IIT Bombay

PH 435 Project

Maze Ball Game Simulation

Student Name	**Student ID**
1. Nahush Rajesh Kolhe	210260034
2. Spandan Sachin Anaokar	210260055

Lecturer in charge:

Prof. Pramod Kumar

Date : 10/11/23

Contents

1 Abstract

In the world of gaming, the classic Maze Ball game (labyrinth game) stands as an iconic challenge where players navigate a ball through a maze while avoiding pitfalls and aiming to reach the goal. This project takes a modern and innovative approach to this timeless game by simulating it through the use of a gyroscope sensor for input and laptop display for the gaming experience.

Our objective was to create a digital representation of the labyrinth game that captures the essence of its physical counterpart while leveraging the capabilities of the MPU-6050 gyroscope sensor. This sensor, mounted on a controller, allows players to manipulate the ball's orientation and movement using gravity, enhancing the gameplay experience.

By interfacing the gyroscope sensor with a laptop, we not only recreated the physical interaction but also explored the dynamics of gyroscope technology. Our focus was on harnessing the angular velocity data from the sensor to calculate and control the ball's acceleration, providing a seamless and immersive gaming experience.

This project report delves into the technical aspects of gyroscope sensor integration, data processing, and real-time display, offering insights into the challenges and solutions encountered during development. We also discuss the pivotal role of MEMS technology in sensor operation. The successful simulation of the maze ball game using the gyroscope sensor opens up possibilities for exciting and intuitive gaming interfaces.

By bridging the gap between the physical and digital gaming worlds, this project exemplifies the fusion of technology and entertainment. It showcases the potential of sensor-driven simulations to create immersive and engaging gaming experiences, emphasizing the role of innovation in transforming classic pastimes into modern marvels.

2 Introduction

The Maze Ball game (labyrinth game), a beloved classic of physical dexterity, has captured the imagination of generations, challenging players to navigate a small ball through a maze while skillfully avoiding treacherous pitfalls. In this age of technological innovation, we embark on a journey to reimagine and simulate this timeless game using the MPU-6050 gyroscope sensor, bridging the physical and digital realms of entertainment.

What sets our project apart is the meticulous crafting of the labyrinth game from the ground up, coded with precision and care. We have embarked on the ambitious task of creating the game from scratch, implementing every facet of its design and mechanics, while incorporating the MPU-6050 gyroscope sensor to offer an authentic and immersive gaming experience. The project showcases our dedication to coding prowess and the creative fusion of traditional gaming with cutting-edge technology.

The allure of the labyrinth game lies in its simplicity and elegance, where a delicate balance of hand-eye coordination, intuition, and strategy determines success. Our project takes a contemporary approach to this venerable pastime by harnessing the capabilities of the MPU-6050 gyroscope sensor, a powerful micro-electro-mechanical system (MEMS) device. This sensor, when integrated with a controller, enables players to manipulate the ball's orientation and movement by simply tilting and rotating the controller, thus imbuing the game with a genuine and immersive feel.

At the core of our project is the idea of using the gyroscope sensor to capture and interpret angular velocity data, which serves as the basis for calculating the ball's acceleration. By doing so, we replicate the physics of the labyrinth game, where gravity is the player's greatest adversary and ally. This simulation leverages modern technology to faithfully replicate the tactile experience of tilting the maze and guiding the ball towards the elusive goal.

Our dedication to coding the game from scratch, combined with the incorporation of the MPU-6050 gyroscope sensor, represents an exciting frontier in game development, demonstrating the potent synergy of tradition and technology. This project report delves into the technical aspects of integrating the gyroscope sensor with a laptop and displaying the game using Processing, shedding light on the challenges encountered and the innovative solutions devised during development. By simulating the labyrinth game through the MPU-6050 gyroscope sensor and our meticulously coded game, we not only pay homage to a beloved classic but also demonstrate the potential of sensor-driven simulations in creating engaging and intuitive gaming experiences. This endeavor exemplifies the fusion of tradition and technology, underscoring the transformative power of innovation in converting cherished pastimes into modern marvels.

In the following sections of this report, we will provide a comprehensive account of our project's design, required components, detailed working, showcasing the synergies between gaming, coding prowess, and cutting-edge technology.

3 Hardware

The project uses the following pieces of Hardware:

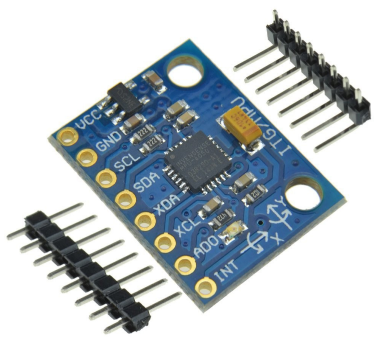
Arduino Uno The Arduino Uno is a popular microcontroller board known for its simplicity and versatility. It features an ATmega328P microcontroller, numerous digital and analog I/O pins, and is widely used for various electronics projects and prototyping.

MPU-6050 Accelerometer and Gyroscope Sensor The MPU-6050 is a motion tracking device that combines a 3-axis gyroscope and a 3-axis accelerometer in a single chip. It is commonly used for measuring motion and orientation in various electronic applications, including robotics and wearable devices. The MPU-6050 provides precise data on acceleration and angular velocity, making it a valuable component for motion sensing and control.

Breadboard and Wires

Laptop (for display) We transmit the data for the location of the game objects to the laptop. The laptop can be replaced with any display device as all game calculations are done by the arduino.

(a) Arduino



(b) MPU- 6050 Gyroscope

Figure 1: Hardware

4 Methods

The primary objective of this project is to meticulously replicate the immersive experience of the *Maze Ball Game* in a digital simulation. To achieve this, our project is structured into two distinct phases.

In the initial phase, we focus on capturing sensor data, specifically the readings from our gyroscopic sensor, to determine the precise orientation of the controller. Simultaneously, we harness these sensor readings to calculate the acceleration experienced by the virtual *ball* within the *maze*. This initial step lays the foundation for accurately recreating the game's dynamic and gravity-dependent behavior.

The second phase of our project is equally crucial. Here, we utilize the calculated acceleration values to perform intricate calculations that account for the *maze's walls*, potential *danger zones*, and the ultimate *goal*. These calculations enable us to generate a comprehensive data set that encompasses all the vital elements required for the visual representation on the screen. In essence, this phase acts as the core engine, driving the game's realism by dynamically adapting the maze, providing a seamless and captivating user experience.

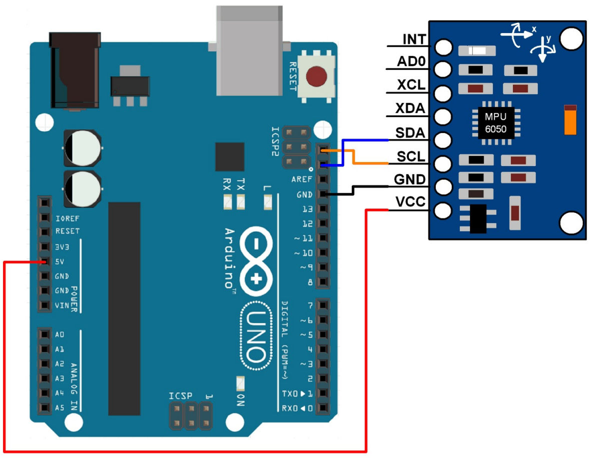


Figure 2: Circuit Diagram

4.1 MPU-6050 Connection

For simulating the game, we need to enforce the physics of the ball. In our game, input is provided by the controller, and the player adjusts the ball's orientation using gravity. To simulate the acceleration of the ball in gravity, we require the measurement of the angle at which the controller is oriented, allowing us to calculate the ball's acceleration along the game board.

We use an MPU 6050 Gyroscope sensor mounted on the controller to determine its orientation. Our interest lies in the reading of angular velocity from the gyroscope, as it does not readily provide angular orientation. Therefore, by integrating angular velocity (essentially using an iterative sum), we obtain the angular orientation.

We are specifically concerned with the rotation about the X and Y axes of the gyroscope. Through this process, we obtain angles denoted as angle X and angle

Y, representing the controller's rotation about the X and Y axes, respectively. The acceleration in the X and Y directions can be determined as $g \cdot \sin(\text{angle Y})$ and $g \cdot \sin(\text{angle X})$, respectively. These values can then be used to update the position of the ball using the equations of kinematics.

4.2 Game Elements

One of the fundamental aspirations guiding our project is the ability to craft our code in the expressive and versatile C language, which can seamlessly reside within the Arduino platform. To accomplish this, we adopt a sophisticated approach centered around the utilization of classes and objects.

In our programming endeavor, classes and objects serve as the cornerstones that allow us to eloquently encapsulate various game elements. By employing the power of object-oriented design, we define and structure essential components of our game. These classes represent the very essence of our game, encapsulating its unique attributes, behaviors, and functionalities.

These meticulously crafted classes are akin to building blocks, each designed to fulfill a specific role or characteristic within the game. By artfully combining these classes and their respective objects, we create a harmonious synergy that defines the intricacies of our game. This approach not only enhances the modularity and maintainability of our code but also enriches our project with a robust and flexible foundation, enabling us to bring the Maze Ball Game to life within the Arduino environment.

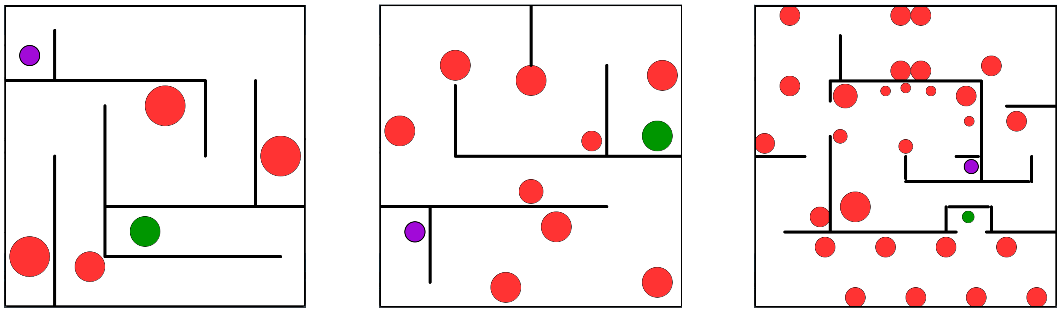
These are the game structures we have used in our game:

- Ball: The main player-controlled object.
- Wall: Obstacles that deny passage of ball through it.
- Goal: The target to reach and complete the level.
- Hole: Dangerous areas to avoid or fall into.

Along with this we use multiple functions that help in running the game elements:

- Ball_update: Updates the position of the Ball
- checkGoal: Check if the ball has reached the Goal
- checkHole: Check if the ball touches the hole
- checkWall: Check if the ball crosses a wall. If it does then update the ball position to just touching the wall
- levelchange: Change the level and along with that all values of the game elements position
- ending: display animation when the level ends

4.3 Levels



(a) Level 1 (b) Level 2 (c) Level 3

Figure 3: Levels of Game

4.4 Display

In our project, we have harnessed the capabilities of Processing, a remarkably potent tool for graphics and animation, enriched with an integrated Arduino Serial Communication feature. The code within Processing is crafted in JAVA, and it acts as our artistic canvas, where we paint the vibrant visuals of our Maze Ball Game.

The concept at the heart of our approach is to channel a stream of data from the Arduino, which comprehensively encapsulates the current game state. This data stream contains a series of numbers that collectively represent all the vital aspects of the game at any given moment. Drawing upon this raw numerical data, we meticulously reconstruct the game's structural elements and dynamics. Leveraging Processing's extensive library of graphical functions, we conjure up the maze, the ball, and all other game elements with precision.

Utilizing functions such as 'circle' and 'line,' we translate the numerical information into visual entities, each with its unique appearance and behavior. This meticulous translation process ensures that the game's representation is faithful to the current state of the Arduino-based simulation.

Furthermore, our data communication setup maintains an ongoing and real-time connection between the Arduino and the laptop. This continuous exchange of data enables Processing to provide dynamic and instantaneous updates on the ball's position within the maze. Consequently, players experience an immersive and interactive gaming environment, where their every move is faithfully mirrored on the screen in real-time.

5 Working

In this section we will discuss in detail the working of MPU 6050 Gyroswcope sensor, and the arduino & processing code.

5.1 MPU 6050 Gyroscope Working

The MPU-6050 is a widely used Inertial Measurement Unit (IMU) in the field of electronics and robotics. It combines a 3-axis accelerometer and a 3-axis gyroscope within a single compact package. This sensor is designed to provide information about an object's motion, orientation, and changes in velocity by measuring acceleration and angular velocity in all three spatial dimensions (X, Y, and Z). To understand how the gyroscope part of the MPU-6050 works, it's essential to delve into the principles behind Micro-Electro-Mechanical Systems (MEMS) technology and the gyroscope's operation.

5.1.1 MEMS Technology

MEMS devices are miniaturized mechanical and electro-mechanical elements that are integrated onto a silicon chip using microfabrication techniques. These tiny mechanical structures, often on the nanometer to micrometer scale, can detect and respond to changes in motion or orientation. In the case of gyroscopes, MEMS technology is used to create extremely small and sensitive mechanical structures that can measure angular velocity.

5.1.2 Coriolis Effect

Gyroscopes measure angular rotation. To accomplish this, they measure the force generated by the Coriolis Effect.

The Coriolis Effect states that when a mass (m) moves in a specific direction with a velocity (v) and an external angular rate (Ω) is applied (depicted by the red arrow), the Coriolis Effect generates a force (indicated by the yellow arrow) that causes the mass to move perpendicularly. The magnitude of this displacement is directly proportional to the angular rate applied.

Consider two masses oscillating in opposite directions at a constant frequency. When an angular rate is applied, the Coriolis effect produced by each mass acts in opposite directions, resulting in a corresponding change in capacitance between the masses. By measuring this change in capacitance, the angular rate can be accurately calculated.

5.1.3 MEMS Gyroscope Working

The MEMS sensor consists of a proof mass (consisting of four parts M1, M2, M3, and M4) that is maintained in a continuous oscillating movement so that it can respond to the coriolis effect. They simultaneously move inward and outward in the horizontal plane. When we begin to rotate the structure, the Coriolis force acting on the moving proof mass causes the vibration to change from horizontal

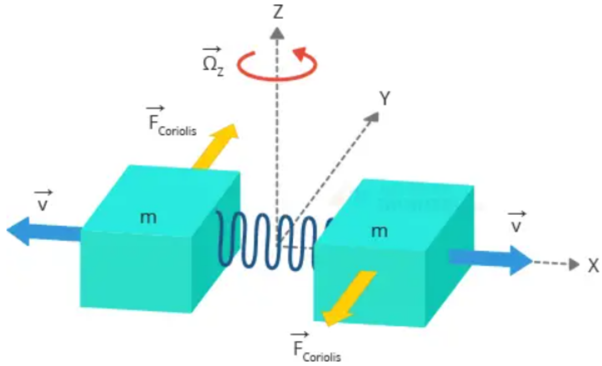


Figure 4: Coriolis force on oscillating masses

to vertical. There are three modes depending on the axis along which the angular rotation is applied.

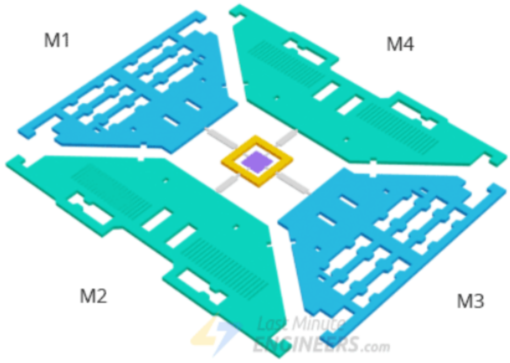


Figure 5: MEMS for Gyroscope

- **Roll Mode**: When an angular rate is applied along the X-axis, M1 and M3 will move up and down out of the plane due to the coriolis effect. This causes a change in the roll angle, hence the name Roll Mode.
- **Pitch Mode**: When an angular rate is applied along the Y-axis, M2 and M4 will move up and down out of the plane. This causes a change in the pitch angle, hence the name Pitch Mode.
- **Yaw Mode**: When an angular rate is applied along the Z-axis, M2 and M4 will move horizontally in opposite directions. This causes a change in the yaw angle, hence the name Yaw Mode.

Whenever the coriolis effect is detected, the constant motion of the driving mass will cause a change in capacitance (ΔC) that is detected by the sensing structure and converted into a voltage signal.

5.1.4 Communication of Gyroscope

I2C (Inter-Integrated Circuit) is a popular serial communication protocol that enables multiple devices to communicate with each other using just two wires: SDA (Serial Data Line) and SCL (Serial Clock Line). It facilitates communication between various integrated circuits in electronic devices. In an I2C communication setup, there is a master device (e.g., Arduino) and one or more slave devices (e.g., the gyroscope).

The master initiates and controls the data transfer, while the slave devices respond to the master's commands. Each device connected to the I2C bus has a unique address. The master sends the address of the specific slave device it wants to communicate with. Data is transmitted on the SDA line, synchronized by clock pulses on the SCL line.

Communication starts with a start condition (SDA transitions from high to low while SCL is high) and ends with a stop condition (SDA transitions from low to high while SCL is high). These signals help in distinguishing the beginning and end of data transfer.

After each byte transfer, the receiving device acknowledges the reception. If the device acknowledges, the data transfer continues. If it doesn't, it may indicate an error. The gyroscope, as a slave device, has its own unique address on the I2C bus. When the Arduino (as the master) wants to read data from the gyroscope, it sends the gyroscope's address, followed by a command indicating the specific data it needs.

The gyroscope responds by sending the requested data back to the Arduino through the SDA line, synchronized by the clock pulses on the SCL line. This way, the gyroscope employs the I2C protocol to send data through the SDA and SCL lines in a synchronized manner, ensuring reliable and efficient communication.

5.2 Arduino Code Working

The Arduino assumes a pivotal role in our project, taking on the crucial responsibilities of housing all game elements, executing intricate calculations, and orchestrating the transmission of display data to Processing on the laptop through the Serial Port. This section delves deep into the intricacies of the code, unraveling the inner workings of this integral process.

5.2.1 Import and Parameter Definition

```
#include <MPU6050_tockn.h>
#include <Wire.h>

// Define global parameters that control the game's behavior
int lim = 6;                // Maximum velocity limit for the ball
int updaterate = 5;         // The rate at which the game updates
float scalea = 1;           // Scaling factor for acceleration
float scalev = 2;           // Scaling factor for velocity
int level = 1;              // The current game level
bool cheatMode = false;     // A cheat mode flag
```

- This section includes necessary libraries for the project.
- Global variables are defined to control various aspects of the game, such as maximum velocity, update rate, scaling factors, current level, and cheat mode.

- Less the update rate, faster the game would be making it harder.

5.2.2 Data Structures

```
// Define data structures (structs) to represent game elements
typedef struct Wall {
    int startx;
    int starty;
    int endx;
    int endy;
    bool vert;
} Wall;

typedef struct Ball {
    float cenx;
    float ceny;
    int r;
    float vx;
    float vy;
} Ball;

typedef struct Hole {
    int x;
    int y;
    int r;
} Hole;
```

- Defines three data structures (structs) representing game elements: Wall, Ball, and Hole.
- These structs store information about walls, the game ball, and holes, respectively.

5.2.3 Ball Update Function

```
// Update the ball's position based on acceleration
void Ball_update(Ball &b, float x, float y) {
    // Update velocity
    b.vx += x / scalea;
    b.vy += y / scalea;

    // Ensure velocity limits
    // ...

    // Update position
    b.cenx += b.vx / scalev;
    b.ceny += b.vy / scalev;
}
```

- The Ball_update function updates the ball's position based on accelerometer data.
- It also ensures that the ball's velocity and position remain within certain limits so as to not break game balance.

5.2.4 Check Goal function

```
// Check if the ball reaches the goal
bool checkGoal(Ball ball, Hole goal) {
    // Check if the distance between the ball and the goal is less
    than their combined radii
    if (/*Condition .../*) {
        return true;
    } else {
        return false;
    }
}
```

- The checkGoal function determines if the ball has reached the goal by comparing the distances between their centers and radii.
- It returns a boolean value indicating whether the goal has been achieved.

5.2.5 Check Hole function

```
// Check if the ball touches a hole
bool checkHole(Ball ball, Hole hole) {
    // Check if the distance between the ball and the hole is less
    than their combined radii
    if (/*Condition .../*) {
        return true;
    } else {
        return false;
    }
}
```

- The checkHole function checks if the ball has touched a hole by comparing the distances between their centers and radii.
- It returns a boolean value indicating whether the ball has entered a hole.

5.2.6 Check Wall function

```
    // Check if the ball crosses a wall and adjust its position
    accordingly
void checkWall(Ball &ball, Wall w) {
    float sc = 0;    //Bounce factor
    if (w.vert) {
        // Check if the ball intersects a vertical wall and adjust
    its position
        if (/*Condition .../*) {
            //Check which side of wall to send the ball to
            if (ball.cenx >= w.startx) {
                ball.cenx = w.startx + ball.r;
                ball.vx = -ball.vx * sc;
            } else {
                ball.cenx = w.startx - ball.r;
                ball.vx = -ball.vx * sc;
            }
        }
    } else {
```

```
        // Check if the ball intersects a horizontal wall and adjust its position
        if (ball.cenx + ball.r > w.startx && ball.cenx - ball.r < w.endx &&
        //     ...
    }
}
```

- The checkWall function checks if the ball crosses a wall and adjusts its position if necessary.
- The conditions for the ball's interaction with the wall are twofold. First, the ball must be positioned between the ends of the wall. Secondly, the center of the ball must fall within a certain distance, equal to its radius, from the wall
- It ensures that the ball's movement is constrained by walls and reflects realistic physics.

5.2.7 Display Function

```
// Send all display data as a long list of integers to the Serial Port
void display2(Ball b, Hole g, Wall w[], unsigned int wlen, Hole h[], unsigned int hlen) {
    // Send display data to the Serial Port
    // ...
}
```

- The display2 function sends a stream of integers representing game data to the Serial Port.
- This data includes information about the ball, goal, walls, and holes, allowing it to be displayed in real-time.

5.2.8 Level Initialization

```
// Initialize game elements based on the current level
void levelchange(int l) {
    // Set up game elements based on the specified level
    if (l == 1) {
        // Level 1 initialization
        // ...
    } else if (l == 2) {
        // Level 2 initialization
        // ...
    } else if (l == 3) {
        // Level 3 initialization
        // ...
    }
}
```

- The levelchange function initializes game elements based on the specified level.
- Different levels have specific configurations for the ball, goal, walls, and holes.

5.2.9 Ending Animation Function

```
// Create an animation where the ball moves to the center of a goal
    or hole
void ending(Ball &b, Hole hol) {
    // ...
}
```

- The ending function creates an animation where the ball smoothly moves to the center of a goal or hole.
- This animation enhances the visual experience when a level is completed.

5.2.10 Setup

```
void setup() {
    // Initialize Serial communication, Wire, and MPU6050
    // ...
    // Start the game by changing the level and sending a signal to
    begin the gaming loop
    levelchange(level);
}
```

- The setup function initializes essential components, including Serial communication, Wire (I2C communication), and the MPU6050 sensor.
- It also triggers the start of the game by changing the level and sending a signal to begin the gaming loop.

5.2.11 Loop

```

//Gaming loop begins
void loop() {
    //Increment the n->no. of loops before display update
    n+=1;

    //Calculate the acceleration values that the ball follows
    // ...

    //Conduct check if Display device sends any information. If it
    does do the corresponding action
    if (Serial.available())
    {
        // Check what message comes from Display device and perform
    corresponding action
        // ...
    }

    //Update the game parameters after every updaterate no of loops
    take place
    if(n%updaterate==0)
    {
        //Update the ball position
        Ball_update(b, accy, accx);

        //Check if goal reached
```

```
        if(checkGoal(b,g)){
            ending(b, g);

            //Send output denoting win and Wait for response from Display device
            // ...
        }

        //Check if any hole is reached
        for (int i=0; i<hlen && !cheatMode; i++){
            if(checkHole(b, h[i])){
                ending(b, h[i]);

                //Send output denoting loss and Wait for response from Display device
                // ...
            }
        }

        //Check and update ball for all walls
        for(int i=0; i<wlen && !cheatMode; i++){
            checkWall(b, w[i]);
        }

        //Send game elements info. to Display device
        display2(b, g, w, wlen, h, hlen);
        n=0; //Reset the count of no of loops after last game update
    }
```

- This is the main gaming loop where the game logic and updates take place.
- The loop handles game dynamics, including level changes, user input, ball movement, collision detection, and display updates.
- After a level ends, it waits for response from the display device and takes action accordingly.
- Using n we change our rate of updation of game elements thereby manipulating the game speed.

5.3 Processing Code Working

Processing is an open-source software and programming language designed to simplify creative coding and visual design. It's a vital tool for artists, designers, and developers, enabling them to create interactive applications, data visualizations, and artworks. Processing's user-friendly environment makes it accessible to a wide audience, bridging the gap between technology and creativity. It empowers users, regardless of their coding experience, to express their creativity through code-driven art and interactive experiences. With extensive libraries for graphics, sound, and interactivity, Processing democratizes coding and fosters innovation in digital art and interactive design

5.3.1 Import Libraries and Define Global Variables

```
import processing.serial.*;
import uibooster.*;

Serial myPort;
String val;       // Store the value gotten from Serial
boolean started = false;  // If gaming loop has started
int scale = 3;  // Screen size (default is 300*300)
```

- Import Processing and "uibooster" libraries.
- Define variables for serial communication and game control.
- Set the screen size scaling factor.

5.3.2 Setup Function

```
void setup() {
  size(900, 900);  // Set the display size (adjust accordingly)
  String portName = Serial.list()[0];  // Get the name of the serial port
  myPort = new Serial(this, portName, 115200);  // Initialize the serial port
  noCursor();  // Hide the mouse cursor
  background(255, 255, 255);  // Set the background color
}
```

- The setup function is executed once at the beginning of the program.
- Set the display size to 900x900 pixels (adjust according to the scale).
- Initialize the serial port with a specified baud rate.
- Hide the mouse cursor and set the background color to white.

5.3.3 State Array Initialization

```
int[] state = {0, 0, 0, 0, 0, 0, 0, ... }; // Define the state array
```

- Declare an integer array named "state" to store the game's state.
- The array contains numerous values representing the state of the game screen.

5.3.4 Display Function

```
void disp() {
  // Draw the goal, holes, walls, and the ball on the screen based on the data stored in the "state" array.
  // ...
}
```

- The "disp()" function is responsible for displaying the game elements.
- Retrieve the number of walls and holes from the "state" array.
- Clear the background by setting it to white.
- Draw the goal, holes, walls, and the ball on the screen based on the data stored in the "state" array.

5.3.5 Draw Loop Function

```
void draw() {

if (myPort.available() > 0) {
  val = myPort.readStringUntil('\n');
  println(val);

    //Check if gaming loop has started
  if (val != null && val.contains("TheWorldEatingPythonIsBack")) {
    started = true;
  } else if (val != null && started == true) {
    int[] data = int(trim(split(trim(val), ' ')));


    if (data[0] == 1) {
      print(data);

      if (data[1] == 1) {
        //Display dialogue box for losing the level
        // ...
      }
      if (data[1] == 0) {
        //Display dialogue box for clearing the level
        // ...
      }
      if (data[1] == 2) {
        //Display dialogue box for winning the game
        // ...
      }
    } else if (data[0] == 0) {
      // Update the state variables if we receive non-null data from
    arduino
      for (int i = 0; i < data.length; i++) {
        state[i] = data[i];
      }
    }
  }
}

if (keyPressed) {
  //For each key pressed write it to myPort
  }
}
```

- The "draw()" function is continuously executed during the program's run-time and it checks if data is available from the serial port.
- Read and store the data in the "val" variable.
- Wait for the code word to start the game loop.
- Update the game state based on the received data.
- Display the game elements using the "disp()" function.
- Send commands to the Arduino for level changes and cheat mode based on key presses.

6 Conclusion

In conclusion, our project has successfully brought the classic Maze Ball game into the digital age, offering a fresh and innovative gaming experience. By incorporating the MPU-6050 gyroscope sensor, we've not only recreated the physical interaction but also explored the dynamic possibilities of gyroscope technology. Our report highlights the technical intricacies of sensor integration, data processing, and real-time display while shedding light on the role of MEMS technology.

Our achievement in simulating the maze ball game using the gyroscope sensor underscores the potential of sensor-driven simulations to create immersive and engaging gaming interfaces. It exemplifies the fusion of technology and entertainment, emphasizing how innovation can transform timeless pastimes into modern marvels. As technology continues to evolve, our project serves as a testament to the exciting possibilities it offers for enhancing classic games and redefining the gaming landscape

7 Author's Contribution

- **Nahush Kolhe** Setting up and reading data from Gyroscope, used processing to display the game, designed the levels of game.
- **Spandan Anaokar** Coded the Maze Ball game from scratch, set upn communication with laptop, optimized the code.

However, numerous aspects remained that necessitated further discussion among us, such as game mechanics, debugging, and parameter optimization.

8 References

- Arduino Language Reference
- Github link for gyroscope library
- https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/
- https://randomnerdtutorials.com/arduino-mpu-6050-accelerometer-gyroscope/
- Processing Reference

9 Appendix

The following is the code for Arduino IDE and Processing Software

9.1 Arduino Code

```
#include <MPU6050_tockn.h>
#include <Wire.h>

// Define a set of parameters that manipulate the entire working of the game itself (level means starting level)
int lim = 6;
int updaterate = 5;
float scalea = 1;
float scalev = 2;
int level = 1;
bool cheatMode = false;


// Define the struct that are used to create the Game Elements
typedef struct Wall{
    int startx;
    int starty;
    int endx;
    int endy;
      bool vert;
}Wall;

typedef struct Ball{
    float cenx;
    float ceny;
    int r;
    float vx;
    float vy;
}Ball;

typedef struct Hole{
    int x;
    int y;
    int r;
}Hole;

//Update the ball position depending on the acceleration(x & y)
void Ball_update(Ball &b, float x, float y)
{
    b.vx+=x/scalea;
    b.vy+=y/scalea;
    if(b.vx<-lim)
        b.vx=-lim;
    if(b.vy<-lim)
        b.vy=-lim;
    if(b.vx>lim)
        b.vx=lim;
    if(b.vy>lim)
        b.vy=lim;

    b.cenx+=b.vx/scalev;
    b.ceny+=b.vy/scalev;
}
```

```
 //Check if the ball reaches the Goal (Hard condition is the 2 circles intersecting)
bool checkGoal(Ball ball, Hole goal){

    if(((long)ball.cenx-goal.x)*((long)ball.cenx-goal.x)+((long)ball.ceny-goal.y)*((long)ball.ceny-goal.y)< ((long)goal.r+ball.r)*((long)goal.r+ball.r) ){
       return true;
    }
    else
       return false;
}

//Check if the ball touches the hole  (Hard condition is the 2 circles intersecting)
bool checkHole(Ball ball, Hole hole){
    if(((long)ball.cenx-hole.x)*((long)ball.cenx-hole.x) + ((long)ball.ceny-hole.y)*((long)ball.ceny-hole.y) < ((long)hole.r+ball.r)*((long)hole.r+ball.r)){
        return true;
    }
    else
        return false;
}

//Check if the ball crosses the wall. If it does then move the ball to just touching wall
void checkWall(Ball &ball, Wall w){
    float sc = 0;
    if(w.vert)
    {
        if(ball.ceny+ball.r>w.starty && ball.ceny-ball.r<w.endy && (ball.cenx + ball.r>=w.startx && ball.cenx - ball.r <= w.startx))
        {
            if(ball.cenx>=w.startx)
            {
                ball.cenx = w.startx+ball.r;//+2;
                ball.vx=-ball.vx*sc;///30;
            }
            else
            {
                ball.cenx = w.startx-ball.r;//-2;
                ball.vx=-ball.vx*sc;///30;
            }
        }
    }
    else
    {
        if(ball.cenx+ball.r>w.startx && ball.cenx-ball.r<w.endx && (ball.ceny + ball.r >= w.starty && ball.ceny - ball.r <= w.starty))
        {
            if(ball.ceny>=w.starty)
            {
                ball.ceny = w.starty+ball.r;//+2;
                ball.vy=-ball.vy*sc;///30;
            }
            else
```

```
            {
                ball.ceny = w.starty-ball.r;//-2;
                ball.vy=-ball.vy*sc;///30;
            }
        }
    }

}

//Send all display data as a long list of integers to the Serial
    Port
void display2(Ball b, Hole g, Wall w[], unsigned int wlen, Hole h[],
    unsigned int hlen){
    Serial.print("0 ");
    Serial.print(wlen); Serial.print(" ");
    Serial.print(hlen); Serial.print(" ");

    Serial.print((int)b.cenx); Serial.print(" ");
    Serial.print((int)b.ceny); Serial.print(" ");
    Serial.print(b.r); Serial.print(" ");

    Serial.print(g.x); Serial.print(" ");
    Serial.print(g.y); Serial.print(" ");
    Serial.print(g.r); Serial.print(" ");

    for(unsigned int i=0; i<hlen; i++){
        Serial.print(h[i].x); Serial.print(" ");
        Serial.print(h[i].y); Serial.print(" ");
        Serial.print(h[i].r); Serial.print(" ");
    }

    for(unsigned int i=0; i<wlen; i++){
        Serial.print(w[i].startx); Serial.print(" ");
        Serial.print(w[i].starty); Serial.print(" ");
        Serial.print(w[i].endx); Serial.print(" ");
        Serial.print(w[i].endy); Serial.print(" ");
    }

    Serial.println("");
}

//Initialize the game elements
Hole g;
Ball b;
int hlen = 0;
int wlen = 0;
Hole h[27];
Wall w[20];

//Update the game elements depending on the level
void levelchange(int l){
    if(l==1){
        // Level 1
        b = (Ball){25, 50, 10, 0, 3};
        g = (Hole){140, 225, 15};
        h[0] = (Hole){275, 150, 20};
        h[1] = (Hole){25, 250, 20};
        h[2] = (Hole){160, 100, 20};
        h[3] = (Hole){85, 260, 15};

```

```c
        w[0] = (Wall){0, 0, 0, 300, 1};
        w[1] = (Wall){300, 0, 300, 300, 1};
        w[2] = (Wall){0, 0, 300, 0, 0};
        w[3] = (Wall){0, 300, 300, 300, 0};
        w[4] = (Wall){0, 75, 200, 75, 0};
        w[5] = (Wall){50, 150, 50, 300, 1};
        w[6] = (Wall){200, 75, 200, 150, 1};
        w[7] = (Wall){100, 100, 100, 200, 1};
        w[8] = (Wall){100, 200, 300, 200, 0};
        w[9] = (Wall){250, 75, 250, 200, 1};
        w[10] = (Wall){100, 200, 100, 250, 1};
        w[11] = (Wall){100, 250, 275, 250, 0};
        w[12] = (Wall){50, 25, 50, 75, 1};
        hlen = 4;
        wlen = 13;
        updaterate = 5;
    }

    else if(l==2){
        // Level 2
        b = (Ball){35, 225, 10};
        g = (Hole){275, 130, 15};
        h[0] = (Hole){125, 280, 15};
        h[1] = (Hole){175, 220, 15};
        h[2] = (Hole){275, 275, 15};
        h[3] = (Hole){150, 185, 12};
        h[4] = (Hole){20, 125, 15};
        h[5] = (Hole){75, 60, 15};
        h[6] = (Hole){150, 75, 15};
        h[7] = (Hole){210, 135, 10};
        h[8] = (Hole){280, 70, 15};

        w[0] = (Wall){0, 0, 0, 300, 1};
        w[1] = (Wall){300, 0, 300, 300, 1};
        w[2] = (Wall){0, 0, 300, 0, 0};
        w[3] = (Wall){0, 300, 300, 300, 0};
        w[4] = (Wall){0, 200, 225, 200, 0};
        w[5] = (Wall){50, 200, 50, 275, 1};
        w[6] = (Wall){75, 150, 300, 150, 0};
        w[7] = (Wall){75, 80, 75, 150, 1};
        w[8] = (Wall){150, 0, 150, 60, 1};
        w[9] = (Wall){225, 60, 225, 150, 1};
        hlen = 9;
        wlen = 10;
        updaterate = 5;
    }

    else if(l==3){
        // Level 3

        b = (Ball){215, 160, 7};
        g = (Hole){212, 210, 6};

        h[0] = (Hole){150, 140, 7};
        h[1] = (Hole){213, 115, 5};
        h[2] = (Hole){210, 90, 10};
        h[3] = (Hole){175, 85, 5};
        h[4] = (Hole){150, 82, 5};
        h[5] = (Hole){130, 85, 5};
        h[6] = (Hole){90, 90, 12};
```

```
        h[7] = (Hole){85, 130, 7};
        h[8] = (Hole){100, 200, 15};
        h[9] = (Hole){260, 115, 10};
        h[10] = (Hole){235, 60, 10};
        h[11] = (Hole){165, 10, 10};
        h[12] = (Hole){165, 65, 10};
        h[13] = (Hole){145, 10, 10};
        h[14] = (Hole){145, 65, 10};
        h[15] = (Hole){35, 10, 10};
        h[16] = (Hole){35, 80, 10};
        h[17] = (Hole){10, 137, 10};
        h[18] = (Hole){65, 210, 10};
        h[19] = (Hole){70, 240, 10};
        h[20] = (Hole){100, 290, 10};
        h[21] = (Hole){130, 240, 10};
        h[22] = (Hole){160, 290, 10};
        h[23] = (Hole){190, 240, 10};
        h[24] = (Hole){220, 290, 10};
        h[25] = (Hole){250, 240, 10};
        h[26]= (Hole){280, 290, 10};
        w[0] = (Wall){0, 0, 0, 300, 1};
        w[1] = (Wall){300, 0, 300, 300, 1};
        w[2] = (Wall){0, 0, 300, 0, 0};
        w[3] = (Wall){0, 300, 300, 300, 0};
        w[4] = (Wall){200, 150, 222, 150, 0};
        w[5] = (Wall){150, 150, 150, 172, 1};
        w[6] = (Wall){150, 175, 272, 175, 0};
        w[7] = (Wall){225, 75, 225, 175, 1};
        w[8] = (Wall){75, 75, 225, 75, 0};
        w[9] = (Wall){75, 75, 75, 95, 1};
        w[10] = (Wall){30, 225, 200, 225, 0};
        w[11] = (Wall){230, 225, 300, 225, 0};
        w[12] = (Wall){275, 150, 275, 175, 1};
        w[13] = (Wall){250, 100, 300, 100, 0};
        w[14] = (Wall){85, 30, 85, 75, 1};
        w[15] = (Wall){0, 150, 50, 150, 0};
        w[16] = (Wall){193, 200, 232, 200, 0};
        w[17] = (Wall){190, 200, 190, 225, 1};
        w[18] = (Wall){235, 200, 235, 225, 1};
        w[19] = (Wall){75, 130, 75, 225, 1};

        hlen = 27;
        wlen = 20;
        updaterate = 3;
    }
}

//Create animation that after level ends ball moves to centre of
    respective goal or hole
void ending(Ball &b, Hole hol)
{
    int dx = hol.x-b.cenx, dy = hol.y-b.ceny;
    int sx = b.cenx, sy=b.ceny;
    for(int j=0; j<8; j++)
    {
        b.cenx = sx+(double) dx*j/8;
        b.ceny = sy+(double) dy*j/8;
        display2(b, g, w, wlen, h, hlen);
        delay(5);
    }
```

```
}

//Initialize the MPU6050 communincation and define global variables
MPU6050 mpu6050(Wire);
double accx = 0;
double accy = 0;
unsigned int n = 0;
char val;

//We initialize the Serial(comm. with laptop) and Wire(comm. with
   MPU6050)
//Also start the gyroscope and calibrate it. At end we change the
   level
//and send coded message to signal begining of gaming loop
void setup() {
    Serial.begin(115200);
    Wire.begin();
    mpu6050.begin();
    mpu6050.calcGyroOffsets(true);
    Serial.println();
    Serial.println("TheWorldEatingPythonIsBack");
    levelchange(level);
}

//Gaming loop begins
void loop() {
    //Increment the n->no. of loops before display update
    //Also calculate the acceleration values that the ball follows
    n+=1;
    mpu6050.update();
    accx = 10*sin(2*3.141*mpu6050.getAngleX()/360.0)+0.2;
    accy = 10*sin(2*3.141*mpu6050.getAngleY()/360.0)+1.76;

    //Conduct check if Display device sends any information. If it
    does do the corresponding action
    if (Serial.available())
    { // If data is available to read,
        val = Serial.read(); // read it and store it in val
        //For 1, 2 & 3 change the level to that number
        if(val=='1')
        {
            level=1;
            levelchange(level);
        }
        if(val=='2')
        {
            level=2;
            levelchange(level);
        }
        if(val=='3')
        {
            level=3;
            levelchange(level);
        }
        //Toggle cheat mode -> Ball passes through all walls and is
    impervious to holes
        if(val=='c')
        {
            cheatMode = !cheatMode;
        }
```

```
        //Clear the Serial buffer
        while(Serial.available())
            Serial.read();
    }

    //Update the game parameters after every updaterate no of loops take place
    if(n%updaterate==0)
    {
        //Update the ball position
        Ball_update(b, accy, accx);

        //Check if goal reached
        if(checkGoal(b,g)){
            ending(b, g);

            //Send output denoting win
            if(level==3)
                Serial.println("1 2");
            else
                Serial.println("1 1");

            //Wait for response from Display device. Depending on it restart or change level
            while(!Serial.available()){}

            if (Serial.available())
            { // If data is available to read,
                val = Serial.read(); // read it and store it in val
            }
            if (val == 'n')
                level = (level)%3+1;
            levelchange(level);
        }

        //Check if any hole is reached
        for (int i=0; i<hlen && !cheatMode; i++){
            if(checkHole(b, h[i])){
                ending(b, h[i]);
                //Send loss result to display device
                Serial.println("1 0");
                //Wait for response
                while(!Serial.available()){}
                //If response comes then restart level
                if (Serial.available())
                { // If data is available to read,
                    val = Serial.read(); // read it and store it in val
                }
                levelchange(level);
            }
        }

        //Check and update ball for all walls
        for(int i=0; i<wlen && !cheatMode; i++){
            checkWall(b, w[i]);
        }

        display2(b, g, w, wlen, h, hlen);

```

```
        //For debugging purposes:
        //Serial.print(accx); Serial.print(" ");Serial.print(accy); Serial.print(" ");Serial.print(b.vx); Serial.print(" ");Serial.print(b.vy); Serial.print(" ");Serial.print(b.cenx); Serial.print(" ");Serial.println(b.ceny);
        n=0; //Reset the count of no of loops after last game update
    }
}
```

9.2 Processing Code

```

import processing.serial.*;
import uibooster.*;

//Define the Parameters for the display and the global variables
Serial myPort;
String val;      //Store the value gotten from Serial
boolean started = false;  //If gaming loop has started
int scale = 3;  //Screen size (default is 300 *300) NOTE: Also cahnge size accordingly

void setup()
{
  size(900, 900);  //Update it as per 300*scale, 300*scale
  String portName = Serial.list()[0]; //change the 0 to a 1 or 2 etc. to match your port
  myPort = new Serial(this, portName, 115200);
  noCursor();
  background(255, 255, 255);
  //Define background color
}

int[] state = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
//int[] state = {0, 13, 4, 25, 50, 10, 140, 225, 15, 275, 150, 20,
    25, 250, 20, 160, 100, 20, 85, 260, 15, 0, 0, 0, 300, 300, 0,
    300, 300, 0, 0, 300, 0, 0, 300, 300, 300, 0, 75, 200, 75, 250,
    75, 250, 200, 50, 150, 50, 300, 200, 75, 200, 150, 100, 100, 100,
     200, 100, 200, 300, 200, 100, 200, 100, 250, 100, 250, 275, 250,
     50, 25, 50, 75};
//int[] state = {0, 10, 9, 35, 225, 10, 275, 130, 15, 125, 280, 15,
    175, 220, 15, 275, 275, 15, 150, 185, 12, 20, 125, 15, 75, 60,
    15, 150, 75, 15, 210, 135, 10, 280, 70, 15, 0, 0, 0, 300, 300, 0,
     300, 300, 0, 0, 300, 0, 0, 300, 300, 300, 0, 200, 225, 200, 50,
    200, 50, 275, 75, 150, 300, 150, 75, 80, 75, 150, 150, 0, 150,
    60, 225, 60, 225, 150};
//int[] state = {0, 20, 27, 215, 160, 7, 212, 210, 6, 150, 140, 7,
    213, 115, 5, 210, 90, 10, 175, 85, 5, 150, 82, 5, 130, 85, 5, 90,
     90, 12, 85, 130, 7, 100, 200, 15, 260, 115, 10, 235, 60, 10,
    165, 10, 10, 165, 65, 10, 145, 10, 10, 145, 65, 10, 35, 10, 10,
    35, 80, 10, 10, 137, 10, 65, 210, 10, 70, 240, 10, 100, 290, 10,
```

```
    130, 240, 10, 160, 290, 10, 190, 240, 10, 220, 290, 10, 250, 240,
     10, 280, 290, 10, 0, 0, 0, 300, 300, 0, 300, 300, 0, 0, 300, 0,
    0, 300, 300, 300, 200, 150, 222, 150, 150, 150, 150, 172, 150,
    175, 272, 175, 225, 75, 225, 175, 75, 75, 225, 75, 75, 75, 75,
    95, 30, 225, 200, 225, 230, 225, 300, 225, 275, 150, 275, 175,
    250, 100, 300, 100, 85, 30, 85, 75, 0, 150, 50, 150, 193, 200,
    232, 200, 190, 200, 190, 225, 235, 200, 235, 225, 75, 130, 75,
    225};
//We start with a blank display but can modify it to start with level1, level2, level3

void disp()
{
    int wlen = state[1];
    int hlen = state[2];

    background(255, 255, 255);

    //Goal
    fill(0, 150, 0);
    circle(state[6]*scale, state[7]*scale, 2*state[8]*scale);

    //Holes
    fill(255, 51, 51);
    for(int i=9; i<9+3*hlen; i+=3){
      circle(state[i]*scale, state[i+1]*scale, 2*state[i+2]*scale);
    }

    //Walls
    stroke(0);
    strokeWeight(3*scale);
    for(int i=9+3*hlen; i<9+3*hlen+4*wlen; i+=4){
      line(state[i]*scale, state[i+1]*scale, state[i+2]*scale, state[i+3]*scale);
    }

    //Ball
    strokeWeight(1*scale);
    fill(161, 10, 216);
    circle(state[3]*scale, state[4]*scale, 2*state[5]*scale);
}

void draw()
{
  if ( myPort.available() > 0)
  {   //If data is available,
    val = myPort.readStringUntil('\n');         // read it and store it in val
    println(val);
    //Wait for the code word to come -> denotes starting of game loop
    if(val!=null && val.contains("TheWorldEatingPythonIsBack"))
      started=true;
    else if(val!=null && started==true) //If the value is not null then we take it into consideration
    {
      //Starts capturing the display from the Arduino
      int[] data= int(trim(split(trim(val), ' '))); //Make an array consisting of integer from all data
      if(data[0]==1) //Level is over
```

```
      {
        //If level is won or lose take corresponding response
   through dialogue box
        print(data);
        if(data[1]==1)
          new UiBooster().showConfirmDialog(
                    "Do you want to go to the next level(else retry)?"
   ,
                    "Congratulations. You have passed the level.",
                    () -> myPort.write('n'),
                    () -> myPort.write('r'));
        if(data[1]==0)
          new UiBooster().showConfirmDialog(
                    "Do you want to retry(else quit)?",
                    "Bad Luck. You have Lost",
                    () -> myPort.write('r'),
                    () -> exit());
        if(data[1]==2)
          new UiBooster().showConfirmDialog(
                    "Do you want to start again(else quit)?",
                    "Congratulations. You have Won",
                    () ->  myPort.write('n'),
                    () -> exit());
      }
      else if(data[0]==0)
      //If game is continuing just update all parameters
      {
        for(int i=0; i<data.length; i++)
          state[i]=data[i];
      }
    }
    // Irrespective of whether the output is null or not continue to
    update the display depending on state array
    disp();
  }

  //Send Serial info to arduino so that we can change levels 1, 2, 3
    or activate cheatMode
  if(keyPressed)
  {
    //myPort.write(key);
    if(key=='1')
      myPort.write('1');
     if(key=='2')
      myPort.write('2');
     if(key=='3')
      myPort.write('3');
     if(key=='c')
      myPort.write('c');
  }
}
```

9.3 Links

Demo Video

Github Repositary