcount2

menu = True

vibrate = 1

start\_an = False

start\_an2 = False

startpos = [width + 200,245]

helppos = [-100,startpos[1] + 225]

scorepos = [width + 100, helppos[1]]

schange = 0

change = 0

while menu:

click = False

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

click = True

screen.blit(background,(0,0))

screen.blit(menu\_car, menu\_car\_pos)

if menu\_car\_pos[0] < 440:

menu\_car\_pos[0] += 20

else:

vibrate = -vibrate

menu\_car\_pos[1] += vibrate

start\_an = True

mouse = pygame.mouse.get\_pos()

if start\_an and startpos[0] != 600:

startpos[0] -= 50

elif startpos[0] == 600:

start\_an2 = True

if start\_an2 and startpos[0] != helppos[0] + 400:

helppos[0] += 50

scorepos[0] -= 50

if startpos[0] == helppos[0] + 400 and start\_an2:

if schange == 0:

change = -1

time.sleep(.01)

elif schange == -5:

change = 1

time.sleep(.01)

schange += change

lcount,lcount2 = effects(lcount,lcount2)

if startpos[0] - 150 < mouse[0] < startpos[0] + 150 and startpos[1] - 30 < mouse[1] < startpos[1] + 30:

font = pygame.font.Font('Road\_Rage.ttf',90 + schange\*2)

text = font.render('START', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (startpos[0], startpos[1])

screen.blit(text,textRect)

if click:

os.system('dgame.py')

else:

font = pygame.font.Font('Road\_Rage.ttf',60)

text = font.render('START', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (startpos[0], startpos[1])

screen.blit(text,textRect)

if helppos[0] - 100 < mouse[0] < helppos[0] + 100 and helppos[1] - 30 < mouse[1] < helppos[1] + 30:

font = pygame.font.Font('Road\_Rage.ttf',45 + schange)

text = font.render('help', True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (helppos[0], helppos[1])

screen.blit(text,textRect)

if click:

os.system('dhelp.py')

else:

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render('help', True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (helppos[0], helppos[1])

screen.blit(text,textRect)

if scorepos[0] - 100 < mouse[0] < scorepos[0] + 100 and scorepos[1] - 30 < mouse[1] < scorepos[1] + 30:

font = pygame.font.Font('Road\_Rage.ttf',45 + schange)

text = font.render('highscores', True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (scorepos[0], scorepos[1])

screen.blit(text,textRect)

if click:

os.system('dleaderboard.py')

else:

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render('highscores', True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (scorepos[0], scorepos[1])

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',130)

text = font.render('Road Rage', True, (100,100,100),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 5, 105)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',130)

text = font.render('Road Rage', True, (255,50,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2, 100)

screen.blit(text,textRect)

pygame.display.update()

**driving help menu:**

The help menu of the driving game is where the user can see the instructions on how the driving game works, go to the credits page where they can see all the names of the people that were used to create the animations for the driving game and/or go to the calibration page where they can either reset the orientation of the controller or the position of where they are in the camera fpr the led tracking.

import pygame

from pygame.locals import \*

import os

import time

pygame.init()

os.environ['SDL\_VIDEO\_CENTERED'] = "True"

width, height = 1200,600

screen = pygame.display.set\_mode((width,height))

background = pygame.image.load('background.PNG')

background = pygame.transform.scale(background, (1200,600))

lcount = 0

lcount2 = 4

def effects(lcount,lcount2):

lframe\_size = [50,350]

lframe = ['1','2','3','2','3','2','1','1','2','1','1','2','1','2','1']

if lcount < len(lframe) - 1:

lcount += 1

else:

lcount = 0

if lcount2 < len(lframe) - 1:

lcount2 += 1

else:

lcount2 = 0

lightning = pygame.image.load('lightning' + lframe[lcount] + '.png')

lightning = pygame.transform.scale(lightning,lframe\_size)

lightning2 = pygame.image.load('lightning' + lframe[lcount2] + '.png')

lightning2 = pygame.transform.scale(lightning2,lframe\_size)

screen.blit(lightning, (350,-20))

screen.blit(lightning2, (800,-100))

screen.blit(lightning, (1000,0))

screen.blit(lightning2, (100,-50))

return lcount,lcount2

menu = True

vibrate = 1

start\_an = False

start\_an2 = False

instpos = [width/2,245]

calpos = [width/2,instpos[1] + 100]

schange = 0

change = 0

while menu:

click = False

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

click = True

screen.blit(background,(0,0))

mouse = pygame.mouse.get\_pos()

lcount,lcount2 = effects(lcount,lcount2)

if instpos[0] - 150 < mouse[0] < instpos[0] + 150 and instpos[1] - 30 < mouse[1] < instpos[1] + 30:

font = pygame.font.Font('Road\_Rage.ttf',60)

text = font.render('Instructions', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (instpos[0], instpos[1])

screen.blit(text,textRect)

else:

font = pygame.font.Font('Road\_Rage.ttf',40)

text = font.render('Insrutctions', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (instpos[0], instpos[1])

screen.blit(text,textRect)

if calpos[0] - 150 < mouse[0] < calpos[0] + 150 and calpos[1] - 30 < mouse[1] < calpos[1] + 30:

font = pygame.font.Font('Road\_Rage.ttf',60)

text = font.render('calibrate', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (calpos[0], calpos[1])

screen.blit(text,textRect)

else:

font = pygame.font.Font('Road\_Rage.ttf',40)

text = font.render('calibrate', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (calpos[0], calpos[1])

screen.blit(text,textRect)

if 50 < mouse[0] < 150 and 530< mouse[1] < 570:

font = pygame.font.Font('Road\_Rage.ttf',45)

text = font.render('back', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (100, 550)

screen.blit(text,textRect)

if click:

import background

else:

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render('back', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (100, 550)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',100)

text = font.render('Help', True, (100,100,100),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 5, 105)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',100)

text = font.render('Help', True, (255,50,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2, 100)

screen.blit(text,textRect)

pygame.display.update()

**Driving leaderboards:**

The driving game leader boards is where the user can view the top 7 scores on the driving game (I am going to make an option to allow them to view their personal top 7 as well). It works by opening and reading the scores from the csv file that they are saved to then performing a bubble sort on them in order to put them in order from highest to lowest. It then takes the name and the score of the top 7 users and writes them in a table that is displayed in the pygame window. (the player icons of the top 2 players will be displayed either side of the table in the future but right now it is just me and joh as these are the only images I have.).

import pygame

from pygame.locals import \*

import os

import time

import csv

pygame.init()

os.environ['SDL\_VIDEO\_CENTERED'] = "True"

width, height = 1200,600

screen = pygame.display.set\_mode((width,height))

background = pygame.image.load('background.PNG')

background = pygame.transform.scale(background, (1200,600))

menu = True

leaderboard = []

leaderboard\_name = []

with open('dleaderboards.csv') as csv\_file:

csv\_reader = csv.reader(csv\_file, delimiter=',')

line\_count = 0

for row in csv\_reader:

if line\_count != 0:

leaderboard.append(row[1])

leaderboard\_name.append(row[0])

line\_count += 1

else:

line\_count += 1

csv\_file.close()

count = 0

i = 0

p = 0

while p != len(leaderboard):

j = i + 1

if leaderboard[i] < leaderboard[j]:

savenum = leaderboard[j]

leaderboard[j] = leaderboard[i]

leaderboard[i] = savenum

savenum\_name = leaderboard\_name[j]

leaderboard\_name[j] = leaderboard\_name[i]

leaderboard\_name[i] = savenum\_name

p = 0

else:

p += 1

i += 1

if i == len(leaderboard) - 1:

i = 0

count += 1

'''with open('fileName.csv', 'w') as f:

for x in range(len(leaderboard)):

f.write(leaderboard[x],)

f.write( leaderboard\_name[x])'''

lcount = 0

lcount2 = 4

fcount = 0

def effects(lcount,lcount2,fcount):

lframe\_size = [50,350]

lframe = ['1','2','3','2','3','2','1','1','2','1','1','2','1','2','1']

fframe = ['1','2','3','4','5','6','7','6','5','4','3','2',]

if lcount < len(lframe) - 1:

lcount += 1

else:

lcount = 0

if lcount2 < len(lframe) - 1:

lcount2 += 1

else:

lcount2 = 0

if fcount < len(fframe) - 1:

fcount += 1

else:

fcount = 0

kweku = pygame.image.load('kweku' + fframe[fcount] + '.png')

kweku = pygame.transform.scale(kweku,(200,200))

joh = pygame.image.load('joh' + fframe[fcount] + '.png')

joh = pygame.transform.scale(joh,(200,200))

lightning = pygame.image.load('lightning' + lframe[lcount] + '.png')

lightning = pygame.transform.scale(lightning,lframe\_size)

lightning2 = pygame.image.load('lightning' + lframe[lcount2] + '.png')

lightning2 = pygame.transform.scale(lightning2,lframe\_size)

screen.blit(lightning, (300,-20))

screen.blit(lightning2, (800,-100))

screen.blit(lightning, (1000,0))

screen.blit(lightning2, (100,-50))

screen.blit(kweku, (100,300))

screen.blit(joh, (width - 375,300))

return lcount,lcount2,fcount

while menu:

click = False

mouse = pygame.mouse.get\_pos()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

click = True

screen.blit(background,(0,0))

lcount,lcount2,fcount = effects(lcount,lcount2,fcount)

font = pygame.font.Font('Road\_Rage.ttf',100)

text = font.render('leaderboards', True, (100,100,100),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 5, 105)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',100)

text = font.render('leaderboards', True, (255,50,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2, 100)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[0], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 200)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[1], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 250)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[2], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 300)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[3], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 350)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[4], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 400)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[5], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 450)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[0], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 200)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[1], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 250)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[2], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 300)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[3], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 350)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[4], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 400)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[5], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 450)

screen.blit(text,textRect)

if 50 < mouse[0] < 150 and 530< mouse[1] < 570:

font = pygame.font.Font('Road\_Rage.ttf',45)

text = font.render('back', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (100, 550)

screen.blit(text,textRect)

if click:

import background

else:

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render('back', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (100, 550)

screen.blit(text,textRect)

pygame.display.update()import pygame

from pygame.locals import \*

import os

import time

import csv

pygame.init()

os.environ['SDL\_VIDEO\_CENTERED'] = "True"

width, height = 1200,600

screen = pygame.display.set\_mode((width,height))

background = pygame.image.load('background.PNG')

background = pygame.transform.scale(background, (1200,600))

menu = True

leaderboard = []

leaderboard\_name = []

with open('dleaderboards.csv') as csv\_file:

csv\_reader = csv.reader(csv\_file, delimiter=',')

line\_count = 0

for row in csv\_reader:

if line\_count != 0:

leaderboard.append(row[1])

leaderboard\_name.append(row[0])

line\_count += 1

else:

line\_count += 1

csv\_file.close()

count = 0

i = 0

p = 0

while p != len(leaderboard):

j = i + 1

if leaderboard[i] < leaderboard[j]:

savenum = leaderboard[j]

leaderboard[j] = leaderboard[i]

leaderboard[i] = savenum

savenum\_name = leaderboard\_name[j]

leaderboard\_name[j] = leaderboard\_name[i]

leaderboard\_name[i] = savenum\_name

p = 0

else:

p += 1

i += 1

if i == len(leaderboard) - 1:

i = 0

count += 1

'''with open('fileName.csv', 'w') as f:

for x in range(len(leaderboard)):

f.write(leaderboard[x],)

f.write( leaderboard\_name[x])'''

lcount = 0

lcount2 = 4

fcount = 0

def effects(lcount,lcount2,fcount):

lframe\_size = [50,350]

lframe = ['1','2','3','2','3','2','1','1','2','1','1','2','1','2','1']

fframe = ['1','2','3','4','5','6','7','6','5','4','3','2',]

if lcount < len(lframe) - 1:

lcount += 1

else:

lcount = 0

if lcount2 < len(lframe) - 1:

lcount2 += 1

else:

lcount2 = 0

if fcount < len(fframe) - 1:

fcount += 1

else:

fcount = 0

kweku = pygame.image.load('kweku' + fframe[fcount] + '.png')

kweku = pygame.transform.scale(kweku,(200,200))

joh = pygame.image.load('joh' + fframe[fcount] + '.png')

joh = pygame.transform.scale(joh,(200,200))

lightning = pygame.image.load('lightning' + lframe[lcount] + '.png')

lightning = pygame.transform.scale(lightning,lframe\_size)

lightning2 = pygame.image.load('lightning' + lframe[lcount2] + '.png')

lightning2 = pygame.transform.scale(lightning2,lframe\_size)

screen.blit(lightning, (300,-20))

screen.blit(lightning2, (800,-100))

screen.blit(lightning, (1000,0))

screen.blit(lightning2, (100,-50))

screen.blit(kweku, (100,300))

screen.blit(joh, (width - 375,300))

return lcount,lcount2,fcount

while menu:

click = False

mouse = pygame.mouse.get\_pos()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

click = True

screen.blit(background,(0,0))

lcount,lcount2,fcount = effects(lcount,lcount2,fcount)

font = pygame.font.Font('Road\_Rage.ttf',100)

text = font.render('leaderboards', True, (100,100,100),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 5, 105)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',100)

text = font.render('leaderboards', True, (255,50,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2, 100)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[0], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 200)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[1], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 250)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[2], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 300)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[3], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 350)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[4], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 400)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard\_name[5], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 - 150 , 450)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[0], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 200)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[1], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 250)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[2], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 300)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[3], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 350)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[4], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 400)

screen.blit(text,textRect)

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render(leaderboard[5], True, (255,255,255),(0))

textRect = text.get\_rect()

textRect.center = (width/2 + 100 , 450)

screen.blit(text,textRect)

if 50 < mouse[0] < 150 and 530< mouse[1] < 570:

font = pygame.font.Font('Road\_Rage.ttf',45)

text = font.render('back', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (100, 550)

screen.blit(text,textRect)

if click:

import background

else:

font = pygame.font.Font('Road\_Rage.ttf',30)

text = font.render('back', True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (100, 550)

screen.blit(text,textRect)

pygame.display.update()

**Rhythm game:**

The rhythm game works by taking the time that has passed since the game started and comparing it to the times that there is supposed to be a hit (currently pre-set by me) each time that the user click does an action to call the ‘hit check’ function and if the timer is within .5 seconds of the one of the pre-set times the hit will be successful, and the hit animation will begin to run to let the user know that their move was successful. The squares that show the user when to time their moves are then drawn on the screen and their distance from the top are drawn by using the difference from their set time to compared to the timer, in order to ensure that the user gets an accurate representation of the timer and the set times. As well as this the timer is also displayed so the user can time their movements better.

import pygame

import os

import pygame

from pygame.locals import \*

import random

import time

os.environ['SDL\_VIDEO\_CENTERED'] = "True"

pygame.init()

width, height = 600, 600

screen=pygame.display.set\_mode((width, height))

keys = [False,False,False,False]

colour = (0,0,255)

size = 50

startan = 10

llsquare,lmsquare,rmsquare,rrsquare = colour,colour,colour,colour

def hit\_check(timer,lltimes,lmtimes,rmtimes,rrtimes):

for x in llhappening:

if timer - .5 <= x <= timer + .5:

return True

for x in lmhappening:

if timer - .5 <= x <= timer + .5:

return True

for x in rmhappening:

if timer - .5 <= x <= timer + .5:

return True

for x in rrhappening:

if timer - .5 <= x <= timer + .5:

return True

def hit\_animation():

pygame.draw.rect(screen,llsquare,(width/4 - 100,height/20,size + 10,size + 10))

pygame.draw.rect(screen,llsquare,(width/2 - 100,height/20,size + 10,size + 10))

pygame.draw.rect(screen,llsquare,(width\*.75 - 100,height/20,size + 10,size + 10))

pygame.draw.rect(screen,llsquare,(width - 100,height/20,size + 10,size + 10))

start\_time = time.time()

while True:

timer = time.time() - start\_time

llsquare,lmsquare,rmsquare,rrsquare = colour,colour,colour,colour

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

exit(0)

if event.type == KEYDOWN and event.key == K\_LEFT:

if hit\_check(timer,lltimes,lmtimes,rmtimes,rrtimes):

startan = 0

screen.fill(0)

pygame.draw.rect(screen,llsquare,(width/4 - 100,height/20,size,size))

pygame.draw.rect(screen,lmsquare,(width/2 - 100,height/20,size,size))

pygame.draw.rect(screen,rmsquare,(width\*0.75 - 100,height/20,size,size))

pygame.draw.rect(screen,rrsquare,(width - 100,height/20,size,size))

if startan < 10:

llsquare,lmsquare,rmsquare,rrsquare = (0,255,0),(0,255,0),(0,255,0),(0,255,0)

hit\_animation()

pygame.draw.rect(screen, (255,255,255), (width/2 - 5, 0, 10, height))

pygame.draw.rect(screen, (255,255,255), (width/4 - 5, 0, 10, height))

pygame.draw.rect(screen, (255,255,255), (width\*.75 - 5, 0, 10, height))

pygame.draw.rect(screen, (255,255,255), (width - 5, 0, 10, height))

pygame.draw.rect(screen, (255,255,255), (-5, 0, 10, height))

#lltimes = [random.randint(2,20) for x in range(5)]

lltimes = [6,9,10,18]

llhappening = [x for x in lltimes if x <= timer]

for x in llhappening:

lltimes.remove(x)

for x in range(len(lltimes)):

pygame.draw.rect(screen,(255,255,255),(width/4 - 100,(lltimes[x] - timer)\*100 + height/20,size,size))

#lmtimes = [random.randint(2,20) for x in range(5)]

lmtimes = [2,5,7,9]

lmhappening = [x for x in lmtimes if x <= timer]

for x in lmhappening:

lmtimes.remove(x)

for x in range(len(lmtimes)):

pygame.draw.rect(screen,(255,255,255),(width/2 - 100,(lmtimes[x] - timer)\*100 + height/20,size,size))

#rmtimes = [random.randint(2,20) for x in range(5)]

rmtimes = [1,3,13,17]

rmhappening = [x for x in rmtimes if x <= timer]

for x in rmhappening:

rmtimes.remove(x)

for x in range(len(rmtimes)):

pygame.draw.rect(screen,(255,255,255),(width\*.75 - 100,(rmtimes[x] - timer)\*100 + height/20,size,size))

#rrtimes = [random.randint(2,20) for x in range(5)]

rrtimes = [1,3,10,12]

rrhappening = [x for x in rrtimes if x <= timer]

for x in rrhappening:

rrtimes.remove(x)

for x in range(len(rrtimes)):

pygame.draw.rect(screen,(255,255,255),(width - 100,(rrtimes[x] - timer)\*100 + height/20,size,size))

font = pygame.font.Font('Road\_Rage.ttf',40)

text = font.render(str(timer), True, (255,100,200),(0))

textRect = text.get\_rect()

textRect.center = (width/2, height/2)

screen.blit(text,textRect)

startan += 1

pygame.display.update()

**Led Tracking (will be implemented into the games):**

The LED tracking works by using opencv in order to access the main camera of the device (in this case a laptops webcam) and begins a live video capture in colour. It then turns each frame of the video into a hsv image. A range of colours in rgb format are then defined that is used to compare to each pixel of the image in order to check if it is the colour that is wanted (in this case blue). If the pixel is in the range it will be turned into a positive binary value. After this is done the image is turned into a binary image where each pixel is either a 1 or a 0 based on whether it is the colour that the camera is searching for.

import cv2

import numpy as numpy

device = cv2.VideoCapture(0)

while True:

ret, frame = device.read()

hsv = cv2.cvtColor(frame,cv2.COLOR\_BGR2HSV)

lower\_range = np.array([110,50,50])

upper\_range = np.array([130,255,255])

mask = cv2.inRange(hsv, lower\_range,upper\_range)

cv2.imshow('Frame',frame)

result = cv2.bitwise\_and(frame,frame,mask=mask)

cv2.imshow('Result',result)  
  
if cv2.waiKey(1) == 27:

break

device.release()

cv2.destroyAllWindows()