BAHÇEŞEHİR UNIVERSITY
FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

GYRONOME

Wearable Virtual Reality Gear

Submitted by

Berk ÇETİNSAYA	Eren ATAŞ	Nazım Gürkan DEMİR
1247083	1334129	1380949

Advisor

Alkan SOYSAL

İSTANBUL, MAY 2016

BAHÇEŞEHİR UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

This project work submitted by Berk ÇETİNSAYA, Eren ATAŞ and Nazım Gürkan DEMİR has been done under my supervision. I hereby state that the work outlined in this report satisfies the requirements of the compulsory "Capstone Project" course, and Berk ÇETİNSAYA, Eren ATAŞ and Nazım Gürkan DEMİR can take the capstone project examination.

Signature and Date

…….

Alkan SOYSAL

Berk ÇETİNSAYA, Eren ATAŞ and Nazım Gürkan DEMİR have taken and passed the capstone project examination in our presence on May ……., 2016. We hereby certify that this project work fulfills all requirements of "Capstone Project" course.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. ……..	……..
2. ……..	……..
3. ……..	……..

I have reviewed this capstone project report submitted by Berk ÇETİNSAYA, Eren ATAŞ and Nazım Gürkan DEMİR, I hereby endorse the project work described here as a "Capstone Project I."

SUMMARY

Gyronome is an affordable, easy to use wearable virtual gear that will enable the user to interact with artificial reality entities by using the gestures of the user's body.

Since the dawn of the video gaming, game developers and console makers have always tried to find new ways for player to interact with game environment. Besides using mouse-keyboard or joystick to provide input to system, target recognizing model guns and motion detecting controllers were also invented. Players that used these controllers also wanted to use their bodies as a controller. Therefore, devices that accurately tracks body movements gained traction.

Gyronome is one of these devices that tracks user movements. It is done via gyroscope/accelerometer sensors, Gyrons, that calculate limbs' rotation and orientation. A Gyron is strapped on every major limb on the user's body. Every Gyron is connected to the main unit via cables. The main unit, Arduino 101 board, collects the motion data and transfers it through serial connection to PC. Then, the PC application, game or simulation, takes the motion data and applies it to humanoid model, which generates the interaction.

Throughout the development of the project, many difficulties and technical insufficiencies have been faced. First, multiple sensor addressing problem has been solved by using an I2C multiplexer. Once connecting more than one sensor to main unit has been solved, an Arduino shield, a PCB that is attached to boards to extend capabilities, has been designed to attach Gyrons properly. Later, data were collected from the Gyrons and sent to PC to control a model on screen. For demonstration, several applications have been prepared. One of them is an Unity game engine application and the other one is NodeJS based web application.

ÖZET

Gyronome (Cayronom), ucuz, kolayca kullanılabilen, giyilebilir, kullanıcının vücut hareketlerini kullanarak bilgisayarda oluşturulmuş bir modeli kontrol etmesini sağlayan sanal gerçeklik donanımıdır.

Video oyunlar ilk çıktığından beri, oyun geliştiricileri ve konsol yapımcıları, oyuncuların oyun ortamıyla etkileşmeleri için farklı yollar bulmaya çalışmışlardır. Fare-klayve ve kumanda kolu kullanmanın haricinde, hedef tanıyabilen model silahlar ve el hareketlerini algılayabilen kontrolcüler icat edildi. Bu cihazları kullanan oyuncular aynı zamanda kendi vücutlarını da bir kontrolcü olarak kullanmak istediler. Bu sayede vücut hareketlerini doğrulukla takip eden cihazlar ilgi çekmeye başladı.

Gyronome kullanıcı hareketlerini takip eden bu cihazlardan biri. Bunu Gyron(Cayron) adı verilen jiroskop/ivmeölçer algılayıcılarının uzuvların dönüşünü ve yönünü hesaplaması ile yapar. Kullanıcının her bir ana uzvuna bir Gyron bağlanır. Her Gyron ana birime kablolar ile bağlanır. Ana birim, Arduino 101 kartı, hareket verilerini toplar ve seri bağlantı yoluyla bilgisayara aktarır. Ardından bir bilgisayar uygulaması, örneğin oyun veya simülasyon, bu veriyi alır ve insansı bir modele uygular. Bu da etkileşimi sağlar

Projenin geliştirme süreci boyunca çeşitli zorluklar ve teknik eksikliklerle karşılaşıldı. Öncelikle birden çok algılayıcıya bağlanma sorunu bir I2C çoklayıcısı ile çözüldü. Birden fazla sensor bağlanması sorunu çözüldükten sonar Gyron'ları düzgün bağlayabilmek için bir Arduino shield, kabiliyetlerini arttırmak için kartlara takılan bir PCB elemanı, tasarlandı. Daha sonra Gyron'lardan veri toplanarak bilgisayar ekranında bir model kontrol edilmek üzere yollandı. Gösterim için çeşitli uygulamalar hazırlandı. Bunlardan bir tane Unity oyun motoru ile hazılanmış bir uygulaması, diğeri ise NodeJS tabanlı bir web uygulamasıdır.

ACKNOWLEDGEMENTS

Firstly, we would like to thank Assoc. Prof. Alkan SOYSAL for providing us necessary resources and for his continuous support throughout our project, Utku GÜLEN, for his invaluable inputs and opinions, Dr. Maksat ASHYRALIYEV, for his priceless information and help on Quaternion equations. Also, we would like to thank our family and friends who have always supported us during this thesis study.

Berk ÇETİNSAYA, Eren ATAŞ, Nazım Gürkan DEMİR

İstanbul, May 2016

1 Contents

Table of Figures

ABBREVIATIONS

MPU: Motion Processing Unit

DMP: Digital Motion Processor

I2C: Inter-Integrated Circuit

PCB: Printed Circuit Board

VR: Virtual Reality

BLE: Bluetooth Low Energy

1 Introduction

Gyronome is a wearable virtual reality motion tracking gear set. We have chosen this project because we started working on wearable technologies and Arduino boards in BAU Future Lab and we were also interested in virtual reality. We wanted to use our readily available knowledge in the area.

There have been some researches on tracking body motions by using inertial sensors. Several methods have been proposed that combines inertial sensor data, such as gyroscope or accelerometer signals, for obtaining motion data [1]. By using these signals, sensors on limbs were used to monitor stroke patients during their recovery [2]. Our project is mainly focused on using the motion data acquired via such methods for gaming and simulation applications.

Main goals of this project were developing a wearable gear that is easy to use and affordable. This gear will be tracking user's body motions and transferring it to an environment where a model human can be controlled with movement input. It also is essential that the gear is light-weight and modular. We will also prepare several demonstration applications. One of them will be a Unity Game Engine application and another will be a NodeJS based web browser application.

In our first tests, we were able to send motion data successfully to target computer although, there we some serious delay between actual motion and visual feedback on computer screen, which is called latency. Improvements on hardware and software parts substantially decreased this. Further developments on the system made it possible to recognize certain gestures that the user can make, which further enhances the interaction between user and the virtual reality environment.

Our project made it easy to interact with virtual reality entities by using sole body movements, integrate body motion data with computer applications.

Thesis Outline

The outline of this thesis is as follows:

Section 2 presents previous researches and insight on the subject. This includes past experiments and technical knowledge on certain key concepts.

Section 3 describes our project's technical details and inner workings. We will explain what are the components of the gear and how the motion data is gathered and transferred in detail.

Section 4 introduces our experimental and demonstrational results.

Section 5 presents our conclusions about our project, the progress we have made and future works.

2 Related Works

Throughout this project we have used various sources as a starting point and a guide.

Since the project deals with movements of an object, physics inevitably plays an essential role. Proper representation of rotation and movement in 3D space requires certain notations. Methods covered in this project are Euler angles and Quaternions.

The Euler angles has been introduced by Leonhard Euler to describe the orientation of a rigid body using a three dimensional point vector [3]. They are usually denoted as α, β, γ, or φ, θ, ψ.

$$\vartheta = (\varphi, \theta, \psi)$$

To rotate each vector that represents a rigid body, vector is multiplied by an appropriate rotation matrix. Because of certain difficulties, we only used Euler angles at the beginning of the project and switched to quaternions.

Quaternions were first described by Irish mathematician William Rowan Hamilton. It is a number system that further extends the complex numbers. It defines a four-dimensional space. There are two parts of it, one “real” part and three “imaginary” parts. It is usually represented as:

$$q = w + xi + yj + zk$$

Quaternions find uses in both theoretical and applied mathematics, in particular for calculations involving three-dimensional rotations such as in three-dimensional computer graphics, computer vision and crystallographic texture analysis [4].

Gyroscope and accelerometer data combined represents rigid body motion via a procedure call Sensor Fusion [5]. First, the gyroscope angular rate is converted to a quaternion. Here, $\omega(t)$ is the angular rate and $q(t)$ is the normalized quaternion.

$$\frac{dq(t)}{dt} = \frac{1}{2}\omega(t) * q(t)$$

Then, accelerometer data is converted to world coodinates. This means using the Quaternion above to get the appropriate coordinate system for world-frame motion.

$$A_W(t) = q(t) * A_b(t) * q(t)'$$

Later, acceleration measurement feedback quaternion is created as following:

$$qf(t) = \begin{bmatrix} 0 & A_{wy}(t) - A_{wx}(t) & 0 \end{bmatrix} * gain$$

Lastly, accel feedback and gain is used to generate a feedback quaternion which is then added to previous quat along with gyro generated quaternion. The result is a Quaternion that will track the gyroscope measured data, but will drift towards the accelerometer

measurement, according to the value chosen for gain.

The quaternion from a particular Euler sequence can be written as the product of three quaternions:

$$\mathbf{q_A} = \mathbf{q_\phi} \cdot \mathbf{q_\theta} \cdot \mathbf{q_\psi}$$

$$\text{where } \mathbf{q_\psi} = \begin{bmatrix} \cos\frac{\psi}{2} \\ 0 \\ 0 \\ \sin\frac{\psi}{2} \end{bmatrix}, \mathbf{q_\theta} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2} \\ 0 \\ 0 \end{bmatrix}, \mathbf{q_\phi} = \begin{bmatrix} \cos\frac{\phi}{2} \\ 0 \\ \sin\frac{\phi}{2} \\ 0 \end{bmatrix}.$$

After quaternion multiplications, the final quaternion is:

$$\mathbf{q_A} = \begin{bmatrix} \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} - \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} \\ \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \end{bmatrix}$$

The inverse mapping (from quaternion to Euler angles) can be obtained by rearranging the equation above:

$$\boldsymbol{\phi} = \boldsymbol{atan2}(-2q_1q_3 + 2q_0q_2, q_3^2 - q_2^2 - q_1^2 + q_0^2)$$
$$\boldsymbol{\theta} = \boldsymbol{asin}(2q_2q_3 + 2q_0q_1)$$
$$\boldsymbol{\psi} = \boldsymbol{atan2}(-2q_1q_2 + 2q_0q_3, q_2^2 - q_3^2 - q_1^2 + q_0^2)$$

3 Materials and Methods

3.1 Gyronome Module

3.1.1 Components

Our main unit, Gyronome module, consists of 3 components. Arduino 101 board, Gyron modules and our own Arduino shield, which is the mounting point of Gyrons and I2C multiplexer.

Arduino 101 board is an entry-level priced learning and development board that uses Intel Curie Module which provides low-power consumption. Besides having a robust form factor as other Arduino boards, it also has Bluetooth LE capabitity and integrated 6-axis inertial sensor. It uses I2C protocol, a multi-master, multi-slave serial computer bus, to communicate with sensor modules.

Gyron modules consist an inertial sensor (MPU-6050) that collects the motion data thorough its gyroscopes and accelerometers and by the use of Digital Motion Processor(DMP), it calculates the orientation of the module. Gyrons are attached to user's limbs with straps and connected to main unit via 4-wire cables.

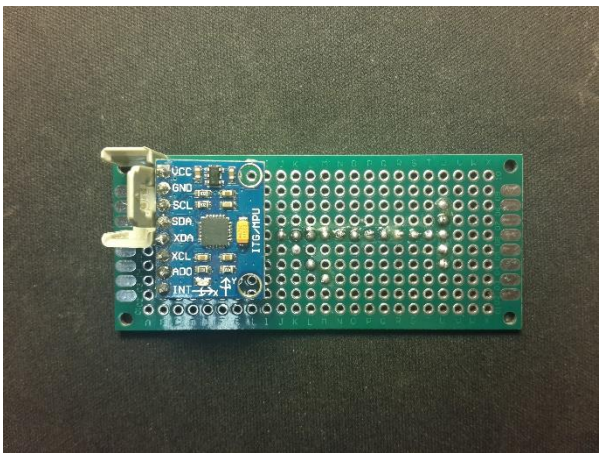


Figure 1: One of the Gyron modules. Number 1.

Our Arduino shield holds the I2C multiplexer, which will be explained on next paragraph, and connection points for Gyrons. The shield is mounted on top of the main module. It gets its power through pins VCC and GND. This power is distributed to Gyrons over the soldered ways on the shield. The are also I2C solder connections for every mounting point of Gyrons, which links them to I2C multiplexer.

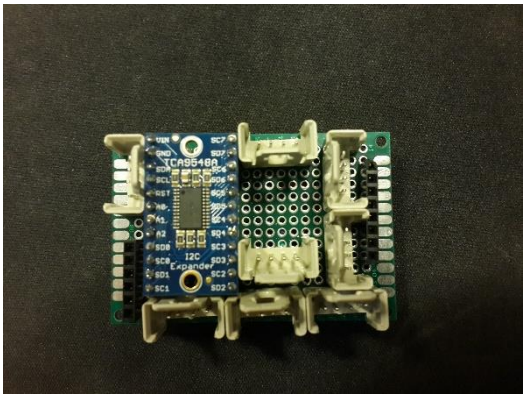


Figure 2: Shield. Top view.

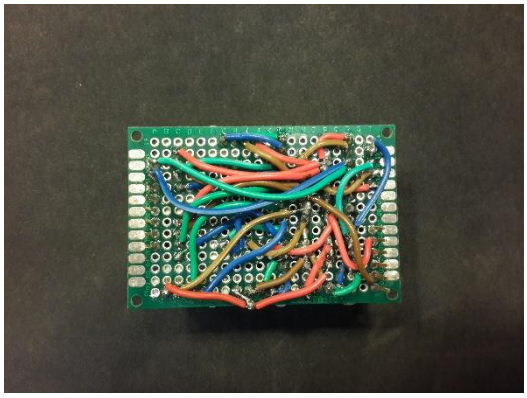


Figure 3: Shield. Bottom view.

As mentioned before, Arduino board uses I2C communication to receive data from sensors. Our first problem presented itself here. Every I2C slave has an address that the master uses when sending data. If two slaves share the same address, there would be conflicts on data transfer. This was the case when multiple sensors were connected to the main unit the first time. Gyrons failed to initialize and neither of them have worked. We have overcome this problem by using an I2C multiplexer, which is an integrated circuit that has 8 I2C buses on the circuit board and it can be programmed to select on of the 8 connected slaves. Then, it joins the slave's bus to main bus, so the main module can transfer data between the selected device. This solved our connecting multiple sensors problem. Theoretically, if 8 of these multiplexers connect to total of 64 sensors by multiplexing buses.

Once we have managed to connect and control 8 Gyrons simultaneously, we advanced to transfering motion data and interpreting this data. Several pieces of software were developed for these purposes. An Arduino sketch for transfering data, Unity and NodeJS applications for interpreting data.

3.1.2 Arduino Sketch

For our Arduino sketch, we use several libraries. Establishing connection between sensors and main unit requires I2C library, which is written by Jeff Rowberg [6]. Another library defines Gyron class which holds the data coming from the sensors.

In the sketch, we first setup our variables for holding data and controlling flow of the code. We present certain option on output of the received data. This output formats are used for debugging and interpretation the motion data. We also define the number of Gyron's we use, so the code will try to connect correct amount of modules.

3.2 Unity Game Engine Application

As been told in the introduction of the game part, we have used the game engine Unity, version 5.4.0b13 Personal Edition. The reasons that we have used this game engine are:

- Unity is a cross-platform game engine and it's been used to develop video games for PC, consoles, mobile devices and websites. We wanted to achieve the possibility of making a game for any kind of device focused on PCs.

- Unity has Virtual Reality (VR) support and as a WIP (Work in Progress) we wanted to use this feature in our project as upcoming features in our game.

- Unity is a free game engine and it has a growing community that helps to those who need.

- Developers of Unity created scripting system on their game engine which makes making game easier than other game engines. Scripting is an essential ingredient in all games. Even the simplest game will need scripts to respond to input from the player and arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behaviour of objects or even implement a custom AI system for characters in the game.

- This is the most important reason: Unity supports Arduino's serial ports. We can read data that's coming from Arduino's serial port and process that data.

What we have created in this part as written on the introduction is a 3D Ball catching game. Player controls the character inside a closed room, and tries to catch the spawned balls with body parts, and if the player doesn't catch them, balls will disappear after an amount of time. There are 3 scenes of our game which are SplashScreen (Scene 0), Menu (Scene 1) and RoomTest (Scene 2). Before explaining the game's parts, I should say that the ways that scripts work is really basic. We can define any object in the game in scripts, call any object and do anything that Unity's API allows us to do [7].

Let's see the game in depth.

3.2.1 Splash Screen

The game starts with the Splash screen as the user can see after the Unity logo (This cannot be removed because it is compulsory for Unity's personal edition (free)).

Figure 4: Unity Main Screen

White lined area is Canvas, which helps you to make UI for the games, enabling the main camera with Canvas' rendering mode and to be able to make it look like a good splashscreen for any kind of ratio that user's monitor has, it was mandatory to use canvas. Also, for some reason even that there is an element of Canvas called Image, it was unable to put the logo and text's logo in it so instead of using image we have used box with a 0.01 Z angle ratio, user won't be able to understand the difference.

Figure 5: Splash Screen

This is the splashscreen in our game when the user starts the game. There is a setting in cameras called "Clear Flags", which makes the background any color needed as you can see,

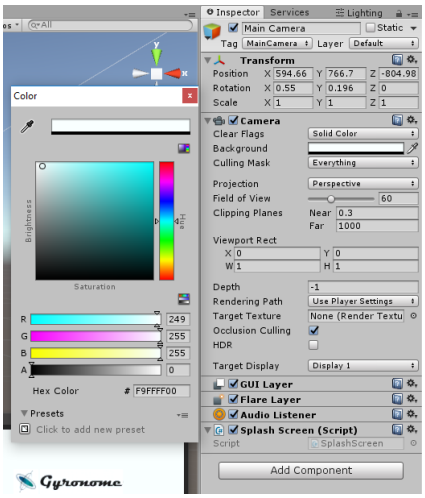
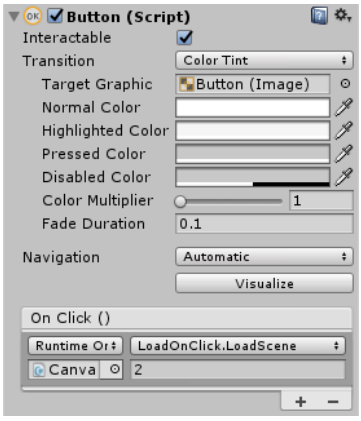


Figure 6: Unity, Inspector Section

After this screen appears when the user starts the game, the C# script called “SplashScreen” which is activated on “Main Camera” (scripts can be assigned to any object as said before) that makes a delay function (3 seconds) and then goes to the scene that needed, which in this case is “Menu”.

3.2.2 Main Menu

Main menu is actually similar to Splash Screen. Canvas has a Text which says “Project Gyronome”, a background (which again made with a box with 0.01 ratio on Z-angle) and 2 buttons for starting the game and exiting the game. There is also a ball that spawns on the screen. The scripts for buttons are activated on Canvas, they are functions so they will be declared on the buttons’ OnClick methods as you can see here,



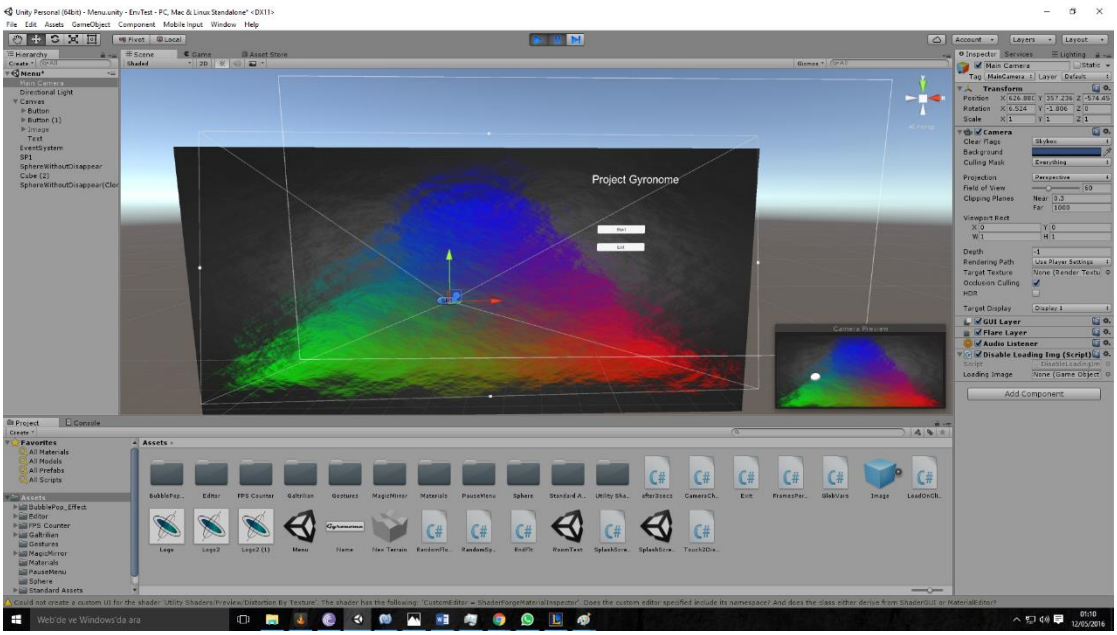


Figure 7: Unity Main Screen, editing Main Menu

There are 4 different scripts for this scene, which are:

- LoadOnClick Script: This is for “Start” button, that enables the user to go to next scene which is “RoomTest” (Scene 2) in this case. There is also an image for loading screen which is also in the scene but it’s not enabled. Clicking the button enables the

script which first enables the loading image to be shown on the scene and then the scene number selected will be loaded.

- Exit Script: This is a simple script for exiting the game.

- DisableLoadingImg Script: This script is to prevent issues that was happening when going back to Main Menu from the game. Loading image was still on the screen and time was frozen. This script disables the loading image and sets the time scale back to 1 when the scene starts.

- RandomSpawn Script: This script is for spawning the balls. It will be explained in depth in game part.

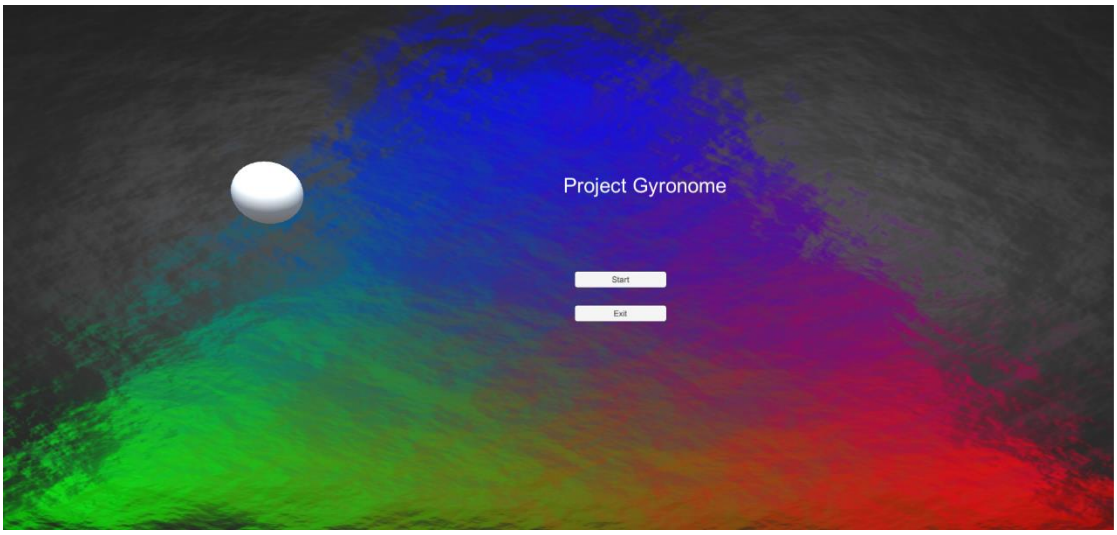


Figure 8: Game Main Menu

This is the screen that user will see when the Main Menu's scene starts. Spawn Point is really close to the Main Camera so, balls that are being spawned can be seen.

3.2.3 Game

Game Scene as known as “RoomTest” Scene will be divided into 3 parts to be able to explain it thoroughly.

3.2.3.a Environment

Our game is inside proportioned 3D cubes that looks likes floor, walls and ceiling which makes it look like a room. The textures of the cubes have been taken from Unity’s Asset Store called “Utility Shaders Free” by Unity [8].

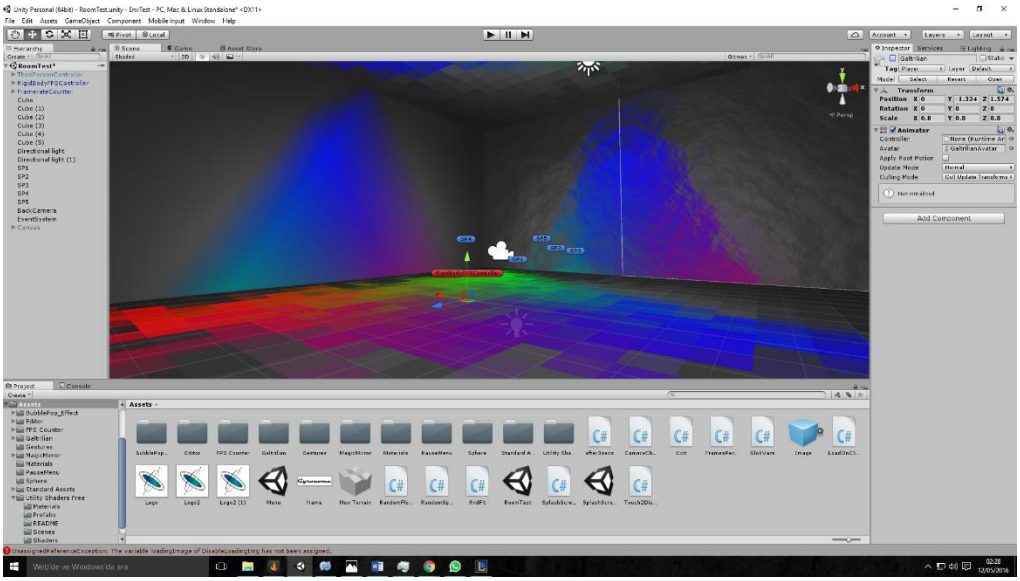


Figure 9: Unity Main Screen, Assets

In here we have another camera in addition to Main Camera to be able to see in a different angle, spawn points, frame rate counter and characters. In depth:

Frame Rate Counter: This has taken from Unity’s Standard Assets. It allows you to see the Frames per second on top of the screen:

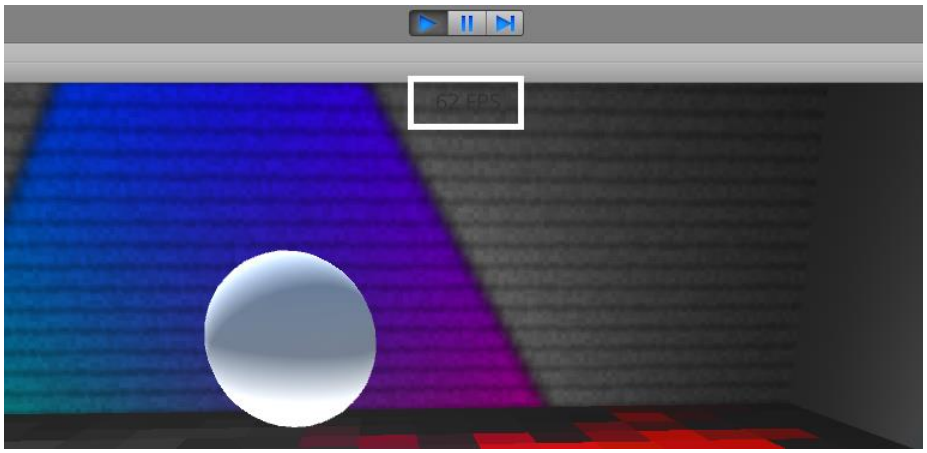


Figure 10: Frame Counter

Back Camera: This camera is for having a different perspective for the game. Instead of using 3rd Person Controller the user can also use this camera. There is a script written called “CameraChange” for this camera, it enables you to change the cameras with pressing “Enter” button on the keyboard. This script is enabled on “SP3” which is a spawn point, since that spawnpoints are actually an empty object that helps for enabling script it has been put there asides with many other script. Here is the view of the Back Camera:

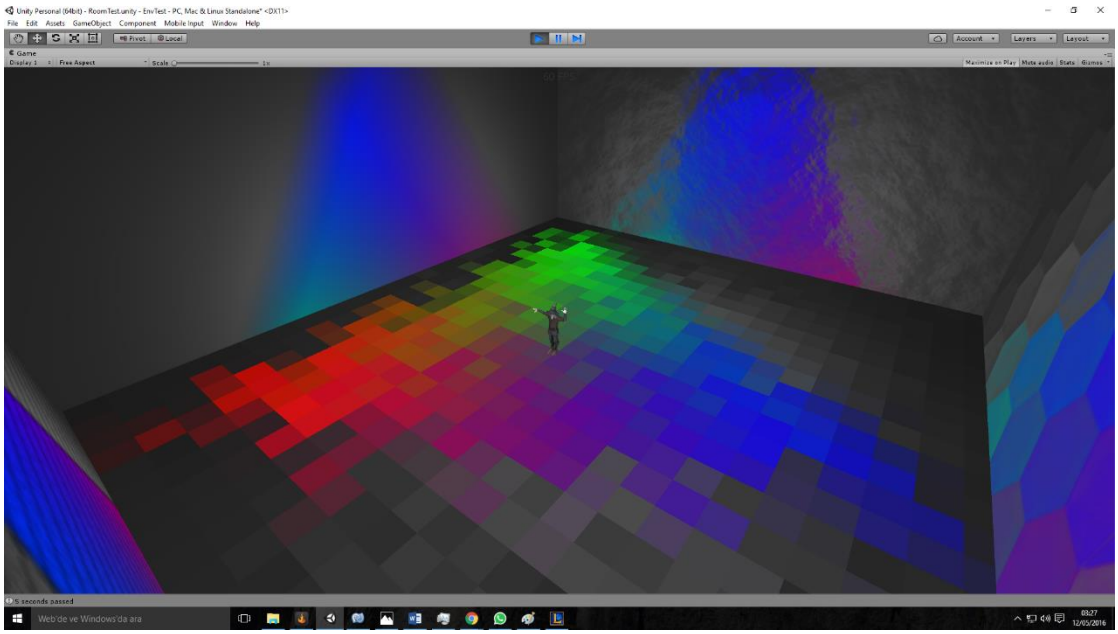


Figure 11: Back Camera

Player Changing Script: Instead of just being able to play with our device, there is also another character added as the Unity’s default Character “Ethan”. And there is also another character “Galtrilian” for using the device with our project. In default, user can play the game with Ethan but if user wants to play with Gyronome, user can press “Right Shift” key and Ethan will disappear and Galtrilian will appear.

Characters: There is a default character in Unity’s Standard Assets called “Ethan” which works quite good with it’s animations and it’s controllers. But for this project we wanted to have a different character. There is a company called “Mixamo” that provides many 3D characters and animations. In this project we are using one of Mixamo’s character called “Galtrilian” but it seems to be a little unstable for the new version of Unity, there will be both default character and Mixamo’s in the project. Also, there is this issue that, when we connect our device to Unity, animations cancel the

movements that coming from the device. In this project there will be no animations of the characters. Here is Ethan:

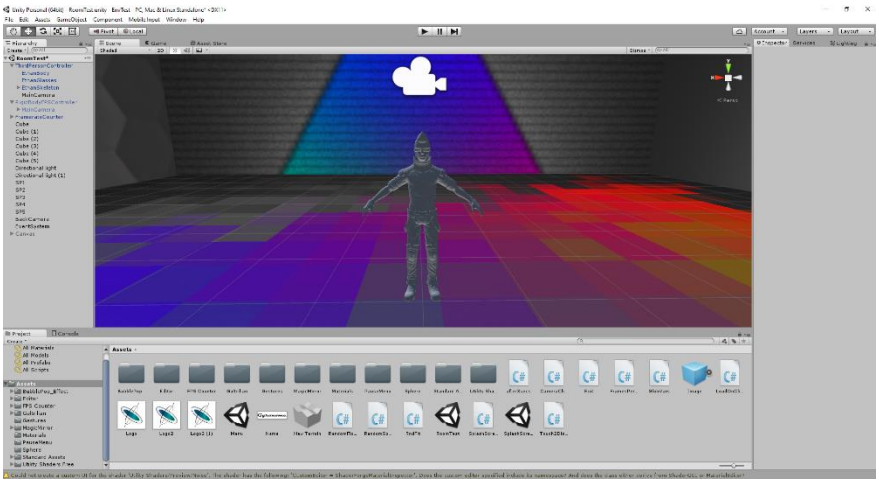


Figure 12: Ethan Model, Third person view

Ethan is being controlled with ThirdPersonController. ThirdPersonController has Ethan's Body, Glasses, Skeleton, and it's camera called "Main Camera" (Because it's our main camera here.) . Skeleton is hierarchical structure for controlling the body and rigging the animations. Our device is controlling the skeleton instead of the body itself, there will be more explanations about this afterwards.

The reason that Ethan is able to stay on the floor and move in any ways is because there is a collision between the floor and Ethan's body which is like real life. The main collision of Ethan's body comes from the ThirdPersonController, called "Capsule Collider":

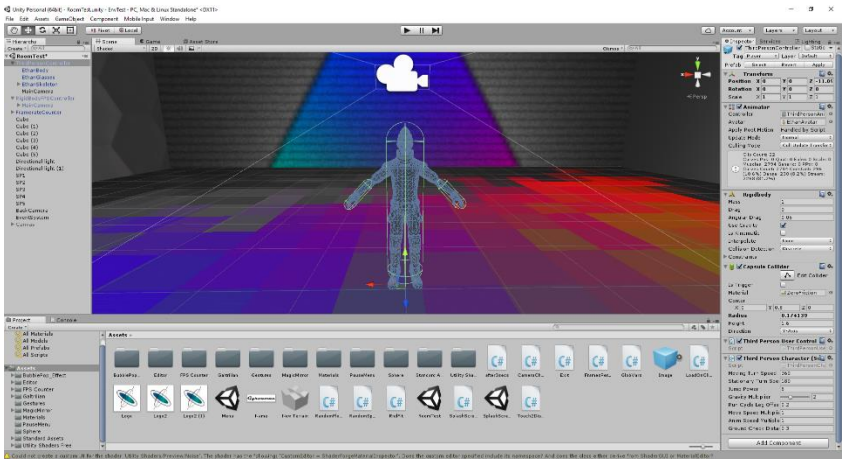


Figure 13: Ethan Model, with Capsule Collider

As can be seen, green lines are the colliders of Ethan. The reason that there are more colliders is that spheres(balls) need to be in contact with colliders to be able to be catched (disappeared). Normally, colliders don't reshape and they stay in the same position according to their parent which in this case is ThirdPersonController, so we have found that adding capsule colliders to his hands will be a solution for spheres to disappear. Otherwise, spheres were just passing through inside of Ethan's arms.

Pause Menu: Pause menu is a menu like all in games, that helps you to pause the game when you need with the pressing of "Escape" button on the keyboard. When the user presses the button, time in the game freezes and the menu's Canvas appears:

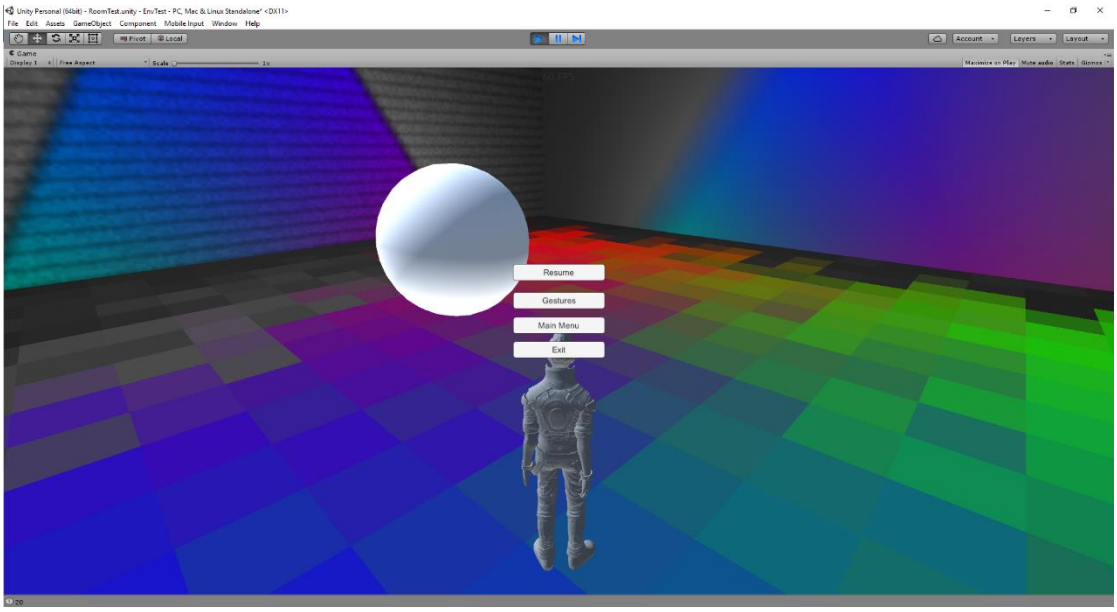


Figure 14: Pause Menu

There are 4 button in this Canvas:

- Resume Button Script: Set's the timescale back to 1 from freezing (0) and hides the canvas.
- Gestures Script: Enables a text that gives instructions to the user.
- Main Menu Script: Changes to Scene to "Menu". This was the issue that after pressing this button the game was stuck with loading image.
- Exit Script: Used the same string in the Main Menu's Exit button.

3.2.3.b Spawn Points

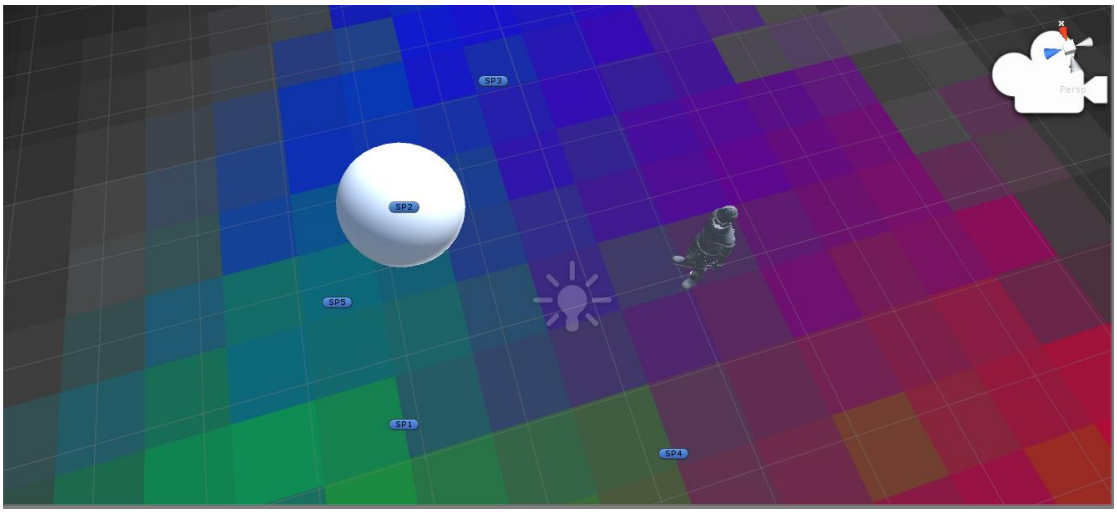


Figure 15: Spawn Points

As mentioned above, Spawn Points are empty objects that's been used for keeping a location or enabling a script. In this case, we have used both. There is a script enabled on "SP3" called "Random Spawn". This script is one of the most important scripts of this project because it allows to randomly spawn a sphere. It keeps an array called "Spawn Points" that allows us to have as many as spawn points as we want. After defining the spawn points inside of the array, it randomly chooses a spawn point within a certain time. In here, we have chosen 6 seconds to spawn a sphere. So, every 6 seconds a new sphere spawns from a different spawn point.

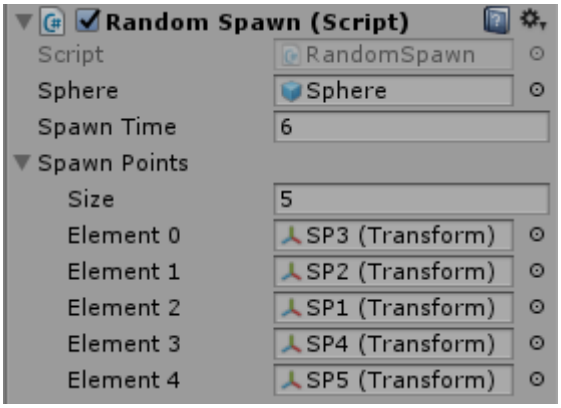


Figure 16: Script Settings

3.2.3.c Spheres(Balls)

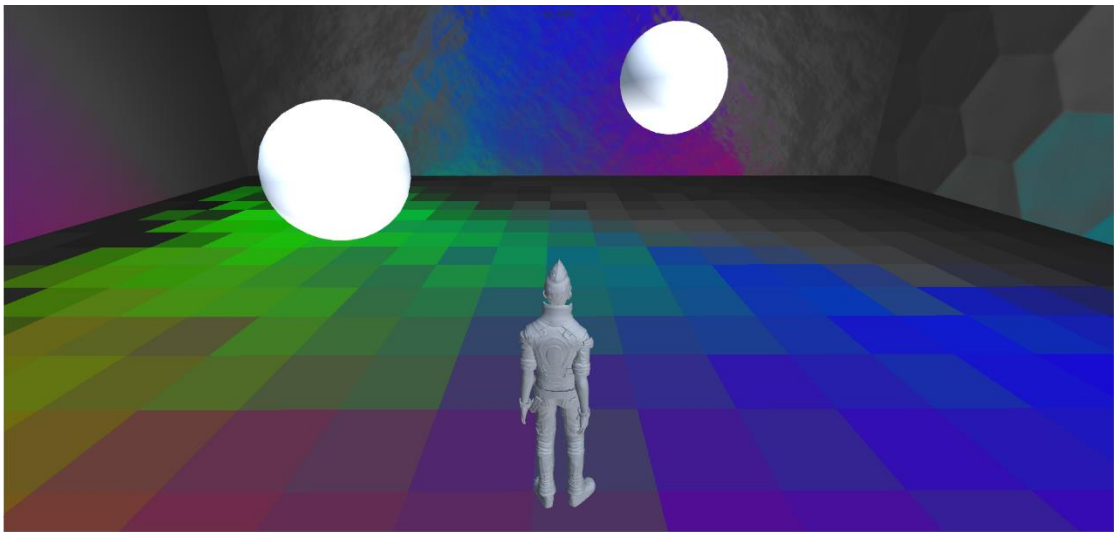


Figure 17: Game Screen, floating balls

The main idea for making the balls was that each ball had a different way of floating and user needed to catch the balls that was spawned. But, this is one of the most problematic part of the project. In the first demo, the character was passing through the balls and they weren't disappearing. For fixing that issue, there is a script called "Touch 2 Disappear". In Unity and many other game engines, there is a tagging system for making a game as a team. Each person that's working for this project can import their work with their tag and the objects that are tagged with that person's tag can be enabled easily and with this way it won't mix up with other people's work. Coming from this idea, player's controller "ThirdPersonController" is tagged as "Player" and "Touch 2 Disappear" script enables to disappear the object that's been enabled on the sphere when the objects that tagged as "Player" has any collision with that object. This is why there are colliders on Ethan's arms. The other issue was that when user touched any of the sphere's that's been spawned there was no other spheres spawned and the ones that was spawned already disappearing as well. To fix that issue, we have exported the sphere into a prefab object to make the scripts on one sphere work on others. Making it a prefab object also helped to use the spheres on the Main Menu as well.

To make the spheres float randomly in the air, there is a script called "Rnd Flt". The main idea was to be able to spawn multiple sphere's that are having different paths

at the same time. But, the fix that we have thought about scripts interfering each other didn't solve this issue. There was a temporary solution that, we have arranged the path of the spheres like a spring so, every time a sphere is spawned it will slightly go in a different way.

3.2.4 Gestures

Main idea for gestures is that when the user holds his arm between y and z axis' degrees for 5 seconds, objects will appear. As can be seen, with a script that holds a timer between those angles there will be a bat spawned [9]:

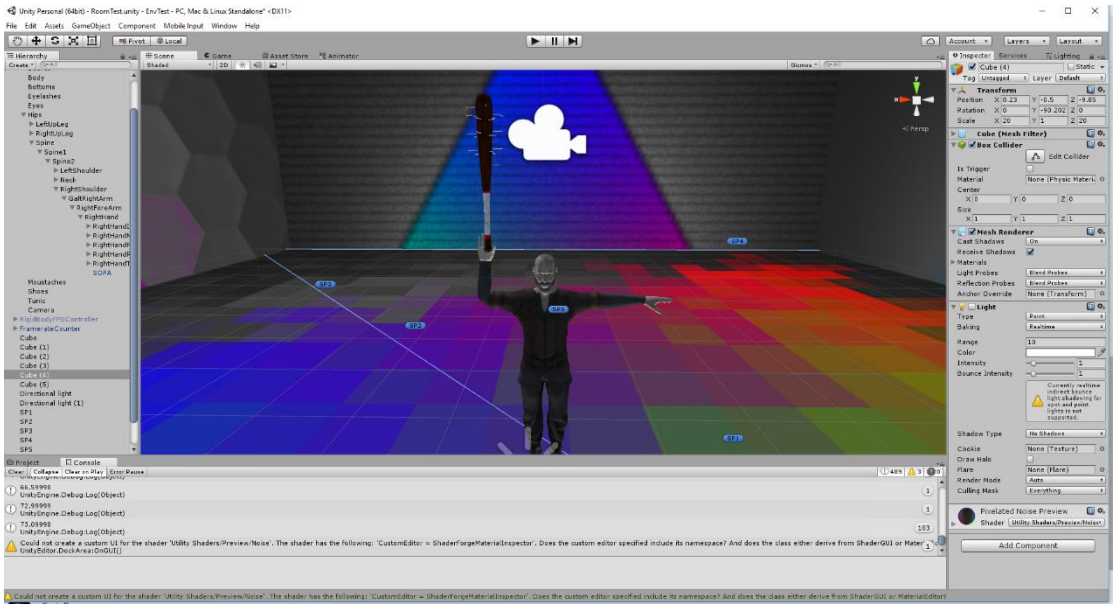


Figure 18: Unity Main Screen, showing bat.

3.2.5 Connecting Arduino to Unity

This is probably the most important part of this game project, why we have chosen Unity to make this game. It is because one of the two game engines that natively supports Arduino and serial port. In here, there has been many issues, to start with:

1-) Even that with the experiments in NodeJS were fine, there is serious lag issue with the values that Unity receives.

2-) 8 modules of Gyrons can only have around 2 frames per second in the game due to too many information for Unity's serial port to stream. And also sometimes there can be huge delays.

3-) To be able to use our device in Unity and move character's body parts, there was and still an issue for converting the 4 Dimensional data to quaternion and its spatial rotation and then to Euler's angles for rotating the character. We happen to believe that the issue is because Unity is not fast enough to convert this data at least 60*8 times per second.

Connecting Arduino to Unity, there has to be configuring done before using it.

1. Go on **Edit** | **Player Settings** to open the **Player Settings** in the inspector;
2. From **Optimization**, look for **API Compatibility Level** and select **.NET 2.0**.

Unity has to initialize the serial port in C# script, we need its port and its baud rate. Main library for using serial port in C# script is System.IO.Ports [10].

In our project, we have prepared our script and named it as "BodyMove". In this script, user can connect our device and use it to move our secondary character "Galtrilian". If the device isn't plugged in or ready for the game, user still can use "R" key button to move specific parts of Galtrilian. Also, there was an issue that, when using Arduino 101 unity was not able to read the data coming from serial port and Unity was timing out every time that Arduino 101 was sending data. After research, we found a solution by adding a code called DtrEnable. DTR means Data Terminal Ready and it solved the reading issue.

3.3 NodeJS Based Web Browser Application

3.3.1 Index.html

In the JavaScript Game, we have 3 files. One of them is index.html which includes scripts and webpage properties.

```
<html>
  <head>
    <title>Gyronome</title>
    <script src="https://cdn.socket.io/socket.io-1.2.0.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.3.16/p5.min.js"></script>
    <script type="text/javascript" src="client.js"></script>
  </head>
  <body>
  </body>
</html>
```

Figure 19: Index.html

First, we need to define this is a html file by using <html> tag. After that, in the <head> tag, we have a title field which is title of webpage and <script> tag is used to define script files. We use socket-io library to receive data from Arduino and client.js file which is our main file to create this game.

3.3.2 Server.js

In the server.js file, we write codes to create a local nodeJS server which provides us communication between an Arduino and a web browser.

```
// server initialization:
var express = require('express');           // include express.js
    io = require('socket.io'),              // include socket.io
    app = express(),                        // make an instance of express.js
    server = app.listen(8080),              // start a server with the express instance
    socketServer = io(server);              // make a socket server using the express server

// serial port initialization:
var serialport = require('serialport'),     // include the serialport library
    SerialPort  = serialport.SerialPort,    // make a local instance of serial
    portName = process.argv[2],             // get the port name from the command line
    portConfig = {
        baudRate: 38400,
        // call myPort.on('data') when a newline is received:
        parser: serialport.parsers.readline('\n')
    };
```

Figure 20: Server.js

In JavaScript, var keyword is used for defining a new variable and require keyword is used for import libraries. So, we import express library to create server,

socket.io library to receive gyrons data and serialport library to start a connection which is coming from serial ports. We create the localserver and listen 8080 port. We start this server by using terminal.

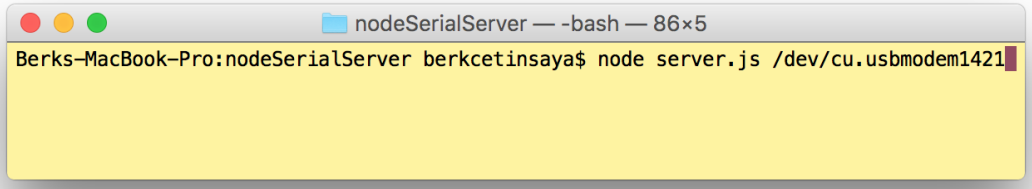


Figure 21: Starting NodeJS Server

As you see, third parameter of node command is port name and in the server.js file we define this third parameter as process.argv[2]. Our baud rate is static and it will be never changed so we define this as 38400.

The last part of server.js file is using for sending data.

```
26  function serveFiles(request, response) {
27      var fileName = request.params.name;     // get the file name from the request
28      response.sendFile(fileName);            // send the file
29  }
30
31  function openSocket(socket){
32      console.log('new user address: ' + socket.handshake.address);
33      // send something to the web client with the data:
34      socket.emit('message', 'Hello, ' + socket.handshake.address);
35
36      // this function runs if there's input from the client:
37      socket.on('message', function(data) {
38          myPort.write(data);                 // send the data to the serial device
39      });
40
41      // this function runs if there's input from the serialport:
42      myPort.on('data', function(data) {
43          socket.emit('message', data);       // send the data to the client
44      });
45  }
```

Figure 22: Server.js, sending data code

In the openSocket function, we open a socket connection and send the data to our serial device. When device takes this message, it sends data back to the serial port. And we print some debug messages to understand where the data is and what that is.

In the last part, we will give an explanation of our game logic. In the client.js, we draw the game. First, we define all variables we use.

```
var socket = io();          // socket.io instance. Connects back to the server

var g0z, g0y;
var g1z, g1y;
var g2z, g2y;
var g3z, g3y;
var g4z, g4y;
var g5z, g5y;
var g6z, g6y;
var g7z, g7y;

var pingpong = false;
var pingpongcounter = 0;

var sword = false;
var swordcounter = 0;

var ballflag = true;

var score = 0;

// var flag1 = false;
// var flag2 = false;   these flags are used to reset score " IMPORTANT !! "
// var flag3 = false;
// var flag4 = false;

/////// HEAD ///////
var hdx = 750;
var hdy = 175;
var hdw = 100;
var hdh = hdw;
/////// HEAD ///////

/////// NECK ///////
var nx1 = 750;
var ny1 = 225;
var nx2 = 750;
var ny2 = 250;
/////// NECK ///////

/////// BODY ///////
var bdx = 700;
var bdy = 250;
var bdw = 100;
var bdh = 200;
/////// BODY ///////

/////// LEFT ARM ///////
var rax1 = 810;
var ray1 = 260;
var rax2 = g7z; //g7z; //690 930;
var ray2 = g7y; //g7y; //140 380;
var rax3 = g6z; //g6z; //560 1060;
var ray3 = g6y; //g6y; //10 510;
/////// LEFT ARM ///////

/////// RIGHT ARM ///////
var lax1 = 690;
var lay1 = 260;
var lax2 = g0z; //g0z; //570 810
var lay2 = g0y; //g0y; //140 380
var lax3 = g1z; //g1z; //440 940
var lay3 = g1y; //g1y; //10 510
/////// RIGHT ARM ///////
```

Figure 23: Client.js, variables

At the first line, we define io() component as a socket. So, it will be ready when the Arduino send data. g*z and g*y is used for data. We have gestures. One of them is ping pong racket the other one is light saber.

```
66   //PING PONG//
67   var ptx1 = lax3;
68   var pty1 = lay3;
69   var ptx2 = lax3 - 40;
70   var pty2 = lay3 - 30;
71
72   var phx1 = lax3 - 65;
73   var phy1 = lay3 - 50;
74   var phw1 = 50;
75   var phh1 = 60;
76   //PING PONG//
77
78   /////// RIGHT LEG ///////
79   var llx1 = 725;
80   var lly1 = 460;
81   var llx2 = g2z; //g2z; 485 965
82   var lly2 = g2y; //g2y; 220 700
83   var llx3 = g3z; //g3z; 225 1225
84   var lly3 = g3y; //g3y; 10 860
85   /////// RIGHT LEG ///////
86
87   /////// LEFT LEG ///////
88   var rlx1 = 775;
89   var rly1 = 460;
90   var rlx2 = g5z; //g5z; 535 1015
91   var rly2 = g5y; //g5y; 220 700
92   var rlx3 = g4z; //g4z; 275 1275
93   var rly3 = g4y; //g4y; 10 860
94   /////// LEFT LEG ///////
```

Figure 24: Variables for ping pong racket

```
96   function setup() {
97     createCanvas(1500, 900);    // set up the canvas
98     if (ballflag)               // to stop game after score = 18
99       balls();
100  }
```

Figure 25: Setup Function

In the setup function, we create a canvas to draw shapes and call balls function to start game if ballflag is true which means score is less than 18.

We have a draw function. In draw function, all game components are called. It is infinite loop so; it will work until you close the connection.

```
function draw() {
  background(255, 204, 0);

  /////// HEAD ///////
  strokeWeight(5);
  fill(32,178,170);
  ellipse(hdx, hdy, hdw, hdh);
  /////// HEAD ///////

  /////// NECK ///////
  stroke(0);
  line(nx1,ny1,nx2,ny2);
  /////// NECK ///////

  /////// BODY ///////
  stroke(0);
  fill(34,139,34);
  rect(bdx,bdy,bdw,bdh);
  fill(0,0,255);
  rect(bdx+30,bdy+30,40,20);
  /////// BODY ///////


  // in order to draw every object again and again, redefine all variables in here

  ////// RIGHT ARM //////
  lax2 = g0z; //g0z; 570 810
  lay2 = g0y; //g0y; 140 380
  lax3 = g1z; //g1z; 440 940
  lay3 = g1y; //g1y; 10 510
  ////// RIGHT ARM //////


  ////// RIGHT LEG //////
  llx2 = g2z; //g2z; 485 965
  lly2 = g2y; //g2y; 220 700
  llx3 = g3z; //g3z; 225 1225
  lly3 = g3y; //g3y; 10 860
  ////// RIGHT LEG //////

  ////// LEFT LEG //////
  rlx3 = g4z; //g4z; 275 1275
  rly3 = g4y; //g4y; 10 860
  rlx2 = g5z; //g5z; 535 1015
  rly2 = g5y; //g5y; 220 700
  ////// LEFT LEG //////

  ////// LEFT ARM //////
  rax3 = g6z; //g6z; 690 930
  ray3 = g6y; //g6y; 140 380
  rax2 = g7z; //g7z; 560 1060
  ray2 = g7y; //g7y; 10 510
  ////// LEFT ARM //////
```

Figure 26: Draw fucntion

background(255,204,0) is used for coloring background to yellow. After that, we start to draw the user. First, we draw an ellipse as a head by using coordinates which are statically defined above. And we give a color by fill() function. strokeWeight() is used for line off site of head. After that, we draw neck by using line() function. This takes 4 parameters. line(x of start point, y of start point, x of end point, y of end point).

We use a rect() function which draws a rectangle to the screen. It takes also 4 parameters. Rect(x,y, width, height). Then, we reassign all gyrons data to their variables. Because, every millisecond they are changed therefore; we need to assign them again.

```
156     /////// PING PONG ///////
157     ptx1 = lax3;
158     pty1 = lay3;
159     ptx2 = lax3 - 40;
160     pty2 = lay3 - 30;
161     phx1 = lax3 - 65;
162     phy1 = lay3 - 50;
163     phw1 = 50;
164     phh1 = 60;
165     /////// PING PONG ///////
```

Figure 27: Ping pong racket variables

Also, ping pong and light saber gestures coordinates should be refreshed.

```
168   ////// LEFT ARM //////
169   noFill();
170   smooth();
171   stroke(102,0,204);
172   strokeWeight(15.0);
173   strokeJoin(ROUND);
174   beginShape();
175   vertex(rax1, ray1);
176   vertex(rax2, ray2);
177   vertex(rax3, ray3);
178   endShape();
179   ////// LEFT ARM //////
180
181   ////// RIGHT ARM //////
182   noFill();
183   smooth();
184   stroke(255,128,0);
185   strokeWeight(15.0);
186   strokeJoin(ROUND);
187   beginShape();
188   vertex(lax1, lay1);
189   vertex(lax2, lay2);
190   vertex(lax3, lay3);
191   endShape();
192   ////// RIGHT ARM //////
193
194   ////// RIGHT LEG //////
195   noFill();
196   smooth();
197   stroke(102,0,204);
198   strokeWeight(15.0);
199   strokeJoin(ROUND);
200   beginShape();
201   vertex(llx1, lly1);
202   vertex(llx2, lly2);
203   vertex(llx3, lly3);
204   endShape();
205   ////// RIGHT LEG //////
206
207   ///// LEFT LEG //////
208   noFill();
209   smooth();
210   stroke(255,128,0);
211   strokeWeight(15.0);
212   strokeJoin(ROUND);
213   beginShape();
214   vertex(rlx1, rly1);
215   vertex(rlx2, rly2);
216   vertex(rlx3, rly3);
217   endShape();
218   ////// LEFT LEG //////
```

Figure 28: Body parts variables

In order to draw all part of body, we use vertex() functions. vertex() function is similar to line. But it takes 2 parameters. So, we use two vertex function consecutively. First indicates start point, second vertex() indicates end point of first vertex().

```
248     if(rax2 > 870    && rax2 < 930    && ray2 > 230 && ray2 < 290 && rax3 > 870    && rax3 < 930    &&
          ray3 > 90 && ray3 < 150){
249       var myInterval1 = setInterval(function () {
250         ++pingpongcounter;
251       }, 3000);
252     }
253     else{
254       pingpongcounter = 0;
255       clearInterval(myInterval1);
256     }
257
258     if(pingpongcounter > 2){
259         pingpong = true;
260         sword = false;
261         pingpongcounter = 0;
262     }
263
264     if(pingpong){
265       noFill();
266       smooth();
267       stroke(167, 85, 2);
268       strokeWeight(15.0);
269       strokeJoin(ROUND);
270       beginShape();
271       vertex(ptx1, pty1);
272       vertex(ptx2, pty2);
273       endShape();
274       strokeWeight(4);
275       fill(235,50,50);
276       ellipse(phx1,phy1,phw1,phh1);
277     }
278     if(rax2 > 870    && rax2 < 930    && ray2 > 230 && ray2 < 290 && rax3 > 950    && rax3 < 1010    &&
          ray3 > 230 && ray3 < 290){
279       var myInterval2 = setInterval(function () {
280         ++swordcounter;
281       }, 3000);
282     }
283     else{
284       swordcounter = 0;
285       clearInterval(myInterval2);
286     }
287
288     if(swordcounter > 2){
289         sword = true;
290         pingpong = false;
291     }
292     if(sword){
293       noFill();
294       smooth();
295       stroke(0, 0, 0);
296       strokeWeight(25.0);
297       strokeJoin(ROUND);
298       beginShape();
299       vertex(ptx1, pty1);
300       vertex(ptx2, pty2);
301       endShape();
302       strokeWeight(15);
303       stroke(0, 0, 255);
304       line(ptx2,pty2,ptx2-150, pty2-130);
305     }
306     strokeWeight(4);
307     stroke(167, 85, 2);
308     fill(235,50,50);
309     sheqil();
310   }
312   var fruits = [[30, 30, 30, 30],[60, 60, 30, 30],[90, 90, 30, 30],[120, 120, 30, 30],[150, 150, 30,
        30],[180, 120, 30, 30],[210, 90, 30, 30],[240, 60, 30, 30],[270, 30, 30, 30],[30, 870, 30, 30],[60
        , 840, 30, 30],[90, 810, 30, 30],[120, 780, 30, 30],[150, 750, 30, 30],[180, 780, 30, 30],[210,
        810, 30, 30],[240, 840, 30, 30],[270, 870, 30, 30]];
313   function sheqil(){
314     for (var i = 0; i < score; i++) {
315       ellipse(fruits[i][0],fruits[i][1],fruits[i][2],fruits[i][3]);
316     }
```

Figure 29: : Gesture statements

In gesture part, we have two if statements. First one is for Ping-Pong racket. If the coordinates are between following coordinates, it will increase ping pong counter every three second, after three second Boolean Ping-Pong will be true and sword will be false also counter will be reset. If Ping-Pong true, it will draw a racket to the user's

right hand. Light saber is same but just coordinates are different. sheqil() function is used to draw score as a little circle.

In the readData() function, we split output of Arduino on the commas. First part of output is gyron id, second is z and the last part is y. So, we assign all of them to their variables. But the important part is calibration. All if statements in this function is necessary to calibrate all sensors.

```
function readData (data) {
  var results = data.split(',');  // split the data on the commas
  if (results[0] == "0"){
    if(results[1] > 570 && results[1] < 930)
      g0z = results[1];                 // x is the first value
    if(results[2] > 140 && results[2] < 380 )
      g0y = results[2];                 // y is the second value
  }
  if (results[0] == "1"){            // button is the third value
    if(results[1] > 440 && results[1] < 940)
      g1z = results[1];                 // x is the first value
    if(results[2] > 10 && results[2] < 510)
      g1y = results[2];                 // y is the second value
  }
  if (results[0] == "2"){            // button is the third value
    if(results[1] > 485 && results[1] < 965 )
      g2z = results[1];                 // x is the first value
    if(results[2] > 460 && results[2] < 700)
      g2y = results[2];                 // y is the second value
  }
  if (results[0] == "3"){            // button is the third value
    if(results[1] > 225 && results[1] < 1225)
      g3z = results[1];                 // x is the first value
    if(results[2] > 460 && results[2] < 860)
      g3y = results[2];                 // y is the second value
  }
  if (results[0] == "4"){            // button is the third value
    if(results[1] > 275 && results[1] < 1275)
      g4z = results[1];                 // x is the first value
    if(results[2] > 460 && results[2] < 860)
      g4y = results[2];                 // y is the second value
  }
  if (results[0] == "5"){            // button is the third value
    if(results[1] > 535 && results[1] < 1015)
      g5z = results[1];                 // x is the first value
    if(results[2] > 460 && results[2] < 700)
      g5y = results[2];                 // y is the second value
  }
  if (results[0] == "7"){            // button is the third value
    if(results[1] > 690 && results[1] < 930)
      g7z = results[1];                 // x is the first value
    if(results[2] > 140 && results[2] < 380)
      g7y = results[2];                 // y is the second value
  }
  if (results[0] == "6"){            // button is the third value
    if(results[1] > 560 && results[1] < 1060)
      g6z = results[1];                 // x is the first value
    if(results[2] > 10 &&results[2] < 510)
      g6y = results[2];                 // y is the second value
  }
   else{}
}
// when new data comes in the websocket, read it:
socket.on('message', readData);
```

Figure 30: : Read data function

```
function balls(){

  var x = 200; //initial position
  var y = 200;

  fill(255,0,0); //color
  var r = 40;

  var dx = 0;
  var dy = 0;
  var delta = 5; // range (from 0) of possible dx or dy change
  var max = 15;  // maximum dx or dy values

  var delay=3000; //1 second

//addEventListener("click", togglestart);
//function togglestart() {
//    if (interval == undefined) interval = window.setInterval(animate, 120 / 60);
//    else {
//        interval = clearInterval(interval);
//        console.log(interval);
//    }
//}

  var interval = window.setInterval(animate, 120 / 60);

  function animate() {
      var d2x = (Math.random() * delta - delta / 2); //change dx and dy by random value
      var d2y = (Math.random() * delta - delta / 2);

      if (Math.abs(d2x + dx) > max) // start slowing down if going too fast
      d2x *= -1;
      if (Math.abs(d2y + dy) > max) d2y *= -1;

      dx += d2x;
      dy += d2y;

      if(score < 18){
        if(pingpong){
          if(x - r < phx1 && y - r < phy1 && y + r > phy1 && x + r > phx1 ){
            score++;
            if (interval == undefined)
              interval = window.setInterval(animate, 120 / 60);
            else {
              interval = clearInterval(interval);
              console.log(interval);
            }
            setTimeout(function() {
              if (interval == undefined)
                interval = window.setInterval(animate, 120 / 60);
              else {
                interval = clearInterval(interval);
                console.log(interval);
              }
              x = 200;
              y = 200;
            }, delay);
          }
        }
```

Figure 31: Balls() Function

In balls() function, we define x and y coordinates for initial position of ball. Then give a color which is red for now. dx and dy are speeds and delta is the change of dx or dy. Interval is used for timer. Ball moves randomly and its speed is changeable. When it reaches 15 it is reset and start over to accelerate. We test that ping pong racket is on or light saber is on or nothing is on. The result of this testing its collision is changed. If there is nothing on hand, it hits to hand; if there is exist a Ping-Pong racket, it hits racket's head; if there is a light saber, it hits to light saber's head. If score is greater than 18, the game is over and a popup appears which has a text Game Over. After click close button on this popup, game is stopped and ball will be not created again

4 Results

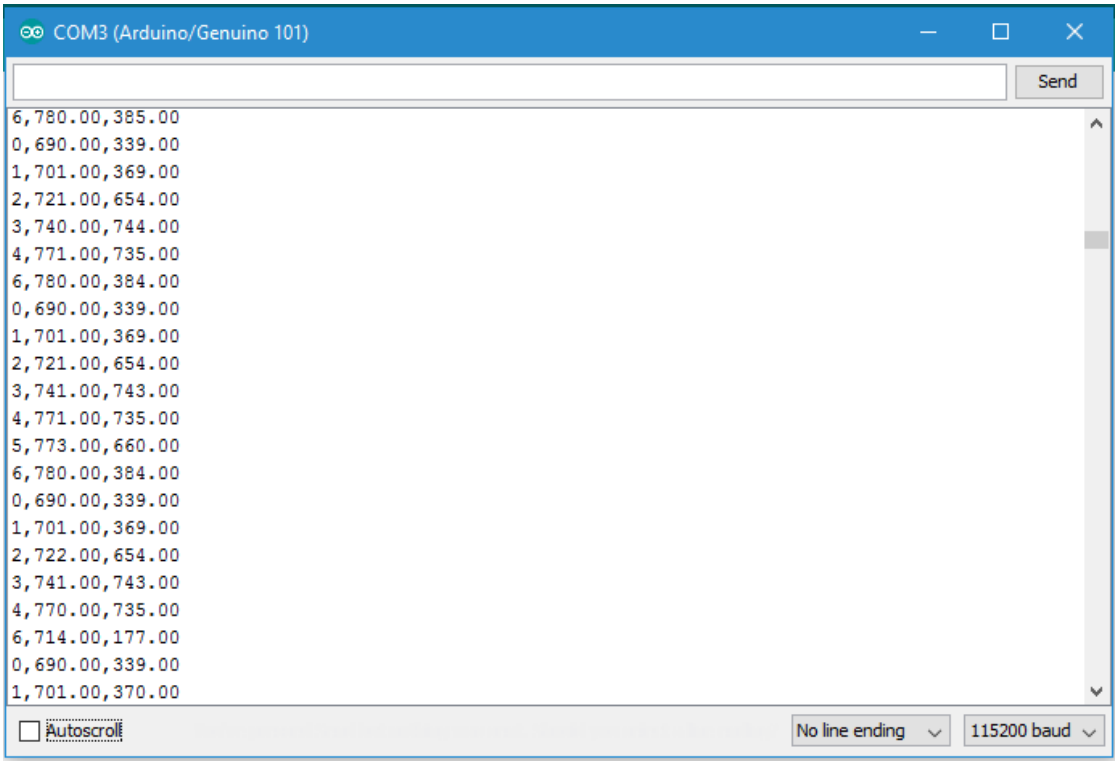


Figure 32: Arduino Serial Monitor

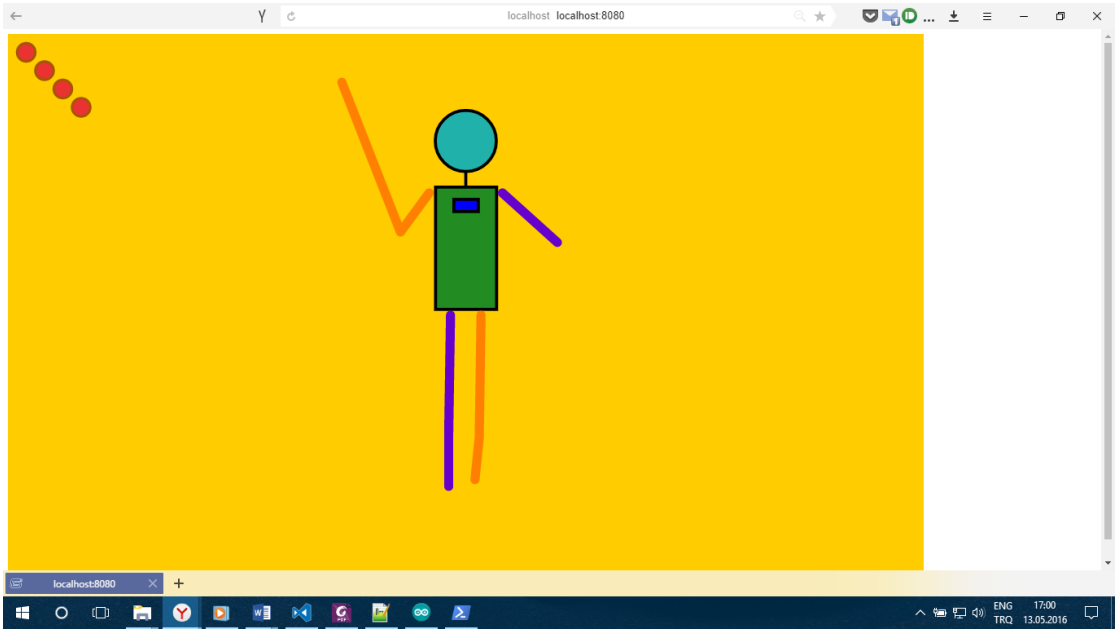


Figure 33: Web Browser Game Screen

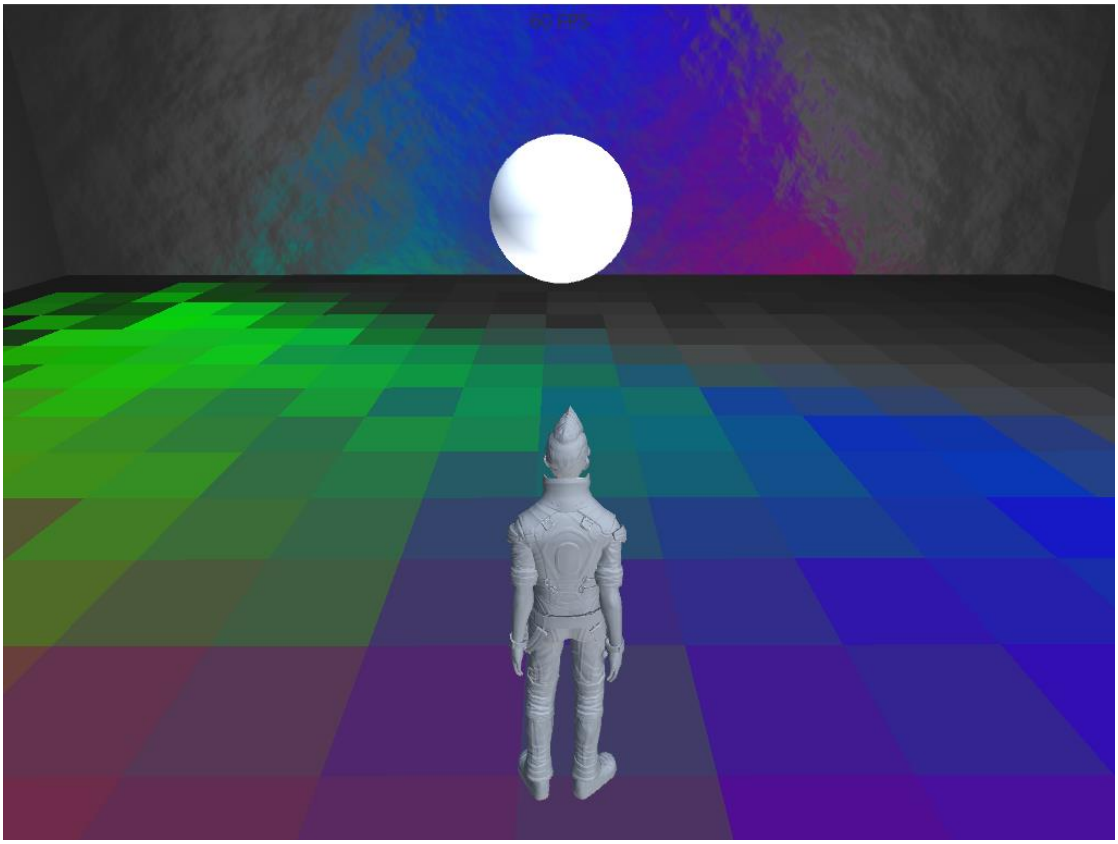


Figure 34: Unity Game Screen

5 Conclusion

In conclusion, our body movements can be shown in a web browser. In figure 30, as a result of Arduino code of Gyronome, we take all data separately;

Gyro0, x, y
Gyro1, x, y
Gyro2, x, y
Gyro3, x, y
Gyro4, x, y
Gyro5, x, y
Gyro6, x, y
Gyro7, x, y
Gyro0, x, y
Gyro1, x, y
.
.
.
.
.
.

In figure 31, all data are mapped and all gyrons are calibrated. readData() function is worked and all values assign to appropriate variables. Player tries to catch the ball which is moving randomly. When score reaches 18, game is over.

In figure 32, this game integrated to unity game engine and user controls player by w,a,s,d keys and tries to catch ball which spawns randomly.

Future works include a virtual reality game in Unity. For now, gyrons are connected to arduino by cables. When it is ready, cables will be gone and wireless transmitters will be replaced instead of them. In this way, there will be no cable mass

and it will work efficiently. Also, we intend to give haptic feedbacks to user in order to give a perfect game experience by using vibration and pressure sensors.

References

[1] E. Bachmann, “Inertial and magnetic tracking of limb segment orientation for inserting humans into synthetic environments,” PhD Thesis, Naval Postgraduate School, 2000.

[2] H. Zhoua, T. Stoneb, H. Huc, N. Harrisb, “Use of multiple wearable inertial sensors in upper limb motion tracking”, 2006.

[3] Novi Commentarii academiae scientiarum Petropolitanae 20, 1776, pp. 189–207

[4] Karsten Kunze, Helmut Schaeben (November 2004). "The Bingham Distribution of Quaternions and Its Spherical Radon Transform in Texture Analysis". Mathematical Geology 8: 917–943.

[5] https://github.com/jrowberg/i2cdevlib

[6] https://store.invensense.com/datasheets/invensense/Sensor-Introduction.pdf

[7] http://docs.unity3d.com/Manual/index.html

[8] u3d.as/ah5

[9] u3d.as/gff

[10] http://www.alanzucconi.com/2015/10/07/how-to-integrate-arduino-with-unity/

[11] http://43zrtwysvxb2gf29r5o0athu.wpengine.netdna-cdn.com/wp-conntent/uploads/2015/02/MPU-6000-Datasheet1.pdf

https://forum.arduino.cc/index.php?topic=387196.0

http://answers.unity3d.com/questions/350721/c-yield-waitforseconds.html

6 Appendix A

MPU-6050 Technical Specifications [11]

The MPU-60X0 is the world's first integrated 6-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor™ (DMP) all in a small 4x4x0.9mm package. With its dedicated I 2C sensor bus, it directly accepts inputs from an external 3-axis compass to provide a complete 9-axis MotionFusion™ output.

Gyroscope Features

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable fullscale range of ±250, ±500, ±1000, and ±2000°/sec
- External sync signal connected to the FSYNC pin supports image, video and GPS synchronization
- Integrated 16-bit ADCs enable simultaneous sampling of gyros
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Improved low-frequency noise performance
- Digitally-programmable low-pass filter
- Gyroscope operating current: 3.6mA
- Standby current: 5µA
- Factory calibrated sensitivity scale factor
- User self-test

Accelerometer Features

- Digital-output triple-axis accelerometer with a programmable full scale range of ±2g, ±4g, ±8g and ±16g

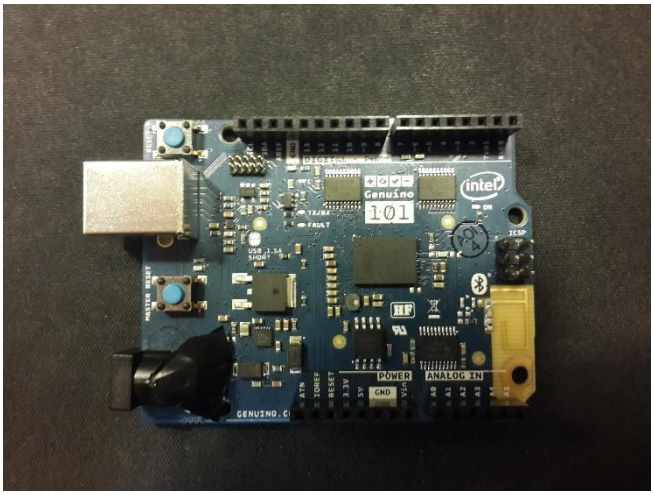
- Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer
- Accelerometer normal operating current: 500µA
- Low power accelerometer mode current: 10µA at 1.25Hz, 20µA at 5Hz, 60µA at 20Hz, 110µA at 40Hz
- Orientation detection and signaling
- Tap detection
- User-programmable interrupts
- High-G interrupt
- User self-test

Additional Features

- 9-Axis MotionFusion by the on-chip Digital Motion Processor (DMP)
- Auxiliary master I 2C bus for reading data from external sensors (e.g., magnetometer)
- 3.9mA operating current when all 6 motion sensing axes and the DMP are enabled
- VDD supply voltage range of 2.375V-3.46V
- Flexible VLOGIC reference voltage supports multiple I2C interface voltages (MPU-6050 only)
- Smallest and thinnest QFN package for portable devices: 4x4x0.9mm
- Minimal cross-axis sensitivity between the accelerometer and gyroscope axes
- 1024 byte FIFO buffer reduces power consumption by allowing host processor to read the data in bursts and then go into a low-power mode as the MPU collects more data
- Digital-output temperature sensor

- User-programmable digital filters for gyroscope, accelerometer, and temp sensor
- 10,000 g shock tolerant
- 400kHz Fast Mode I 2C for communicating with all registers

Arduino 101 Technical Specifications



Microcontroller: Intel Curie

Operating Voltage: 3.3V (5V tolerant I/O)

Input Voltage (recommended): 7-12V

Input Voltage (limit): 7-20V

Digital I/O Pins: 14 (of which 4 provide PWM output)

PWM Digital I/O Pins: 4

Analog Input Pins: 6

DC Current per I/O Pin: 20 mA

Flash Memory: 196 kB

SRAM: 24 kB

Clock Speed: 32MHz

Features: Bluetooth LE, 6-axis accelerometer/gyro

Length: 68.6 mm

Width: 53.4 mm

NodeJS Application Source Code

Server.js

```
// server initialization:
var express = require('express');               // include express.js
io = require('socket.io'),                      // include socket.io
app = express(),                                      // make an instance of
express.js
server = app.listen(8080),                      // start a server with the
express instance
socketServer = io(server);                      // make a socket server using
the express server

// serial port initialization:
var serialport = require('serialport'),         // include the serialport
library
      SerialPort = serialport.SerialPort,       // make a local instance of
serial
      portName = process.argv[2],                     // get the port name from
the command line
      portConfig = {
          baudRate: 38400,
          // call myPort.on('data') when a newline is received:
          parser: serialport.parsers.readline('\n')
      };

// open the serial port:
var myPort = new SerialPort(portName, portConfig);

//  set up server and socketServer listener functions:
app.use(express.static('public'));                    // serve files from the
public folder
app.get('/:name', serveFiles);                        // listener for all
static file requests
socketServer.on('connection', openSocket);      // listener for websocket data

function serveFiles(request, response) {
    var fileName = request.params.name;         // get the file name from the
request
    response.sendFile(fileName);                // send the file
}

function openSocket(socket) {
    console.log('new user address: ' + socket.handshake.address);
    // send something to the web client with the data:
    socket.emit('message', 'Hello, ' + socket.handshake.address);

    // this function runs if there's input from the client:
    socket.on('message', function (data) {
        myPort.write(data);                           // send the data to the
serial device
    });

    // this function runs if there's input from the serialport:
```

```
    myPort.on('data', function (data) {
        socket.emit('message', data);          // send the data to the client
    });
}
```

Client.js

```
var socket = io();               // socket.io instance. Connects back to the
server

var g0z, g0y;
var g1z, g1y;
var g2z, g2y;
var g3z, g3y;
var g4z, g4y;
var g5z, g5y;
var g6z, g6y;
var g7z, g7y;


var padding = 0;
var paddingcounter = 0;
var paddingstate = true;

var pingpong = false;
var pingpongcounter = 0;

var sword = false;
var swordcounter = 0;

var ballflag = true;

var score = 0;

var flag1 = false;
var flag2 = false;
var flag3 = false;
var flag4 = false;

/////// HEAD ///////
var hdx = 750 + padding;
var hdy = 175;
var hdw = 100;
var hdh = hdw;
/////// HEAD ///////

/////// NECK ///////
var nx1 = 750 + padding;
var ny1 = 225;
var nx2 = 750 + padding;
var ny2 = 250;
/////// NECK ///////

/////// BODY ///////
var bdx = 700 + padding;
var bdy = 250;
var bdw = 100;
var bdh = 200;
/////// BODY ///////
```

```
/////// LEFT ARM ///////
var rax1 = 810 + padding;
var ray1 = 260;
var rax2 = g7z + padding;//g7z; //690 930; pif(rax2 > 870 + padding && rax2 <
930 + padding && ray2 > 230 && ray2 < 290 && rax3 > 900 + padding && rax3 < 960
+ padding && ray3 > 90 && ray3 < 120
var ray2 = g7y;//g7y; //140 380;
var rax3 = g6z + padding;//g6z; //560 1060; sif(rax2 > 870 + padding && rax2 <
930 + padding && ray2 > 230 && ray2 < 290 && rax3 > 950 + padding && rax3 <
1010 + padding && ray3 > 230 && ray3 < 290)
var ray3 = g6y;//g6y;  //10 510;
/////// LEFT ARM ///////

/////// RIGHT ARM ///////
var lax1 = 690 + padding;
var lay1 = 260;
var lax2 = g0z + padding;//g0z; //570 810
var lay2 = g0y;//g0y; //140 380
var lax3 = g1z + padding;//g1z; //440 940
var lay3 = g1y;//g1y; //10 510
/////// RIGHT ARM ///////

//PING PONG//
var ptx1 = lax3;
var pty1 = lay3;
var ptx2 = lax3 - 40;
var pty2 = lay3 - 30;

var phx1 = lax3 - 65;
var phy1 = lay3 - 50;
var phw1 = 50;
var phh1 = 60;
//PING PONG//

/////// RIGHT LEG ///////
var llx1 = 725 + padding;
var lly1 = 460;
var llx2 = g2z + padding; //g2z; 485 965      if(llx2 > 670 + padding && llx2 <
700 + padding && lly2 > 570 && lly2 < 600){
var lly2 = g2y; //220 700
var llx3 = g3z + padding;//g3z 225 1225
var lly3 = g3y; //10 860
/////// RIGHT LEG ///////

/////// LEFT LEG ///////
var rlx1 = 775 + padding;
var rly1 = 460;
var rlx2 = g5z + padding; //g5z 535 1015      if(rlx2 > 800 + padding && rlx2 <
830 + padding && rly2 > 570 && rly2 < 600){
var rly2 = g5y; //g5y 220 700
var rlx3 = g4z + padding; //g4 275 1275
var rly3 = g4y; //10 860
/////// LEFT LEG ///////

function setup() {
  createCanvas(1500, 900);   // set up the canvas
  if (ballflag)
    balls();
}

function draw() {
```

```
  background(255, 204, 0);        // make the screen white
  var fillColor = 230;      // set the fill color to black


  strokeWeight(5);
  fill(32,178,170);
  ellipse(hdx, hdy, hdw, hdh);    //head

  stroke(0);
  line(nx1,ny1,nx2,ny2);          //neck

  stroke(0);
  fill(34,139,34);
  rect(bdx,bdy,bdw,bdh);          //body
  fill(0,0,255);
  rect(bdx+30,bdy+20,40,20);

////// RIGHT ARM //////
lax2 = g0z + padding;//g0z; 570 810
lay2 = g0y;//g0y; 140 380
lax3 = g1z + padding;//g1z; 440 940
lay3 = g1y;//g1y; 10 510
////// RIGHT ARM //////


////// RIGHT LEG //////
llx2 = g2z + padding;//g2z; 485 965
lly2 = g2y;//g2y; 220 700
llx3 = g3z + padding;//g3z; 225 1225
lly3 = g3y;//g3y; 10 860
////// RIGHT LEG //////

////// LEFT LEG //////
rlx3 = g4z + padding; //g4z 275 1275
rly3 = g4y;// + padding; //10 860
rlx2 = g5z + padding; //g5z 535 1015
rly2 = g5y;// + padding; //g5y 220 700
////// LEFT LEG //////

////// LEFT ARM //////
rax3 = g6z + padding;//g6z; 690 930;
ray3 = g6y;// + padding;//g6y; 140 380;
rax2 = g7z + padding;//g7z; 560 1060;
ray2 = g7y;// + padding;//g7y; 10 510;
////// LEFT ARM //////

ptx1 = lax3;
pty1 = lay3;
ptx2 = lax3 - 40;
pty2 = lay3 - 30;

phx1 = lax3 - 65;
phy1 = lay3 - 50;
phw1 = 50;
phh1 = 60;

padding = padding;

////// LEFT ARM //////
```

```
  noFill();
  smooth();
  stroke(102,0,204);
  strokeWeight(15.0);
  strokeJoin(ROUND);
  beginShape();
  vertex(rax1, ray1);
  vertex(rax2, ray2);
  vertex(rax3, ray3);
  endShape();
////// LEFT ARM //////

////// RIGHT ARM //////
  noFill();
  smooth();
  stroke(255,128,0);
  strokeWeight(15.0);
  strokeJoin(ROUND);
  beginShape();
  vertex(lax1, lay1);
  vertex(lax2, lay2);
  vertex(lax3, lay3);
  endShape();
////// RIGHT ARM //////

////// RIGHT LEG //////
  noFill();
  smooth();
  stroke(102,0,204);
  strokeWeight(15.0);
  strokeJoin(ROUND);
  beginShape();
  vertex(llx1, lly1);
  vertex(llx2, lly2);
  vertex(llx3, lly3);
  endShape();
////// RIGHT LEG //////

///// LEFT LEG //////
  noFill();
  smooth();
  stroke(255,128,0);
  strokeWeight(15.0);
  strokeJoin(ROUND);
  beginShape();
  vertex(rlx1, rly1);
  vertex(rlx2, rly2);
  vertex(rlx3, rly3);
  endShape();
////// LEFT LEG //////

  // noFill();
  // smooth();
  // stroke(255,255,255);
  // strokeWeight(15.0);
  // strokeJoin(ROUND);
  // beginShape();
  // vertex(800 + padding, 570);
  // vertex(830 + padding, 600);
  // endShape();
```

```
  if(rlx2 > 800 + padding && rlx2 < 830 + padding && rly2 > 570 && rly2 < 600){
    var myInterval2 = setInterval(function () {
    ++paddingcounter;
    console.log(paddingcounter);
    }, 3000);
  }

    else{
      paddingcounter = 0;
      clearInterval(myInterval2);
    }

    if(paddingcounter > 2){
      padding = padding + 30;
      paddingcounter = 0;
    }
//console.log("rax2 = " + rax2 + "ray2 = " + ray2 + "lax3 = " + lax3 + "lay3 =
" + lay3);
    if(llx2 > 670 + padding && llx2 < 700 + padding && lly2 > 570 && lly2 <
600){
    var myInterval3 = setInterval(function () {
    ++paddingcounter;
    console.log(paddingcounter);
    }, 3000);
  }

    else{
      paddingcounter = 0;
      clearInterval(myInterval3);
    }

    if(paddingcounter > 2){
      padding = padding - 30;
      paddingcounter = 0;
    }

  if(lax2 > 660 + padding && lax2 < 720 + padding && lay2 > 230 && lay2 < 290
&& lax3 > 660 + padding && lax3 < 690 + padding && lay3 > 270 && lay3 < 300 &&
flag1 == false && flag2 == false && flag3 == false && flag4 == false){
    flag1 = true;
    flag2 = false;
    flag3 = false;
    flag4 = false;
  }
  if(lax2 > 660 + padding && lax2 < 720 + padding && lay2 > 230 && lay2 < 290
&& lax3 > 690 + padding && lax3 < 720 + padding && lay3 > 300 && lay3 < 330 &&
flag1 == true && flag2 == false && flag3 == false && flag4 == false){
    flag1 = true;
    flag2 = true;
    flag3 = false;
    flag4 = false;
  }
  if(lax2 > 660 + padding && lax2 < 720 + padding && lay2 > 230 && lay2 < 290
&& lax3 > 720 + padding && lax3 < 750 + padding && lay3 > 330 && lay3 < 360 &&
flag1 == true && flag2 == true && flag3 == false && flag4 == false){
    flag1 = true;
    flag2 = true;
    flag3 = true;
    flag4 = false;
  }
```

```
  if(lax2 > 660 + padding && lax2 < 720 + padding && lay2 > 230 && lay2 < 290
&& lax3 > 750 + padding && lax3 < 780 + padding && lay3 > 360 && lay3 < 390 &&
flag1 == true && flag2 == true && flag3 == true && flag4 == false){
    flag1 = true;
    flag2 = true;
    flag3 = true;
    flag4 = true;
  }
  if(flag1 && flag2 && flag3 && flag4){
    score = 0;
  }
//console.log("lax2 = " + rax2 + "lay2 = " + ray2 + "lax3 = " + rax3 + "lay3 =
" + ray3);

  if(rax2 > 870 + padding && rax2 < 930 + padding && ray2 > 230 && ray2 < 290
&& rax3 > 870 + padding && rax3 < 930 + padding && ray3 > 90 && ray3 < 150){
    var myInterval1 = setInterval(function () {
    ++pingpongcounter;
    console.log(pingpongcounter);
    }, 3000);
  }
  else{
    pingpongcounter = 0;
    clearInterval(myInterval1);
  }

  if(pingpongcounter > 2){
      pingpong = true;
      sword = false;
      pingpongcounter = 0;
  }
  if(pingpong){
    noFill();
    smooth();
    stroke(167, 85, 2);
    strokeWeight(15.0);
    strokeJoin(ROUND);
    beginShape();
    vertex(ptx1, pty1);
    vertex(ptx2, pty2);
    endShape();
    strokeWeight(4);
    fill(235,50,50);
    ellipse(phx1,phy1,phw1,phh1);
  }
if(rax2 > 870 + padding && rax2 < 930 + padding && ray2 > 230 && ray2 < 290 &&
rax3 > 950 + padding && rax3 < 1010 + padding && ray3 > 230 && ray3 < 290){
    var myInterval4 = setInterval(function () {
    ++swordcounter;
    console.log(swordcounter);
    }, 3000);
  }
  else{
    swordcounter = 0;
    clearInterval(myInterval4);
  }

  if(swordcounter > 2){
      sword = true;
      pingpong = false;
  }
```

```
  if(sword){
    noFill();
    smooth();
    stroke(0, 0, 0);
    strokeWeight(25.0);
    strokeJoin(ROUND);
    beginShape();
    vertex(ptx1, pty1);
    vertex(ptx2, pty2);
    endShape();

    strokeWeight(15);
    stroke(0, 0, 255);
    line(ptx2,pty2,ptx2-150, pty2-130);
  }

  strokeWeight(4);
  stroke(167, 85, 2);
  fill(235,50,50);
  sheqil();

}

var fruits = [[30, 30, 30, 30],[60, 60, 30, 30],[90, 90, 30, 30],[120, 120, 30,
30],[150, 150, 30, 30],[180, 120, 30, 30],[210, 90, 30, 30],[240, 60, 30,
30],[270, 30, 30, 30],[30, 870, 30, 30],[60, 840, 30, 30],[90, 810, 30,
30],[120, 780, 30, 30],[150, 750, 30, 30],[180, 780, 30, 30],[210, 810, 30,
30],[240, 840, 30, 30],[270, 870, 30, 30]];
function sheqil(){
  for (var i = 0; i < score; i++) {
    ellipse(fruits[i][0],fruits[i][1],fruits[i][2],fruits[i][3]);
  }

  // ellipse(30, 30, 30, 30);    // draw the ellipse
  // ellipse(60, 60, 30, 30);    // draw the ellipse
  // ellipse(90, 90, 30, 30);    // draw the ellipse
  // ellipse(120, 120, 30, 30);    // draw the ellipse
  // ellipse(150, 150, 30, 30);    // draw the ellipse     ////////  top left
  // ellipse(180, 120, 30, 30);    // draw the ellipse
  // ellipse(210, 90, 30, 30);    // draw the ellipse
  // ellipse(240, 60, 30, 30);    // draw the ellipse
  // ellipse(270, 30, 30, 30);    // draw the ellipse

  // ellipse(30, 870, 30, 30);    // draw the ellipse
  // ellipse(60, 840, 30, 30);    // draw the ellipse
  // ellipse(90, 810, 30, 30);    // draw the ellipse
  // ellipse(120, 780, 30, 30);    // draw the ellipse
  // ellipse(150, 750, 30, 30);    // draw the ellipse     ///////// bottom
left
  // ellipse(180, 780, 30, 30);    // draw the ellipse
  // ellipse(210, 810, 30, 30);    // draw the ellipse
  // ellipse(240, 840, 30, 30);    // draw the ellipse
  // ellipse(270, 870, 30, 30);    // draw the ellipse

  // ellipse(1470, 30, 30, 30);    // draw the ellipse
  // ellipse(1440, 60, 30, 30);    // draw the ellipse
  // ellipse(1410, 90, 30, 30);    // draw the ellipse
  // ellipse(1380, 120, 30, 30);    // draw the ellipse
  // ellipse(1350, 150, 30, 30);    // draw the ellipse    ///////// top right
  // ellipse(1320, 120, 30, 30);    // draw the ellipse
  // ellipse(1290, 90, 30, 30);    // draw the ellipse
```

```
  // ellipse(1260, 60, 30, 30);    // draw the ellipse
  // ellipse(1230, 30, 30, 30);    // draw the ellipse

  // ellipse(1470, 870, 30, 30);    // draw the ellipse
  // ellipse(1440, 840, 30, 30);    // draw the ellipse
  // ellipse(1410, 810, 30, 30);    // draw the ellipse
  // ellipse(1380, 780, 30, 30);    // draw the ellipse
  // ellipse(1350, 750, 30, 30);    // draw the ellipse     ///////// bottom
right
  // ellipse(1320, 780, 30, 30);    // draw the ellipse
  // ellipse(1290, 810, 30, 30);    // draw the ellipse
  // ellipse(1260, 840, 30, 30);    // draw the ellipse
  // ellipse(1230, 870, 30, 30);    // draw the ellipse
}
function balls(){

var x = 200;// + padding; //initial position
var y = 200;

fill(255,0,0); //color
var r = 40;

var dx = 0;
var dy = 0;
var delta = 5; // range (from 0) of possible dx or dy change
var max = 15; // maximum dx or dy values
//addEventListener("click", togglestart);
var delay=3000; //1 second


//function togglestart() {
//    if (interval == undefined) interval = window.setInterval(animate, 120 /
60); // 60 FPS
//    else {
//        interval = clearInterval(interval);
//        console.log(interval);
//    }
//}

  var interval = window.setInterval(animate, 240 / 60);


  function animate() {
      var d2x = (Math.random() * delta - delta / 2); //change dx and dy by
random value
      var d2y = (Math.random() * delta - delta / 2);

      if (Math.abs(d2x + dx) > max) // start slowing down if going too fast
      d2x *= -1;
      if (Math.abs(d2y + dy) > max) d2y *= -1;

      dx += d2x;
      dy += d2y;

      if(score < 18){
        if(pingpong){
          if(x - r < phx1 && y - r < phy1 && y + r > phy1 && x + r > phx1 ){
          score++;
          if (interval == undefined)
            interval = window.setInterval(animate, 240 / 60); // 60 FPS
          else {
```

```
            interval = clearInterval(interval);
            console.log(interval);
          }
            setTimeout(function() {
              if (interval == undefined)
                interval = window.setInterval(animate, 240 / 60); // 60 FPS
              else {
                interval = clearInterval(interval);
                console.log(interval);
              }
              x = 200 + padding;
              y = 200;
            }, delay);

          }
        }
        else if(sword){
        if(x - r < ptx2-150 && y - r < pty2-130 && y + r > pty2-130 && x + r >
ptx2-150 ){
          score++;
          if (interval == undefined)
            interval = window.setInterval(animate, 240 / 60); // 60 FPS
          else {
            interval = clearInterval(interval);
            console.log(interval);
          }
          setTimeout(function() {
            if (interval == undefined)
              interval = window.setInterval(animate, 240 / 60); // 60 FPS
            else {
              interval = clearInterval(interval);
              console.log(interval);
            }
            x = 200 + padding;
            y = 200;
          }, delay);

        }
      }
      else{
        if(x - r < lax3 && y - r < lay3 && y + r > lay3 && x + r > lax3 ){
          score++;
          if (interval == undefined)
            interval = window.setInterval(animate, 240 / 60); // 60 FPS
          else {
            interval = clearInterval(interval);
            console.log(interval);
          }
          setTimeout(function() {
            if (interval == undefined)
              interval = window.setInterval(animate, 240 / 60); // 60 FPS
            else {
              interval = clearInterval(interval);
              console.log(interval);
            }
            x = 200 + padding;
            y = 200;
          }, delay);

        }
      }
```

```
    }
    else
    {
      strokeWeight(4);
      stroke(167, 85, 2);
      fill(235,50,50);
      sheqil();
      interval = clearInterval(interval);
      ballflag = false;
      alert("GAME OVER");
    }
    if ((x + dx) < 0 || (x + dx) > (width /*+ padding*/) / 2) // bounce off
walls
    dx *= -1;
    if ((y + dy) < 0 || (y + dy) > height / 2) dy *= -1;
    x += dx;
    y += dy;
    ellipse(x, y, r, r);
  }
}
function readData (data) {
  var results = data.split(',');  // split the data on the commas
  // if(g0z > 570 && g0z < 930 && g1z > 440 && g1z < 940 && g2z > 485 && g2z <
965 && g3z > 225 && g3z < 1225 && g4z > 275 && g4z < 1275 && g5z > 535 && g5z <
1015 && g6z > 690 && g6z < 930 && g7z > 560 && g7z < 1060  && g0y > 140 && g0y
< 380 && g1y > 10 && g1y < 510 && g2y > 220 && g2y < 700 && g3y > 10 && g3y <
860 && g4y > 10 && g4y < 860 && g5y > 220 && g5y < 700 && g6y > 140 && g6y <
380 && g7y > 10 &&g7y < 510){
  if (results[0] == "0"){
    if(results[1] > 570 && results[1] < 930)
  g0z = results[1];                 // x is the first value
if(results[2] > 140 && results[2] < 380 )
  g0y = results[2];                 // y is the second value
}
  if (results[0] == "1"){            // button is the third value
    if(results[1] > 440 && results[1] < 940)
  g1z = results[1];                 // x is the first value
if(results[2] > 10 && results[2] < 510)
  g1y = results[2];                 // y is the second value
  //btn1 = results[3];
  }
  if (results[0] == "2"){            // button is the third value
    if(results[1] > 485 && results[1] < 965 )
  g2z = results[1];                 // x is the first value
if(results[2] > 460 && results[2] < 700)
  g2y = results[2];                 // y is the second value
  //btn2 = results[3];
  }
  if (results[0] == "3"){            // button is the third value
    if(results[1] > 225 && results[1] < 1225)
  g3z = results[1];                 // x is the first value
if(results[2] > 460 && results[2] < 860)
  g3y = results[2];                 // y is the second value
  //btn3 = results[3];
  }
  if (results[0] == "4"){            // button is the third value
    if(results[1] > 275 && results[1] < 1275)
  g4z = results[1];                 // x is the first value
if(results[2] > 460 && results[2] < 860)
  g4y = results[2];                 // y is the second value
  //btn4 = results[3];
```

```
  }
  if (results[0] == "5"){            // button is the third value
    if(results[1] > 535 && results[1] < 1015)
  g5z = results[1];                  // x is the first value
if(results[2] > 460 && results[2] < 700)
  g5y = results[2];                  // y is the second value
  //btn5 = results[3];
  }
  if (results[0] == "7"){            // button is the third value
    if(results[1] > 690 && results[1] < 930)
  g7z = results[1];                  // x is the first value
if(results[2] > 140 && results[2] < 380)
  g7y = results[2];                  // y is the second value
  //btn6 = results[3];
  }
  if (results[0] == "6"){            // button is the third value
    if(results[1] > 560 && results[1] < 1060)
  g6z = results[1];                  // x is the first value
if(results[2] > 10 &&results[2] < 510)
  g6y = results[2];                  // y is the second value
  //btn7 = results[3];
  }


   else{

   }
}
// when new data comes in the websocket, read it:
socket.on('message', readData);
```

Index.html

```
<html>
  <head>
  <script src="https://cdn.socket.io/socket.io-1.2.0.js"></script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.3.16/p5.min.js"></script>
    <script type="text/javascript" src="client.js"></script>
    <title></title>
  </head>
  <body>
  </body>
</html>
```

Gyronome Arduino Source Code

```
/*////////////////////////////////////////////////////////////
*                Gyronome v1.5.6 Source Code
* - Renamed gyro to "gyron"
* - Added Unity formatting
*
*                Gyronome v1.5.5 Source Code
* - Cleanup
* - Non-Mux Output Added
* - Mapped Output Added
*/

#include "I2Cdev.h"
#include "GyronMPU.h"
#include "MPU6050_6Axis_MotionApps20.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

#define OUTPUT_READABLE_QUATERNION
//#define OUTPUT_MAPPED
//#define OUTPUT_READABLE_EULER
//#define UNITY

//#define nonMUX
#define MUX


#define LED_PIN 13
#define M_PI 3.14159265358979323846
#define TCAADDR 0x70
#define GYRON_COUNT 8 //Change to appropriate
bool blinkState = false;
GyronMPU gyron[GYRON_COUNT];

#ifdef nonMUX
MPU6050 mpu;
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 =
success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer
Quaternion q;           // [w, x, y, z]         quaternion container
#endif

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin
has gone high
void dmpDataReady()
{
#ifdef nonMUX
       mpuInterrupt = true;
#endif
       for (int i = 0; i<GYRON_COUNT; i++)
       {
              gyron[i].mpuInterrupt = true;
       }
```

```
}

void tcaselect(uint8_t i) {
	if (i > 7) return;

	Wire.beginTransmission(TCAADDR);
	Wire.write(1 << i);
	Wire.endTransmission();
}


/*#############################################
*                  SETUP
*############################################*/
void setup()
{
	// join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
	Wire.begin(); //TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz).
Comment this line if having compilation difficulties with TWBR.
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
	Fastwire::setup(400, true);
#endif

	Serial.begin(115200); // initialize serial communication
	while (!Serial); // wait for Leonardo enumeration, others continue
immediately

#ifdef nonMUX
	Serial.println(F("Initializing I2C devices..."));
	mpu.initialize();

	Serial.println(F("Testing device connections..."));
	Serial.println(mpu.testConnection() ? F("MPU6050 connection successful")
: F("MPU6050 connection failed"));
	Serial.println(F("\nSend any character to begin DMP programming and
demo: "));
	Serial.println(F("Initializing DMP..."));
	devStatus = mpu.dmpInitialize();

	mpu.setXGyroOffset(62); //49
	mpu.setYGyroOffset(-44); //-32
	mpu.setZGyroOffset(-4); //43
	mpu.setXAccelOffset(-2846); //-1295
	mpu.setYAccelOffset(1968); //-3733
	mpu.setZAccelOffset(2094); //1485

	if (devStatus == 0)
	{
		Serial.println(F("Enabling DMP..."));
		mpu.setDMPEnabled(true);
		Serial.println(F("Enabling interrupt detection (Arduino external
interrupt 0)..."));
		attachInterrupt(0, dmpDataReady, RISING);
		mpuIntStatus = mpu.getIntStatus();
		Serial.println(F("DMP ready! Waiting for first interrupt..."));
		dmpReady = true;
		packetSize = mpu.dmpGetFIFOPacketSize();
	}
	else
	{
```

```
                // ERROR!
                // 1 = initial memory load failed
                // 2 = DMP configuration updates failed
                Serial.print(F("DMP Initialization failed (code "));
                Serial.print(devStatus);
                Serial.println(F(")"));
        }
#endif

#ifdef MUX
        for (int i = 0; i<GYRON_COUNT; i++)
        {
                tcaselect(i);

                //Serial.println(F("Initializing I2C devices..."));
                gyron[i].mpu.initialize();

                //Serial.println(F("Testing device connections..."));
                //Serial.println(gyron[i].mpu.testConnection() ? F("MPU6050
connection successful") : F("MPU6050 connection failed"));
                if (gyron[i].mpu.testConnection())
                {
                        Serial.print("Gyron");
                        Serial.print(i);
                        Serial.println(" connection successful");
                }
                else
                {
                        Serial.print("Gyron");
                        Serial.print(i);
                        Serial.println(" connection failed");
                }
                //Serial.println(F("\nSend any character to begin DMP programming
and demo: "));

                Serial.println(F("Initializing DMP..."));
                gyron[i].devStatus = gyron[i].mpu.dmpInitialize();

                switch (i) //Configure Offsets
                {
                case 0:
                        gyron[i].mpu.setXGyroOffset(35);
                        gyron[i].mpu.setYGyroOffset(-94);
                        gyron[i].mpu.setZGyroOffset(-21);
                        gyron[i].mpu.setXAccelOffset(2041);
                        gyron[i].mpu.setYAccelOffset(-607);
                        gyron[i].mpu.setZAccelOffset(1130);
                        break;
                case 1:
                        gyron[i].mpu.setXGyroOffset(61);
                        gyron[i].mpu.setYGyroOffset(-67);
                        gyron[i].mpu.setZGyroOffset(23);
                        gyron[i].mpu.setXAccelOffset(-5432);
                        gyron[i].mpu.setYAccelOffset(677);
                        gyron[i].mpu.setZAccelOffset(1361);
                        break;
                case 2:
                        gyron[i].mpu.setXGyroOffset(55);
                        gyron[i].mpu.setYGyroOffset(-29);
                        gyron[i].mpu.setZGyroOffset(46);
                        gyron[i].mpu.setXAccelOffset(-1439);
```

```
                gyron[i].mpu.setYAccelOffset(-3758);
                gyron[i].mpu.setZAccelOffset(1514);
                break;
            case 3:
                gyron[i].mpu.setXGyroOffset(53);
                gyron[i].mpu.setYGyroOffset(190);
                gyron[i].mpu.setZGyroOffset(-79);
                gyron[i].mpu.setXAccelOffset(-2019);
                gyron[i].mpu.setYAccelOffset(-2042);
                gyron[i].mpu.setZAccelOffset(641);
                break;
            case 4:
                gyron[i].mpu.setXGyroOffset(91);
                gyron[i].mpu.setYGyroOffset(3);
                gyron[i].mpu.setZGyroOffset(13);
                gyron[i].mpu.setXAccelOffset(-2179);
                gyron[i].mpu.setYAccelOffset(1406);
                gyron[i].mpu.setZAccelOffset(850);
                break;
            case 5:
                gyron[i].mpu.setXGyroOffset(4);
                gyron[i].mpu.setYGyroOffset(-30);
                gyron[i].mpu.setZGyroOffset(-30);
                gyron[i].mpu.setXAccelOffset(-4191);
                gyron[i].mpu.setYAccelOffset(1647);
                gyron[i].mpu.setZAccelOffset(1326);
                break;
            case 6:
                gyron[i].mpu.setXGyroOffset(91);
                gyron[i].mpu.setYGyroOffset(-45);
                gyron[i].mpu.setZGyroOffset(-43);
                gyron[i].mpu.setXAccelOffset(-695);
                gyron[i].mpu.setYAccelOffset(-3424);
                gyron[i].mpu.setZAccelOffset(1200);
                break;
            case 7:
                gyron[i].mpu.setXGyroOffset(139);
                gyron[i].mpu.setYGyroOffset(42);
                gyron[i].mpu.setZGyroOffset(-1);
                gyron[i].mpu.setXAccelOffset(-4803);
                gyron[i].mpu.setYAccelOffset(-2608);
                gyron[i].mpu.setZAccelOffset(1033);
                break;
            default:
                gyron[i].mpu.setXGyroOffset(51); //49
                gyron[i].mpu.setYGyroOffset(-30); //-32
                gyron[i].mpu.setZGyroOffset(43); //43
                gyron[i].mpu.setXAccelOffset(-1295); //-1295
                gyron[i].mpu.setYAccelOffset(-3733); //-3733
                gyron[i].mpu.setZAccelOffset(1485); //1485 // 1688 factory
default
                break;
            }

            if (gyron[i].devStatus == 0)
            {
                //Serial.println(F("Enabling DMP..."));
                gyron[i].mpu.setDMPEnabled(true);

                //Serial.println(F("Enabling interrupt detection (Arduino
external interrupt 0)..."));
```

```
                        attachInterrupt(0, dmpDataReady, RISING);
                        gyron[i].mpuIntStatus = gyron[i].mpu.getIntStatus();

                        Serial.print("Gyron");
                        Serial.print(i);
                        Serial.println(F(" DMP ready! Waiting for first
interrupt..."));
                        gyron[i].dmpReady = true;

                        gyron[i].packetSize = gyron[i].mpu.dmpGetFIFOPacketSize();
                }
                else
                {
                        // ERROR!
                        // 1 = initial memory load failed
                        // 2 = DMP configuration updates failed
                        Serial.print("Gyron");
                        Serial.print(i);
                        Serial.print(F(" DMP Initialization failed (code "));
                        Serial.print(gyron[i].devStatus);
                        Serial.println(F(")"));
                }
        }
#endif
        pinMode(LED_PIN, OUTPUT);
}

/*#######################################
*          MAIN LOOP
*######################################*/

void loop()
{
#ifdef nonMUX
        if (!dmpReady) return; // if programming failed, don't try to do
anything
                                                // wait for MPU interrupt or extra
packet(s) available
        while (!mpuInterrupt && fifoCount < packetSize)
        {
                if (mpuInterrupt)
                {
                        mpuInterrupt = false;
                        break;
                }
                else
                {
                        mpu.resetFIFO();
                        break;
                }
        }
        mpuInterrupt = false; // reset interrupt flag and get INT_STATUS byte
        mpuIntStatus = mpu.getIntStatus();
        fifoCount = mpu.getFIFOCount();
        if ((mpuIntStatus & 0x10) || fifoCount == 1024) // check for overflow
        {
                mpu.resetFIFO(); // reset so we can continue cleanly
        }
        else if (mpuIntStatus & 0x02)
        {
                while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
```

```
            mpu.getFIFOBytes(fifoBuffer, packetSize); // read a packet from
FIFO
            fifoCount -= packetSize;

            mpu.dmpGetQuaternion(&q, fifoBuffer); // Get Quaternion Values


//Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("*");
            Serial.print(q.x);
            Serial.print("*");
            Serial.print(q.y);
            Serial.print("*");
            Serial.println(q.z);
      }
#endif

#ifdef MUX
      for (int i = 0; i<GYRON_COUNT; i++)
      {
            tcaselect(i);
            if (!gyron[i].dmpReady) return; // if programming failed, don't
try to do anything

                                                                  // wait for
MPU interrupt or extra packet(s) available
            while (!gyron[i].mpuInterrupt && gyron[i].fifoCount <
gyron[i].packetSize)
            {
                  //Serial.print(i);
                  //Serial.println(" inside first while");
                  if (gyron[i].mpuInterrupt)
                  {
                        gyron[i].mpuInterrupt = false;
                        break;
                  }
                  else
                  {
                        gyron[i].mpu.resetFIFO();
                        break;
                  }
                  Serial.println("me");
            }

            gyron[i].mpuInterrupt = false; // reset interrupt flag and get
INT_STATUS byte
            gyron[i].mpuIntStatus = gyron[i].mpu.getIntStatus();

            gyron[i].fifoCount = gyron[i].mpu.getFIFOCount();

            if ((gyron[i].mpuIntStatus & 0x10) || gyron[i].fifoCount == 1024)
// check for overflow
            {
                  gyron[i].mpu.resetFIFO(); // reset so we can continue
cleanly

            }
            else if (gyron[i].mpuIntStatus & 0x02)
            {
```

```
                    while (gyron[i].fifoCount < gyron[i].packetSize)
                    {
                           //Serial.print(i);
                           //Serial.println(" inside second while");
                           gyron[i].fifoCount = gyron[i].mpu.getFIFOCount();
                    }


                    gyron[i].mpu.getFIFOBytes(gyron[i].fifoBuffer,
gyron[i].packetSize); // read a packet from FIFO
                    gyron[i].fifoCount -= gyron[i].packetSize;
                    gyron[i].mpu.dmpGetQuaternion(&gyron[i].q,
gyron[i].fifoBuffer); // Get Quaternion Values
                    gyron[i].mpu.resetFIFO();
#ifdef OUTPUT_READABLE_QUATERNION
                    // display quaternion values in easy matrix form: w x y z

                    Serial.print("quat");
                    Serial.print(i);
                    Serial.print("\t");
                    Serial.print(gyron[i].q.w);
                    Serial.print(",");
                    Serial.print(gyron[i].q.x);
                    Serial.print(",");
                    Serial.print(gyron[i].q.y);
                    Serial.print(",");
                    Serial.println(gyron[i].q.z);
#endif

#ifdef UNITY
                    Serial.print(i);
                    Serial.print("*");
                    Serial.print(gyron[i].q.w);
                    Serial.print("*");
                    Serial.print(gyron[i].q.x);
                    Serial.print("*");
                    Serial.print(gyron[i].q.y);
                    Serial.print("*");
                    Serial.println(gyron[i].q.z);
                    delay(8);
#endif

#ifdef OUTPUT_MAPPED
                    gyron[0].q.z = gyron[0].q.z * 1000;
                    gyron[0].q.z = map(gyron[0].q.z, -1000, 1000, 570, 810);
                    gyron[0].q.y = gyron[0].q.y * 1000;
                    gyron[0].q.y = map(gyron[0].q.y, -1000, 1000, 140, 380);

                    gyron[1].q.z = gyron[1].q.z * 1000;
                    gyron[1].q.z = map(gyron[1].q.z, -1000, 1000, 440, 940);
                    gyron[1].q.y = gyron[1].q.y * 1000;
                    gyron[1].q.y = map(gyron[1].q.y, -1000, 1000, 10, 510);

                    gyron[2].q.z = gyron[2].q.z * 1000;
                    gyron[2].q.z = map(gyron[2].q.z, -1000, 1000, 485, 965);
                    gyron[2].q.y = gyron[2].q.y * 1000;
                    gyron[2].q.y = map(gyron[2].q.y, -1000, 1000, 220, 700);

                    gyron[3].q.z = gyron[3].q.z * 1000;
                    gyron[3].q.z = map(gyron[3].q.z, -1000, 1000, 225, 1225);
                    gyron[3].q.y = gyron[3].q.y * 1000;
```

```
                    gyron[3].q.y = map(gyron[3].q.y, -1000, 1000, 10, 860);

                    gyron[4].q.z = gyron[4].q.z * 1000;
                    gyron[4].q.z = map(gyron[4].q.z, -1000, 1000, 275, 1275);
                    gyron[4].q.y = gyron[4].q.y * 1000;
                    gyron[4].q.y = map(gyron[4].q.y, -1000, 1000, 10, 860);

                    gyron[5].q.z = gyron[5].q.z * 1000;
                    gyron[5].q.z = map(gyron[5].q.z, -1000, 1000, 535, 1015);
                    gyron[5].q.y = gyron[5].q.y * 1000;
                    gyron[5].q.y = map(gyron[5].q.y, -1000, 1000, 220, 700);

                    gyron[6].q.z = gyron[6].q.z * 1000;
                    gyron[6].q.z = map(gyron[6].q.z, -1000, 1000, 690, 930);
                    gyron[6].q.y = gyron[6].q.y * 1000;
                    gyron[6].q.y = map(gyron[6].q.y, -1000, 1000, 140, 380);

                    gyron[7].q.z = gyron[7].q.z * 1000;
                    gyron[7].q.z = map(gyron[7].q.z, -1000, 1000, 560, 1060);
                    gyron[7].q.y = gyron[7].q.y * 1000;
                    gyron[7].q.y = map(gyron[7].q.y, -1000, 1000, 10, 510);
#endif

#ifdef OUTPUT_READABLE_EULER
                    // display Euler angles in degrees
                    gyron[i].mpu.dmpGetQuaternion(&gyron[i].q,
gyron[i].fifoBuffer);
                    gyron[i].mpu.dmpGetEuler(gyron[i].euler, &gyron[i].q);
                    //Serial.print("euler\t");
                    Serial.print(gyron[i].euler[0] * 180 / M_PI * 10);
                    Serial.print("*");
                    Serial.print(gyron[i].euler[1] * 180 / M_PI * 10);
                    Serial.print("*");
                    Serial.println(gyron[i].euler[2] * 180 / M_PI * 10);
#endif
                }
        }

#endif
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState); // blink LED to indicate activity
}
```

Gyron Header File

```
#pragma once
//#include "MPU6050.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "helper_3dmath.h"
//#include "MPU6050_6Axis_MotionApps20.h"

class GyronMPU
{
public:
        MPU6050 mpu;

        bool dmpReady;  // set true if DMP init was successful
        uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
        uint8_t devStatus;      // return status after each device operation (0
= success, !0 = error)
```

```
        uint16_t packetSize;    // expected DMP packet size (default is 42
bytes)
        uint16_t fifoCount;     // count of all bytes currently in FIFO
        uint8_t fifoBuffer[64]; // FIFO storage buffer
        Quaternion q;
        float euler[3];
        volatile bool mpuInterrupt;

        GyronMPU()
        {
                dmpReady = false;
                mpuInterrupt = false;
                euler[0] = 0;
                euler[1] = 0;
                euler[2] = 0;
        }
};
```