CONTENT

PONG: PROJECT 23

ELEC3607: MILESTONE IV

TEAM MEMBERS

NAME	STUDENT ID	UNIKEY
SALEEM GAGGUTURU	450 114 711	SGAG4244
SHUBHKARAN SINGH	450 226 003	SSIN4752

1. PROBLEM

The aim of this project is to design a 2D pong game, on the Arduino DUE. This game will be displayed on a 2.8” TFT LCD screen connected to the Arduino board. There will be two players associated with the game: the human player, and the computer (AI). The players have the ability to move left or right. The movement of the human player is determined by the embedded accelerometer of your Android phone, such that the data will be transmitted over to the Arduino via Bluetooth. Other components are also used to make the game more interactive, such as buttons (to move states) or the potentiometers (to adjust the settings of the game). All of these individual components will be analyzed in detail later in this report.

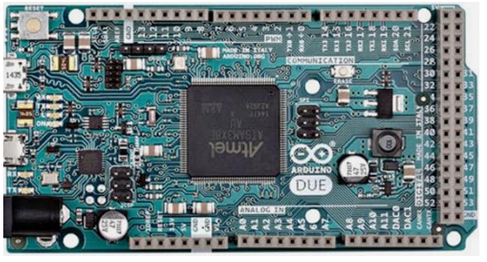
2. DESIGN

Our design of the pong game will consist of a LCD screen with a resolution of 240 by 320 pixels and 2.8” screen - this is large enough to play an optimal game of pong. The LCD connects to the Arduino microcontroller which renders the pixels on the screen as well as calculating the position of the ball and player, updating it frequently such that it appears as normal motion to the human eye. The movement of the human player is determined via the Android app and received through the Bluetooth module connected to the Arduino. The phone acts as the controller for the human player, its output being the tilt of the player on the LCD screen. Once a game has finished (one of the players have reached the score limit), then the Bluetooth module will be used to transmit the games outcome to the connected master device. The game could then be started again via the button connected to the Arduino. The app is also used to keep the details of all the previous matches played. These details include; your score, the AI score, the outcome of the game, and the date and time of the game played. These match histories will be saved in your phone internally, so it will remain there until you clear the items or uninstall the app.

LCD Screen

The LCD screen is used for rendering the pong game, it is connected to the Arduino and is powered with 5v power supply of the board. The screen has a resolution of 240X320 pixels and has dimensions 68.6mm X 53.3mm X 1.6mm. The screen is used in conjunction with the UTFT library to update each pixel on the screen in real-time.

Arduino Due



The microcontroller of the system, Arduino Due with an ARM SAM3X chip is used to perform all the calculations of the system. The Due runs at 84Mhz clock which is fast enough for it to process the pixels on the LCD in time for it to look smooth to the human eye. The Due is also responsible for communicating with the Bluetooth module on Serial 2 and processing the position of the player at certain intervals as well as processing the data that needs to be sent via Bluetooth.

Bluetooth Module

The Bluetooth module, which will be connected to the Arduino, will be used for wireless communication between an android phone and the board. For the Bluetooth module to talk to the Arduino via Serial2, the TX and RX of the module are connected to pins 16 and 17 of the board.

Android Application

The application installed on the phone is used to determine the position of the human player on the LCD screen. This is done by using the phone's on board 3D accelerometer to measure the change in position, then send the deviation via Bluetooth. Rotating the device clockwise will generate a positive number between 0 to 10, and rotating it counter clockwise will generate a negative number between 0 and –10. This process is explained later in this report. The interval at which the transmission occurs is changeable from inside the app by toggling the volume up and down button. The app is also used to show the details of all the matches played, once a match is finished a new entry is created which includes the following details; player score, the AI score, the outcome of the game, and the date and time of the game.

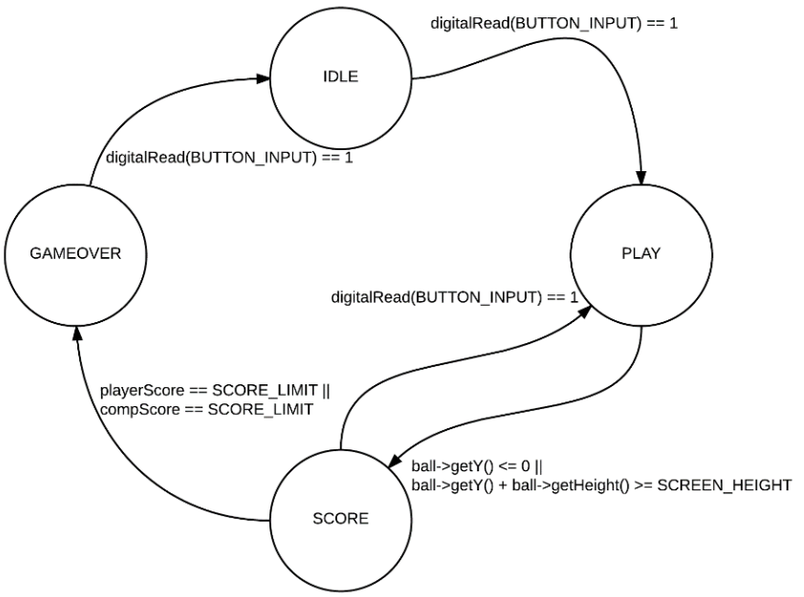
Other Components

Potentiometer: will be connected to the Arduino. This will be used for 2 functions, to increase the speed of the ball and increase the difficulty of the game, hence we'll have to make use of 2 potentiometers. The 2 potentiometers will be connected to analog pins 8 & 9 on the Due.

Button: A button will be used to start/continue the game. The button will be connected to digital Pin 6 on the Arduino, as well as an it being connected to the GND to complete circuit.

3. HOW IT WORKS

This embedded system works primarily by executing instructions in the relevant current *state* of the game. The states of this Pong game are as follows: IDLE, PLAY, SCORE and GAMEOVER. To transition between states, different conditions have been met. These conditions have been summarized in the state diagram below.



As you can see, to progress from IDLE state to the PLAY state, a signal in BUTTON_INPUT will need to be made. In doing so, the current state will change to the Play State.

To exit the PLAY state, the ball must exit the screen's bounds. This indicates either the human or the AI player has won. Once this is established, you the current state will transition into the SCORE state.

The score state is responsible to show the current scores of the game. If the score limit has been reached, the current state will transition into the GAMEOVER state. Otherwise, it will wait in this state until BUTTION_INPUT has been pressed.

Finally, in the GAMEOVER state, the outcome of the game will be displayed in the screen, and will also be sent to the Bluetooth master device. You can return to the IDLE state once the BUTTON_INPUT has been pressed.

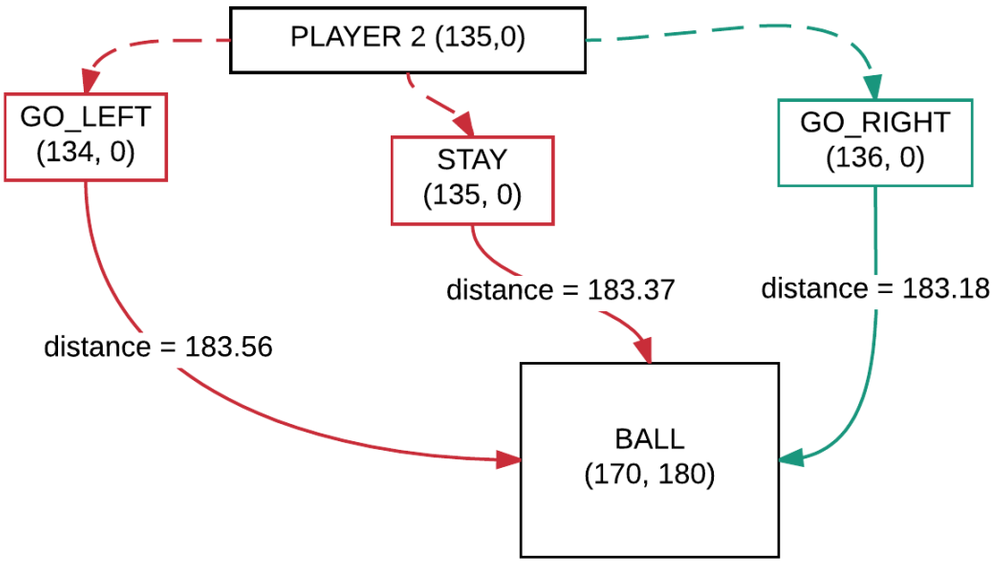
IDLE STATE

The IDLE state is the most simplistic state: the screen renderer prints "Welcome to PONG" in the middle of the screen, followed by "press start to play". This latter text will be changing colours from 0x0 (black) to 0xFFFFFF (white). In addition, this state is also responsible for resetting the scores of the players. This IDLE state will remain IDLE until there is an input in BUTTON_PIN input.

PLAY STATE

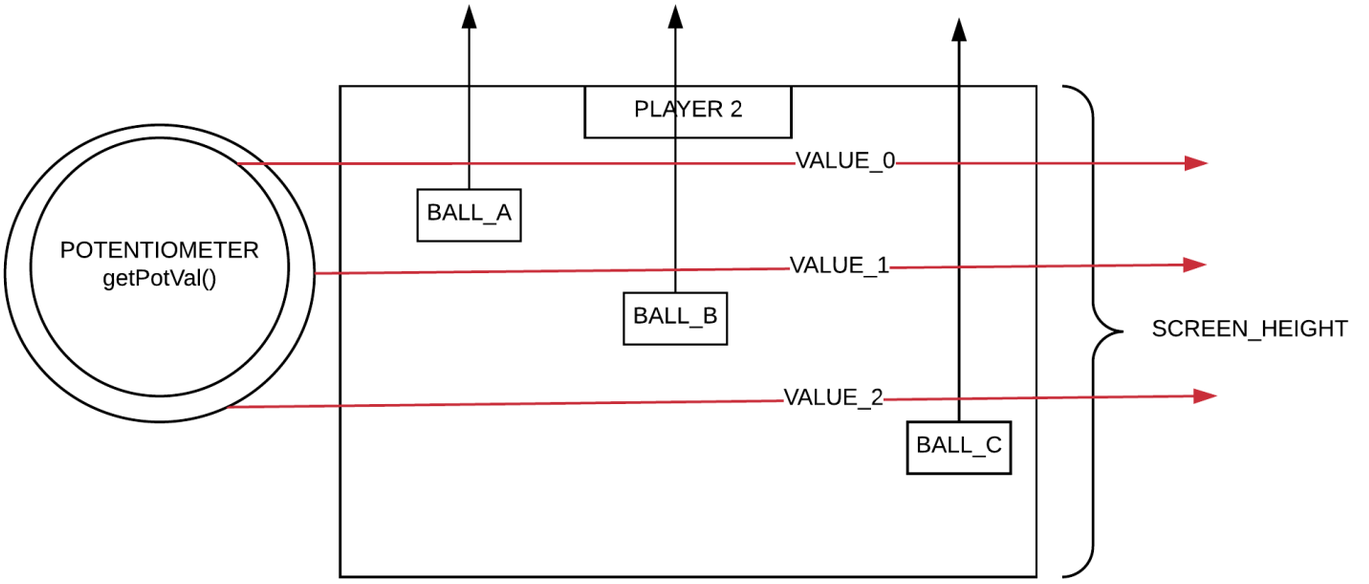
This is the state where all the action occurs. This state is responsible of making the players and the ball move. The first instruction of this state will call the `updateTilt()` method, which retrieves the degree of rotation from the master Android device. The next instructions will check whether the ball has hit the left and right borders of the screen, and if so, will bounce the ball. It will also check, afterwards, whether the ball touched the top and bottom borders of the screen, in which case, the current round is over as one of the players have won: the current game state will be transitioned to the SCORE state and the relevant players will have their scores incremented.

The next process in this state involves the AI player to choose the optimal decision to win the game. Its choices are defined in an enumerated object: GO_LEFT, GO_RIGHT and STAY. The optimal move can be found by calling the Rectangle's `directionTowards(Rectangle*)` member method, which will determine what to do next. This function has been experimented heavily to find the most efficient algorithm to choose its movement. After extensive research, we've decided to use the **Greedy Search algorithm** to control the behavior of the computer. The algorithm is very simple but is guaranteed to find an optimal solution. It works by calculating the Euclidean distance between the ball if it were to go left, stay or go right, and chooses the minimum distance. This diagram describes this process:



Assume that PLAYER 2 (AI) is currently on position (135, 0) and the ball is on (170, 180). If the AI is to go left, then its X coordinate will be subtracted by one, i.e., its new X position will be (134, 0). You can calculate the Euclidean distance between this and the ball, which turns out to be 183.56. Similarly, you repeat the steps for staying in the current position (X + 0, 0) and going right, (X + 1, 0). As can be seen in the diagram, the distance between player 2 and the ball is minimum if you were to go right (highlighted in green), therefore, the player will move right.

However, using this approach will mean the AI will never lose, because the greedy search is complete and optimal. Therefore, to allow the human player (player 1) to win, a potentiometer is used to adjust the vision of player 2. This simple algorithm is described in this diagram below:



Assume there are three balls on the screen, and all these balls are approaching player 2. The potentiometer has a `getPotVal()` method, which returns an integer between 0 and SCREEN_HEIGHT – 1, depending on how much you adjust the input – where the maximum is SCREEN_HEIGHT – 1 and minimum is 0. For demonstrating purposes, three example values (VALUE_0, VALUE_1 and VALUE_2) have been noted in the diagram. These values represent how far the player 2 can see. If the player can't see the ball in this range, it will not be able to move, i.e., `directionTowards(Rectangle*)` will always return STAY. If the value of `getPotVal()` is VALUE_0, for example, the player 2 will not be able to see any of the balls, therefore, will not react. However, if the value is VALUE_2, the player can see both BALL_A and BALL_B (but not BALL_C). In doing so, you can adjust the difficulty of the game because if the value is low, say VALUE_0, the computer will not be able to respond fast enough to see the ball, therefore, will lose. It is, however, worth noting, that the minimum value is not zero, but is DIFF_LEVEL, which is defined in the constants file. This is just to add some challenge to the game, otherwise, it's no fun at all!

Finally, the last few instructions of this state will move the player 1 based off the 3D accelerometer data provided by the Android device, where tilting the device at a greater angle will move the player faster towards the appropriate direction, and vice versa. Finally, the code checks whether the ball collides with the players, and if so, make a bounce effect, and move the ball.

SCORE STATE

The goal of the score state is to display the scores and return to the play state after input has been observed. This state will first reset the positions of the rectangles, by moving them to their initial positions. Next, the instructions check whether the score has been reached, and if so, transition to the GAME OVER state. If not, the score will be displayed on the screen until the button has been clicked.

GAMEOVER STATE

The LCD screen will display "GAME OVER!" and a win or lose string based on the outcome of the game. It will also send the score to the Bluetooth master device and wait for input to transition back to the IDLE state.

The Android App

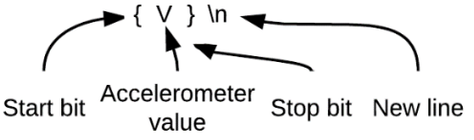
The layout of the Android app is most simple, as is illustrated in the screenshot below, but there are a bunch of classes working together to make it look appealing.

As is illustrated in this diagram, the main activity of this app will show you a list of match histories. As can be seen, this player has only won two out of the seven games played, so he/she obviously needs more practice!

The "CONNECT" button at the top action bar is what it says it is, that is: the application will search your paired Bluetooth devices in your device and try to connect to the relevant Bluetooth module. If it can't find it, a message will be displayed in the status text, allowing the user to make appropriated changes and reconsider what is going on.

After successful connection, the user can change the frequency of the transmission of the accelerometer value to the slave device by using the volume up and down to increase/decrease the delay. Moreover, upon successful connection, three additional threads will be initiated and will start running concurrently:

- The Bluetooth receiver thread, which is responsible for handling all data that is incoming from the slave device (the outcome of a match).
- The Bluetooth sender thread, will continually send the status of the 3D accelerometer to the slave device. This value will be in range of [-10, 10], where a positive 10 will mean the device is being rotated to the maximum counter-clockwise value. Furthermore, this value is encoded in a special string, where the rules are defined below:



As you can see, the accelerometer value is wrapped around curly braces. This is used to prevent garbage output being fed in the sense that the receiver will only need to consider the value inside these braces. If the string does not match this pattern, it will be dropped. This thread will sleep for exactly the amount of time based on the transmission delay. Upon waking up, it will retrieve the value of the accelerometer from its thread and send it to the slave device. This will repeat until the Bluetooth device is disconnected.

- The final thread is the 3D accelerometer thread. This thread is responsible to register the event handler which will get called approximately every 200 milliseconds. This handler will update the local variable of the accelerometer value. The Bluetooth sender thread can retrieve this value through a synchronized public member variable. This thread will eventually terminate after the Bluetooth device is disconnected.

The outcome of the game sent by the slave device will be stored in an SQLite database, thereby, the set of match histories will remain persistent in the end user's device - until of course, the user chooses to clear all records by using the *clear* button, or simply by uninstalling this application.

4. TESTING METHODOLOGY / RESULTS

To test the different aspects of our design, we employed various methods such as; printf() statements and serial monitor outputs to more sophisticated approaches like JUNIT testing. For the hardware testing, we mainly made use of the serial monitor to examine the output received from each individual component. The LCD screen didn't require any testing to be done since the open source UTFT library was being used to control the screen.

Bluetooth Module: The Bluetooth module was tested by first connecting the module on its own, sending it sample strings from a connected device and printing that string to the serial monitor. This was done to see what other data, if any at all, is being received with the sample strings. Once it was clear that only the string sent through the connected device is being received, we moved on to the next phase, testing the transmitter. This was also done in the same manner, by first only transmitting sample strings to the connected device and seeing their outputs, then adjusting to exclude any garbage values being transmitted.

Potentiometers and Button: were also tested using the old approach of using the serial monitor to visualize the values of the individual components.

To test the software side of this system, we had to rely on the limited debugging tools provided by the Arduino IDE. The only option we could use was outputting data to the Serial and analysing this through the Serial Monitor interface. For example, to see the output of one of the potentiometer, we have `Serial.println(analogRead(8))` in the `void loop()` method to analyse its output with extreme changes in input. However, using this approach is cumbersome when it comes to testing methods in the Rectangle class. As an example, the collidesWith(Rectangle*) method returns true if *this* Rectangle is adjacent, or intersects another Rectangle. This method needs to be most precise, especially with the boundary test cases and the corner cases. If this method is not written properly, the ball will go straight through the player objects or bounce before it even intersects with them! Therefore, to test this method, we've translated this method to a Java method to take advantage of the debugging tools higher programming languages offer.

C++

```
boolean Rectangle::collidesWith(Rectangle* r){
      return !(
        this->getX() > r->getX()+r->getWidth()     ||
        this->getX()+this->getWidth() < r->getX() ||
        this->getY() > r->getY()+r->getHeight()    ||
        this->getY()+this->getHeight() < r->getY()
     );
}
```

JAVA

```
boolean collidesWith(Rectangle r){
      return !(
        this.getX() >  r.getX()+r.getWidth()    ||
this.getX()+this.getWidth() < r.getX() ||
        this.getY() > r.getY()+r.getHeight()    ||
this.getY()+this.getHeight() < r.getY()
     );
}
```

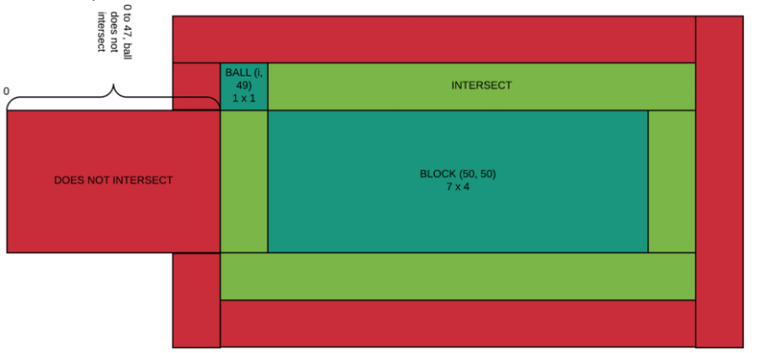
```
import static org.junit.Assert.*;
import org.junit.Test;

public class CollisionTest {

    Rectangle block, ball; // create the balls

    public CollisionTest() {
        // The constructor takes width and height
        block = new Rectangle(7, 4);
        block.x = 50;
        block.y = 50;
        ball = new Rectangle(1, 1);
    }1
    @Test
    public void testCollision() {
        /*Test the collides with method*/
        ball.y = 49;
        // from [0,48], the ball does NOT intersect the
block: ensure this
        for (int i = 0; i <= 48; i++) {
            ball.x = i;
            assertFalse(block.intersect(ball));
            assertFalse(ball.intersect(block));
        }
    }
}
```

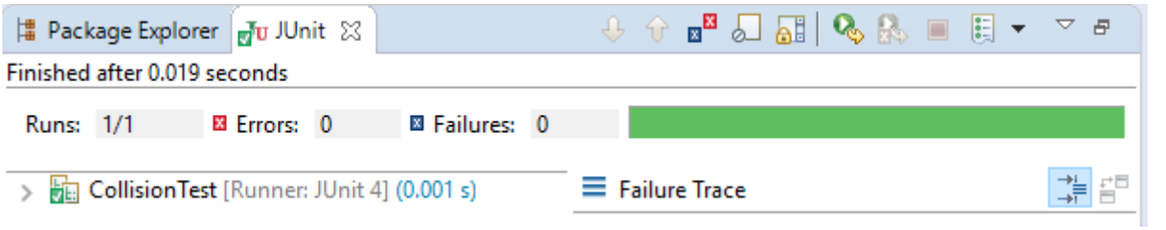
This, on the left block code, is used to automatically test the `collidesWith()` method. **In the constructor of the class will c**reate a new block and ball Rectangle, that we are going to use to test the collision.
The @Test annotation tells the compiler that we are testing this function, and therefore, the assert* methods can be used inside.



As can be seen in this diagram, the ball is in the top-left corner of the BLOCK, and we are testing from X: [0, 48] that the ball does NOT intersect with this block.
The green borders around the block indicate the positions where the `collidesWith()` method should return true.
This code snippet is just an excerpt from the full code, where in the full code, other functions are thoroughly tested.

Now with this, we can use JUNIT testing offered by Java to test that this method returns the value that we expect it to return.

You can run the test file simply by compiling and executing this Java class. Once done, the compiler will show whether the test passed or failed, without having to manually test it one by one. Here is a screenshot on how a successful test would look like:



As you can see, all the tests in the CollisionTest class have passed!

After testing all the hardware components and the software for the pong game, it was time to combine it all together and start fine tuning the design. After running the game for the first time, we immediately saw that the ball was being rendered too quickly, making it appear as though it was moving extremely fast. We decided to include an adjustable delay function using one of the potentiometers.

6. IMPROVEMENT

This game can be improved by adding more complexity to it to make the game more fun and challenging. As a start, there could be random blocks or barriers that appear in random positions on the screen and if the ball were to collide with them, it would bounce. In doing so, would make the game less predictable and makes the users more alert. Just adding a vast variety of uncertainty in the game will make the game more challenging and less boring.

Moreover, the users should be given more control of the game. Currently, they can only change the speed of the ball and the difficulty of the computer player. But allowing the users to play around with more settings will make the game more engaging. Some possibilities include controlling the acceleration of the ball, changing background colour of the game based on some sensors (such as light or noise) or even, allowing human against human (instead of human against AI).

Furthermore, the functionality of the Android app will need to be extended so the users can use it more. As a start, there should be a share widget that allows the users to share their match histories to social media or email. In addition, the Arduino slave only sends three bits to the master device (the human score, the AI score and a new line). It'll be most entertaining if it can send more data entries, such as the position of the objects and output of this screen: in this way, we can have a *spectator* display in the Android device that shows the live progress of the match. Finally, instead of saving the results of the game locally in the user's device, perhaps considering uploading it to a global shared server will be more ideal. This way, the users will be able to compare their scores to all those that played the game before; they can see where they stand compared to their peers across the globe.

7. References

a. "Library: UTFT" - Rinky-Dink Electronics, 10th June 2017
http://www.rinkydinkelectronics.com/library.php?id=51
b. "Android Documentation" – Google, 10th June 2017
https://developer.android.com

GitHub Link:

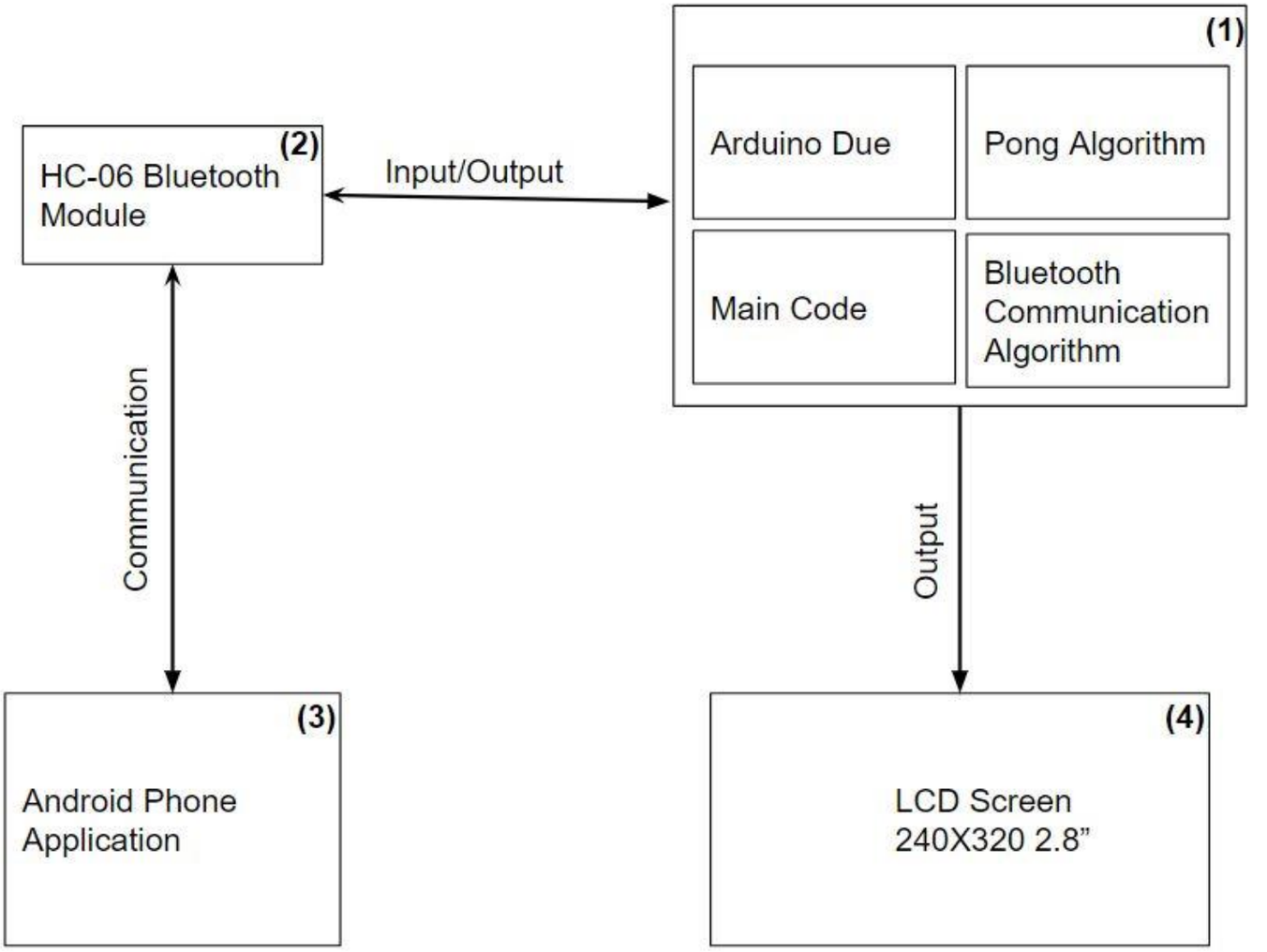
a. https://github.com/fatduckling/Pong

8. Summary Table

Category	List of items, with web link or page number where item is described
Open source compliance Are you publishing your code as open source? Explain what license you are using, or link to where you have published your project.	• We will be publishing our work as open source on github. https://github.com/fatduckling/Pong , we use the Creative Commons (CC BY-NC-SA 3.0) licensing for our project.
Platforms List the platforms you have used, including processor architecture (ARM, AVR, x86, etc.), programming languages (Python, C, C++, etc.), and IDEs. If you are not using Arduino Due, justify your choice of platform.	• We used Arduino DUE's SAM3X8E. • ARM Cortex M3 • We wrote C++ using the standard, official Arduino IDE and compiled Java code for the Android device using Google's Android Studio IDE. • All Junit testing was also written in Java, where the Eclipse IDE platform was used.
Sensors and inputs List your hardware inputs (push buttons, analog sensors, I2C or SPI sensors including accelerometers, etc.) and mention on which page each input is described.	• An Android device's inbuilt 3D accelerometer. (3) • Push Buttons. (3) • Potentiometers. (3)
Outputs List any mechanical, visual or other type of outputs controlled by your hardware (motors, magnets, LED, 7-Segs, LCD, etc.) and mention on which page each input is described.	• A LCD Screen. (2) • An Android device's screen. (6)
Connectivity List any protocols used for communicating between your main controller and any other microcontroller excluding sensors or actuators (USART, UART over USB, Bluetooth 2, Bluetooth 4, Wi-Fi, ZigBee, etc.) and mention on which page each input is described.	• Bluetooth 2
Graphical User Interface List any GUI used for interfacing users (local LCD, Phone app, Desktop app, Web app, Web dashboard, etc.) and mention on which page each input is described.	• LCD (2) • Phone app (6)
Algorithms / Logic List substantial control algorithms (state-machines, real-time operating system, filesystems, feedback algorithms, mathematical transformations, etc.) and mention on which page each input is described.	State machines (4) • Greedy search algorithm (4)
Physical case and other mechanical consideration List what casing or mechanical systems have been used in your project (3d prints, pipes, cardboard mechanics, etc.) and mention on which page each input is described.	

9. APPENDIX

BLOCK DIAGRAM



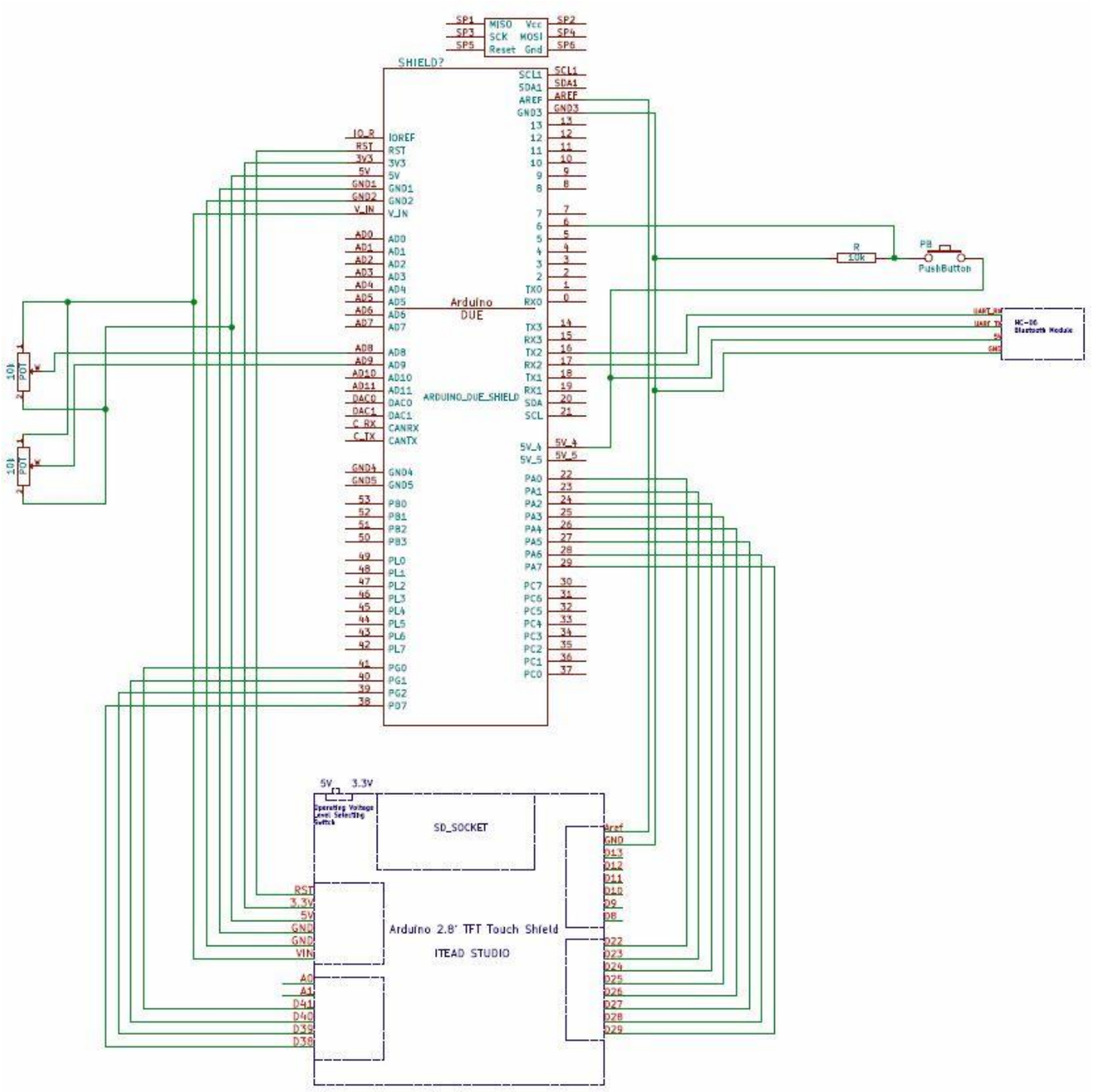
(1) **Arduino Due** with SAM3x microcontroller to do the calculations. **Pong algorithm** to light up the pixels correctly on the LCD screen. **Main Code** determining player position as well as balls position, speed and angle. **Bluetooth communication algorithm** is used to receive player 1’s movements from the phone app as well as send game data once the game is finished.

(2) **Bluetooth Module** used for wireless transfer of player scores data to the phone as well as receive player’s movements from the phone.

(3) **Android Phone App** communicates wirelessly with Bluetooth and views the game statistics as well as make use of the 3D Accelerometer on the phone for player 1 (human) movement and send it.

(4) **LCD Screen** to display the pong game.

HARDWARE SCHEMATIC DIAGRAM



ARDUINO CODE

PONG.INO

```
#include "constants.h"

#include "rectangle.h"

#include "utilities.h"

/* Create the objects to display in the screen */
Rectangle* player1;
Rectangle* player2;
Rectangle* ball;

// sore the current game state in this variable
```

```
GameStates currentGameState;

int playerScore = 0, // score for player 1
    compScore = 0;   // score for player 2


UTFT* renderer;
void setup() {
  // initialise random number generator
  randomSeed(analogRead(0));
  // set the baud rate for Serial2
  bluetooth.begin(9600);

  pinMode(BUTTON_INPUT,INPUT);

  // create the renderer
  renderer = new UTFT(ITDB28, 38, 39, 40, 41);

  //This will reset color to white with black background. Selected font will be reset
to none.
  renderer->InitLCD();
  //Clear the screen. The background-color will be set to black.
  renderer->clrScr();

  // create the players (player 1)
  player1 = new Rectangle(PLAYER_WIDTH, PLAYER_HEIGHT); // human player
  player1->setRenderer(renderer);
  // place player1 in the bottom-centre of the screen
  player1->setPosition(((SCREEN_WIDTH - player1->getWidth()) / 2), (SCREEN_HEIGHT -
player1->getHeight()));

  // create the computer player
  player2 = new Rectangle(PLAYER_WIDTH, PLAYER_HEIGHT); // computer
  player2->setRenderer(renderer);
  // place player 2 at the top-center of the screen
  player2->setPosition(((SCREEN_WIDTH - player1->getWidth()) / 2), 0);

  // create the ball
  ball = new Rectangle(BALL_RADIUS, BALL_RADIUS);
  ball->setRenderer(renderer);
  // place ball in the middle of the screen
  ball->setPosition(((SCREEN_WIDTH - ball->getWidth()) / 2), ((SCREEN_HEIGHT - ball-
>getHeight()) / 2));

  // current state of the game is IDLE
  currentGameState = IDLE;
}

void idleState() {
  /*
     Idle state: Just display a menu and do nothing until the start button is pressed;
     If the start button is pressed, go to the play state
  */

  // adjust and print font
  renderer->setFont(BigFont);
  renderer->setColor(VGA_GREEN);
  renderer->print("Welcome to", CENTER, 30);
  renderer->setColor(VGA_RED);
  renderer->print("PONG", CENTER, 50);
```

```
  // block until there is input from the button
  waitForInput(renderer, "Press start to play", CENTER, 180);

  // reset scores
  playerScore = 0;
  compScore = 0;

  // go to play state
  renderer->clrScr();
  currentGameState = PLAY;
}

// this is used to control the displacement of the ball
int speedX = 1, speedY = 1;
void playState() {
  /*
     This is the main state of the game where the user is playing pong against the AI;
     Once the score limit is reached, change the currentState to the gameOver state.
  */
  updateTilt();

  // check if the ball hits the left or right of the screen's borders
  if (ball->getX() <= 0 || ball->getX() + ball->getWidth() >= SCREEN_WIDTH) {
    speedX *= -1;
  }

  //check if the ball hit the top of the screen
  if (ball->getY() <= 0) {
    //player 1 has won this round; change state to score
    playerScore++;
    renderer->clrScr();
    currentGameState = SCORE;
    return;

  //check if the ball hits the bottom of the screen
  } else if (ball->getY() + ball->getHeight() >= SCREEN_HEIGHT) {
    compScore++; // increase AI score
    renderer->clrScr();
    currentGameState = SCORE;
    return;
  }

  // determine which way the computer should go to win
  switch (player2->directionTowards(ball)){
    case GO_LEFT: {
      player2->setPosition(player2->getX() - 1, player2->getY());
      break;
    }
    case GO_RIGHT: {
      player2->setPosition(player2->getX() + 1, player2->getY());
      break;
    }
    case STAY: { break; } // don't do anything
  }

  // get the value from the master device
  int tilt = getTilt();
  if (player1->getX() + tilt > 0 && player1->getX() + player1->getWidth() + tilt <
SCREEN_WIDTH) {
    // move player 1 based on this value
    player1->setPosition(player1->getX() + tilt, player1->getY());
```

```
  }

  // check if ball collides with players
  if (player1->collidesWith(ball) || player2->collidesWith(ball)){
    speedY *= -1;
  }

  // move the ball
  ball->setPosition(ball->getX() + speedX, ball->getY() + speedY);
}

void scoreState(){
  /* Displays the score;
     Compare the scores and check to see which one has won.
     Display the scores.
  */

  // reset the position of the ball and the players, to their initial state
  ball->setPosition(((SCREEN_WIDTH - ball->getWidth()) / 2), ((SCREEN_HEIGHT - ball-
>getHeight()) / 2));
  player2->setPosition(((SCREEN_WIDTH - player1->getWidth()) / 2), 0);
  player1->setPosition(((SCREEN_WIDTH - player1->getWidth()) / 2), (SCREEN_HEIGHT -
player1->getHeight()));

  // check if the score limit has been reached
  if(playerScore == SCORE_LIMIT || compScore == SCORE_LIMIT){
    renderer->clrScr();
    currentGameState = GAMEOVER;
    return;
  }

  renderer->setColor(VGA_WHITE);
  renderer->setFont(SmallFont);

  String score  = "Score: " + (String) playerScore + ":" + (String) compScore;
  renderer->print(score, CENTER, 80);

  // wait for the start button to be pressed
  waitForInput(renderer, "Press start to continue...", CENTER, 90);

  renderer->clrScr();
  currentGameState = PLAY;
}

void gameOverState() {
  /* Either human or computer has won!
   * Display who has won;
   * Send the scores to the bluetooth master device
  */

   renderer->setFont(BigFont);

   renderer->setColor(VGA_WHITE);
   renderer->print("GAME OVER!",CENTER,50);

   // see whether computer or human has won1
// pins
#define BUTTON_INPUT 6
#define GAME_FPS_POTENTIOMETER 9
#define THRESHOLD_POTENTIOMETER_PIN 8
```

```
// screen properties
#define SCREEN_WIDTH 320 // you can also do renderer->getDisplayXSize();
#define SCREEN_HEIGHT 240

// game states
enum GameStates { IDLE, PLAY, SCORE, GAMEOVER };

// fonts for UTFT renderer
extern uint8_t SevenSegNumFont[];
extern uint8_t BigFont[];
extern uint8_t SmallFont[];

#endif //CONSTANTS_H
```

POINT.H

```
#ifndef POINT_H

#define POINT_H

class Point {

public:
  // returns the euclidean distance between two points
  static double distance(Point* a, Point* b);

  Point(int x, int y);
  Point();
  int x, y;
};


#endif
```

POINT.CPP

```
#include "point.h"

#include "math.h"
/* A basicimplementation of a point class  */
double Point::distance(Point* a, Point* b){
  // calculate the Euclidean distance between two points
  return sqrt(((b->x - a->x) * (b->x - a->x)) + ((b->y - a->y) * (b->y - a->y)));
}

Point::Point(){} // empty constructor
Point::Point(int x1, int y1){
  x = x1;
  y = y1;
}
```

RECTANGLE.H

```
#ifndef RECTANGLE_H

#define RECTANGLE_H

#ifndef UTFT_h
// make to not include this library more than once
    #include <memorysaver.h>
```

```
    #include <UTFT.h>
#endif

#include "utilities.h"
#include "constants.h"
#include "point.h"

enum Directions {
  GO_LEFT, STAY, GO_RIGHT
};

class Rectangle {

  public:
    /* create a new rectangle of width "width" and height "height"
    OPTIONAL: specify its colour */
    Rectangle(int width, int height, int colour = 0xFFFFFF);

    // will re-draw the rectangle to its new position
    void update();

    // returns true if "this" rectangle intersects another rectangle
    boolean collidesWith(Rectangle * anotherObject);

    // returns true if "this" rectangle contains the point (px, py)
    boolean intersect(int px, int py);

    // moves the rectangle to a new position
    void setPosition(int x, int y);

    // attaches an UTFT renderer to enable drawing
    void setRenderer(UTFT* renderer);

    // only change X position; You'll also need to call update() to apply changes
    void setX(int x);

    // only change Y position; You'll also need to call update() to apply changes
    void setY(int y);

    // change colour of rectangle
    void setColour(int colour);

    /*
      If this Rectangle is blinded by the potentiometer value, it will return STAY;
      If this rectangle's X position is the same as the ball's X position, will also
return STAY;
      Otherwise, this will return GO_LEFT or GO_RIGHT depending on the position of the
this and the ball
    */
    Directions directionTowards(Rectangle * ball);

    /* Returns the center position of the rectangle */
    Point * getCenteredPosition();

    // returns the distance between "this" Rectangle and another Rectagle from its
center points
    double distanceCenteredDistance(Rectangle * anotherRectangle);

    // getter methods
    int getX();
    int getY();
```

```
    int getWidth();
    int getHeight();
    int getColour();

  private:
    int m_x, // xPosition
    m_y, // yPosition
    m_width, // height
    m_height, // width
    m_colour, // colour
    m_previousX, m_previousY; // previous x and y values (before update is called();

    UTFT * m_renderer; // renderer
};

#endif
```

RECTANGLE.CPP

```
#include "rectangle.h"


/* CONSTRUCTOR: set the width, height and colour variable. */
Rectangle::Rectangle(int width, int height, int colour): m_width(width),
m_height(height), m_colour(colour) {
  m_renderer = NULL;
  m_previousX = 0;
  m_previousY = 0;
}

/* Create a new rectangle at position (m_width, m_height);
  For the old rectangle at position (m_previousX, m_previousY), if any point in this
rectangle do not intersect with
  new rectangle, then colour it black.

  The other approach would be to paint the old rectangle black, and draw a new
rectangle. This method will not work because
  the rectangle's creation and deletion will cause it to flicker, and hence, is
unstable.
  Therefore, only colour the pixels black for those that are not part of the new
rectangle now
*/
void Rectangle::update() {
  if (m_renderer != NULL) {
    // draw the new rectangle
    m_renderer->setColor(this->getColour());
    m_renderer->fillRect(this->getX(), this->getY(), this->getX() + this->getWidth() -
1, this->getY() + this->getHeight() - 1);

    // loop through every point in the prevoius rectangle and paint it black it doesn't
intersect with the new rectangle
    m_renderer->setColor(VGA_BLACK);
    for (int y = m_previousY; y < m_previousY + m_height; y++) {
      for (int x = m_previousX; x < m_previousX + m_width; x++) {
        if (!intersect(x, y)) {
          m_renderer->drawPixel(x, y);
        }
      }
    }
  }
}
```

```
// returns true if "this" Rectangle intersects another Rectangle
boolean Rectangle::collidesWith(Rectangle* anotherObject) {
  return !(this->getX() > anotherObject->getX() + anotherObject->getWidth() || this->getX() + this->getWidth() < anotherObject->getX() ||
    this->getY() > anotherObject->getY() + anotherObject->getHeight() || this->getY() + this->getHeight() < anotherObject->getY());
}

// moves the rectangle to a new position (x, y)
void Rectangle::setPosition(int x, int y) {
  this->setX(x);
  this->setY(y);
  this->update();
}

// attaches an UTFT renderer
void Rectangle::setRenderer(UTFT* renderer) {
  m_renderer = renderer;
}

/* stores the prevoius X coordinate in a variable and updates the current coordinate */
void Rectangle::setX(int x) {
  m_previousX = m_x;
  m_x = x;
}

/* stores the prevoius Y coordinate in a variable and updates the current coordinate */
void Rectangle::setY(int y) {
  m_previousY = m_y;
  m_y = y;
}

// changes the colour of the rectangle
void Rectangle::setColour(int colour) {
  m_colour = colour;
  update();
}

// returns true if the point (px, py) is inside this rectangle
boolean Rectangle::intersect(int px, int py) {
  return px >= m_x && px < m_x + m_width && py >= m_y && py < m_y + m_height;
}


// gets the center of the rectangle and calculates the distance to another rectangle's center
double Rectangle::distanceCenteredDistance(Rectangle* anotherRectangle) {
  // get "this" Rectangle's center position
  Point* z = this->getCenteredPosition();
  // get "anotherRectangle"'s center position
  Point* y = anotherRectangle->getCenteredPosition();

  // calculate the distance between the two points
  double d = Point::distance(z, y);

  // delete the points (free some memory)
  delete(z);
  delete(y);

  return d;
```

```
}

/*
    This method will generate three nodes from the initial position of this rectangle.
    The first node will be the distance between this rectangle and 'anotherRectangle'
object, if this rectangle were to go left
    The second node will be the distance between this rectangle if it didn't move
    The third node will be the distance between this rectangle and 'anotherRectangle'
object, if this rectangle were to go right;

    Find the smallest distance and return the direction based off that
*/

Directions Rectangle::directionTowards(Rectangle* ball) {

  // if this rectangle can't see the ball, return the STAY direction
  if (ball->getY() > getPotVal()) {
    // if this rectangle can't see the ball, return the STAY direction
    this->setColour(VGA_RED); // change colour to red
    return STAY;
  }

  this->setColour(VGA_WHITE);

  int x = m_x;
  // caluclate the distance to the ball if the rectangle were not to move
  double stop = distanceCenteredDistance(ball);
  // calculate the distance to the ball if the rectangle were to go left
  double left;
  if (m_x > 0) {
    // move left one place and calculate the distance
    m_x -= 1;
    left = distanceCenteredDistance(ball);
  } else {
    // this rectangle is at the LEFT MOST position on the screen - you can't go any
more left
    left = INFINITY; // return a big number
  }

  // calcuate the distance to the ball if you were to the right
  double right;
  if (m_x + m_width < SCREEN_WIDTH) {
    // move right one place and calculate the distance
    m_x = x + 1;
    right = distanceCenteredDistance(ball);
  } else {
    // this rectangle is at the RIGHT MOST position on the screen - you can't go any
more right
    right = INFINITY;
  }

  // determine the minimum of the three points
  double tmp;
  Directions go;

  if (left < stop) {
    tmp = left;
    go = GO_LEFT;
  } else {
    tmp = stop;
    go = STAY;
```

```
  }
  if (right < tmp) {
    go = GO_RIGHT;
  }
  m_x = x;
  return go;
}

// returns the center point of "this" Rectangle
Point* Rectangle::getCenteredPosition() {
  Point* tmp = new Point();
  tmp->x = m_x + (m_width / 2);
  tmp->y = m_y + (m_height / 2);
  return tmp;
}

// getter methods
int Rectangle::getX() {
  return m_x;
}
int Rectangle::getY() {
  return m_y;
}
int Rectangle::getWidth() {
  return m_width;
}
int Rectangle::getHeight() {
  return m_height;
}
int Rectangle::getColour() {
  return m_colour;
}
```

UTILITIES.H

```
#ifndef UTILITIES_H

#define UTILITIES_H

#include "constants.h"

// return the accelerometer value from the master device
int getTilt();

// updates the accelerometer value from the master device
void updateTilt();

// returns the integer representation of a character. For example, '3' would return 3
inline int charToInt(const char c);

/* extracts the data from the master device; eg {5} will return 5;
   For any invalid data, returns INFINITY
*/
int extractTilt(const char* inputs, int size);

// returns true if the number is in range [-10, 10]
inline boolean isValidNum(int num);

// block execution until button has been pressed
void waitForInput(UTFT* renderer, String s, int x, int y, int deg = 0);
```

```
// send the outcome of the game to the master device
void sendScore(int score1,int score2);

// returns the current tilt value from the master device
int getPotVal();

#endif //UTILITIES_H
```

UTILITIES.CPP

```
#include "utilities.h"


#define INPUT_SIZE 7

int tilt = 0;

/* Returns a number from DIFF_LEVEL to SCREEN_HEIGHT - 1 based on the potentiometer pin
*/
int getPotVal(){
    int val = analogRead(THRESHOLD_POTENTIOMETER_PIN);
    //map(value, fromLow, fromHigh, toLow, toHigh)
    // remap "val" from (0 to 1023) to (DIFF_LEVEL to SCREEN_HEIGHT - 1)
    int mapped = map(val,0,1023,DIFF_LEVEL, SCREEN_HEIGHT - 1);
    return mapped;
}

// reached the \n, therefore, act upon this
void act(const char* inputs, const int size){
  // make sure the input string starts and ends with {} respectively
  if (inputs[0] != '{' || inputs[size - 1] != '}'){
    return; // garbage data
  }
  // convert the string to a float. For example, {123} would return 123.00
  int d = extractTilt(inputs, size);
  // invalid number
  if (!isValidNum(d)){
    return;
  }
  // valid number, update the tilt
  tilt = d;
}

/* extracts the data from the master device; eg {5} will return 5;
   For any invalid data, returns INFINITY
*/
int extractTilt(const char* inputs, int size){
  int tmp;
  switch (size){
    case 3: { // for exampe {5}
      tmp = charToInt(inputs[1]); // extract the number
      // if this number is valid, return it; otherwise return infinity
      return (isValidNum(tmp)) ? tmp : INFINITY;
    }
    case 4: { // for example {-3} or {10}
      tmp = charToInt(inputs[2]);
      if (inputs[1] == '-'){ // first character is a minus sign
        /* get the number in the braces and multiply it by -1; if this number
        is valid, then return it; otherwise, return INFINITY */
        return -1 * ((isValidNum(tmp)) ? tmp : INFINITY);
      }
```

```
      else {
        return 10; // only valid two digit number that is positive
      }
    }
    case 5: return -10; // only valid three digit number
  }
  return INFINITY; // invalid string
}

// returns true if this number is in range of [-10, 10]
inline boolean isValidNum(int num){
  return (num >= -10 && num <= 10);
}

// returns the integer representation of a character. For example, '3' would return 3
inline int charToInt(const char c){
  return (((int) c) - 48);
}

/*
  Read incoming byte from the master device (if such exist); if not, return;
  Add this byte to our "inputs" array; If we've reached the '\n', call method "act()"
*/
void updateTilt(){
  // create an array of inputs
  static char* inputs = NULL;
  // index of the character we are trying to insert
  static int index;
  if (inputs == NULL){
    inputs = (char*) malloc(INPUT_SIZE);
    inputs[INPUT_SIZE - 1] = '\0';
    index = 0;
  }

  int c;
  // read the incoming byte (if there are bytes incoming)
  while (c = bluetooth.read(), c >= 0) {
    if (c == '\n') {
      // we read the entire string, so act upon this
      act(inputs, index);
      // reset
      bluetooth.flush();
      index = 0;
    }
    // buffer full: string, reset
    else if (index == INPUT_SIZE - 1){
      // buffer full
      bluetooth.flush();
      index = 0;
    }
    // insert the new character to our input array
    else {
      inputs[index++] = char(c);
    }
  }
}

/* Continually check whether the button has been pressed;
   If it has not been pressed, print the string "s" at (x,y), rotated at degree "deg"
*/
void waitForInput(UTFT* renderer, String s, int x, int y, int deg){
```

```
  for (unsigned int i = 0; digitalRead(BUTTON_INPUT) != 1; i ++){
    renderer->setColor(i);
    renderer->print(s, CENTER, y, deg);
    if (i == 0xFFFFFF) {   // maximum
      i = 0;
    }
  }
}

// sends the outcome of the game to the master device
void sendScore(int playerScore, int aiScore){
  bluetooth.print(playerScore);
  bluetooth.println(aiScore);
}

// returns the tilt
int getTilt(){
  return tilt;
}
```

ANDROID (JAVA) CODE

1. **Layout of Main Activity class (activity_main.xml)**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/status_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:padding="5dp"
        android:layout_margin="0dp"
        android:text="Welcome to PONG!" />

    <android.support.v7.widget.RecyclerView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/match_history_recycler_view"
        android:scrollbars="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

2. MainActivity.java

```
package saleem.elec3607_pong;

import android.app.Activity;
import android.app.ProgressDialog;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class MainActivity extends Activity {
    static String DEVICE_NAME = "HC-06"; // name of the bluetooth device
    static int REQUEST_ENABLE_BT = 1;

    public Menu menu; // action bar menu
    private SQLiteDatabase db; // reference to the database
    private MatchHistoryAdapter adapter; // reference to the recycler view (to display the match histories)
    public RecyclerView recyclerView; // the view that displays all the match histories
    private BluetoothTransmission bluetoothTransmission;
    public SensorThread sensorThread;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // view has been created
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // create the database object
    DatabaseHelper databaseHelper = new DatabaseHelper(this);
    db = databaseHelper.getWritableDatabase();

    // create the query
    // select p1_score, p2_score, date from matches order by date desc
    Cursor cursor = db.query("matches",
            new String[]{"p1_score", "p2_score", "date"},
            null, null, null, null, "date desc");

    // create the recyclerview
    recyclerView = (RecyclerView) findViewById(R.id.match_history_recycler_view);
    adapter = new MatchHistoryAdapter();

    // for each row, add it to our recycler view
    while (cursor.moveToNext()) {
        adapter.add(new MatchHistory(cursor.getInt(0), cursor.getInt(1), cursor.getLong(2)));
    }
    cursor.close();

    // display it one by one
    LinearLayoutManager layoutManager = new LinearLayoutManager(this);
    recyclerView.setLayoutManager(layoutManager);
    recyclerView.setAdapter(adapter);

    // start the sensor thread
    sensorThread = new SensorThread(this);
    new Thread(sensorThread).start();

    // create the bluetooth transmission thread
    bluetoothTransmission = new BluetoothTransmission(this);
}

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (bluetoothTransmission == null){
        return true;
    }
    if (keyCode == KeyEvent.KEYCODE_VOLUME_UP){
        // volume-up button is pressed
        bluetoothTransmission.incrementDelay();
    }
    else if ((keyCode == KeyEvent.KEYCODE_VOLUME_DOWN)){
        // volume-down button is pressed
        bluetoothTransmission.decrementDelay();
    }
    return true;
}
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.bluetooth_connect_button: {
            // when the user clicks the connect button, start this thread
            Thread thread = new Thread(new BluetoothHandler(this, adapter, bluetoothTransmission));
            thread.start();
            return true;
        }
        case R.id.clear_match_history_button: {
            // when the delete all button has been cleared, start this async task
            new BluetoothAsyncDeleteTask().execute();
            return true;
        }
        default: {
            return super.onOptionsItemSelected(item);
        }
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // create the actionbar menu
    getMenuInflater().inflate(R.menu.menu_main, menu);
    this.menu = menu;
    return super.onCreateOptionsMenu(menu);
}

private class BluetoothAsyncDeleteTask extends AsyncTask<Void, Void, String> {
    // this gets executed when the user tries to delete all match histories from the device
    private ProgressDialog dialog;

    protected void onPreExecute() {
        MainActivity.this.runOnUiThread(new Runnable() {
            public void run() {
                // show the loading dialog
                dialog = ProgressDialog.show(MainActivity.this, "",
                        "Deleting all match histories", true);
            }
        });
    }

    protected String doInBackground(Void... params) {
        try {
            // delete all rows from the database
            db.delete("matches", null, null);
            MainActivity.this.runOnUiThread(new Runnable() {
                public void run() {
                    // clear all entries in the recycler view
                    adapter.clear();
                    adapter.notifyDataSetChanged();
                }
```

```
            });
            return null;
        } catch (Exception e) {
            // something went wrong, return the message
            return e.getMessage();
        }
    }

    protected void onPostExecute(final String str) {
        MainActivity.this.runOnUiThread(new Runnable() {
            public void run() {
                TextView textView = (TextView) findViewById(R.id.status_text);
                if (str == null) {
                    // successfully deleted all messages
                    textView.setText("Successfully deleted all match histories!");
                } else {
                    // if something went wrong, display the error mesasge in the status view text
                    textView.setText(str);
                }
                // hide the loading dialog
                dialog.dismiss();
            }
        });
    }
  }
}
```

3. DatabaseHelper.java

```
package saleem.elec3607_pong;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

/**
 * Created by saleem on 20/05/17.
 */

public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "matchHistoryDatabase"; // the name of our database
    private static final int DB_VERSION = 1; // the version of the database

    private String databaseCreateSql;

    public DatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
        /*
        * create table matches (
            id integer primary key autoincrement,
            p1_score integer,
            p2_score integer,
            date text
        );
        * */
        databaseCreateSql = context.getResources().getString(R.string.database_create);
    }
```

```
    @Override
    public void onCreate(SQLiteDatabase db) {
        // first time the app has run, so create the database
        db.execSQL(databaseCreateSql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
}
```

4. **BluetoothTransmission.java**

```
package saleem.elec3607_pong;

import android.util.Log;
import android.widget.TextView;
import java.io.OutputStream;
import java.io.PrintStream;

/**
 * Created by saleem on 24/05/17.
 */

public class BluetoothTransmission implements Runnable {
    private PrintStream bluetoothWriter;
    private long milliseconds = 500;
    private boolean isRunning;
    private MainActivity activity;
    private TextView statusText;

    public BluetoothTransmission(MainActivity activity) {
        this.activity = activity;
        statusText = (TextView) activity.findViewById(R.id.status_text);
        isRunning = true;
    }

    public void run() {
        while (isRunning) {
            // send to bluetooth code here
            sendData();
            sleep();
        }
        try {
            bluetoothWriter.close();
        }
        catch (Exception e){
            displayText(e.getMessage());
        }
    }
    private void sendData(){
        if (bluetoothWriter == null){
            return;
        }
        int data = -1 * Math.round(activity.sensorThread.readSensorValue());
        String s = "{" + data  + "}";
        try {
            bluetoothWriter.println(s);
        }
        catch (Exception e){
            displayText(e.getMessage());
```

```
            isRunning = false;
        }
    }
    public void attach(OutputStream outputStream){
        try {
            bluetoothWriter = new PrintStream(outputStream);
        }
        catch (Exception e){
            bluetoothWriter = null;
            displayText(e.getMessage());
        }
    }
    public void stop() {
        isRunning = false;
    }

    public void incrementDelay() {
        // increase the delay
        milliseconds += 10;
        displayNewDelay();
    }

    public void decrementDelay() {
        // decrease the delay
        if (milliseconds > 10) {
            milliseconds -= 10;
        }
        displayNewDelay();
    }

    public void displayText(final String string){
        activity.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                statusText.setText(string);
            }
        });
    }
    private void displayNewDelay() {
        displayText("Bluetooth transmission rate changed to " + milliseconds + "ms");
    }

    public void sleep(long ms) {
        // sleeps this thread for "ms" milliseconds
        try {
            Thread.sleep(milliseconds);
        } catch (Exception e) {
            return;
        }
    }

    public void sleep(){
        sleep(this.milliseconds);
    }
}
```

5. **BluetoothHandler.java**

```
package saleem.elec3607_pong;

import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
```

```
import android.bluetooth.BluetoothSocket;
import android.content.ContentValues;
import android.content.Intent;
import android.database.sqlite.SQLiteDatabase;
import android.os.AsyncTask;
import android.view.MenuItem;
import android.widget.TextView;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.util.Date;
import java.util.Set;
import java.util.UUID;

import static saleem.elec3607_pong.MainActivity.REQUEST_ENABLE_BT;

/**
 * Created by saleem on 18/05/17.
 */

public class BluetoothHandler implements Runnable {
    boolean exitCondition = false; // set this to true if you want to stop reading the data
from the bluetooth device
    private TextChanger textChanger; // used to display messages on the status text
    private MainActivity activity; // reference to the main activity class; this is used to
update the UI
    private BluetoothAdapter bluetoothAdapter;  // the built in adapter to communicate with
bluetooth devices
    private SQLiteDatabase db; // database that stores our match histories
    private MatchHistoryAdapter adapter; // recycler view to display a list of our match
histories
    private BluetoothTransmission bluetoothTransmission;
    public BluetoothHandler(MainActivity activity, MatchHistoryAdapter adapter,
BluetoothTransmission bluetoothTransmission) {
        this.adapter = adapter;
        this.activity = activity;
        this.bluetoothTransmission = bluetoothTransmission;

        // the "connect bluetooth" button (used for enabling and disabling the button)
        MenuItem bluetoothItem = activity.menu.findItem(R.id.bluetooth_connect_button);

        // the status text
        TextView textView = (TextView) activity.findViewById(R.id.status_text);

        // create the new textChanger object
        textChanger = new TextChanger(textView, bluetoothItem);
        // set the status text to this, and set isConnected to false
        setText("Started Bluetooth Handler Thread!", false);

        // create the bluetooth adapter
        bluetoothAdapter = bluetoothAdapter.getDefaultAdapter();

        if (bluetoothAdapter == null) {
            // if  the device doesn't support bluetooth
            setText("This device doesn't support Bluetooth!", false);
            return;
        }

        // if bluetooth is not enabled, ask to enable it
```

```
        if (!bluetoothAdapter.isEnabled()) {
            setText("Bluetooth is not enabled!", false);
            Intent enableBtIntent = new Intent(bluetoothAdapter.ACTION_REQUEST_ENABLE);
            activity.startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
        }

        // create the database reference object
        DatabaseHelper databaseHelper = new DatabaseHelper(activity);
        db = databaseHelper.getWritableDatabase();
    }

    @Override
    public void run() {
        if (bluetoothAdapter == null) {
            return;
        }

        // used to check whether we found HC-06 device or not
        String deviceName, deviceHardwareAddress;

        // check the paired devices
        setText("Checking paired devices", false);

        // get a list of all paired devices for this device
        Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
        setText("Found " + pairedDevices.size() + " device(s).", false);

        // stop trying to find devices (make it go faster)
        bluetoothAdapter.cancelDiscovery();

        // loop through paired devices until we find the arduino devices
        BluetoothDevice arduinoShield = null;
        for (BluetoothDevice device : pairedDevices) {
            deviceName = device.getName(); // name of the device
            deviceHardwareAddress = device.getAddress(); // MAC address
            if (deviceName.equals(MainActivity.DEVICE_NAME)) {
                setText("Found device " + deviceName + " (" + deviceHardwareAddress + ")",
false);
                arduinoShield = device;
                break;
            }
        }

        // not found arduino shield
        if (arduinoShield == null) {
            setText("Cannot find device " + MainActivity.DEVICE_NAME, false);
            return;
        }

        // create the socket to send and receive data
        BluetoothSocket bluetoothSocket = createSocket(arduinoShield);
        if (bluetoothSocket == null) {
            setText("Cannot create socket", false);
            return;
        }

        // handle the incoming and outgoing messages
        manageMyConnectedSocket(bluetoothSocket);
    }

    private void manageMyConnectedSocket(BluetoothSocket socket) {
        // handle the incoming and outgoing messages
```

```
        String message;
        try {
            // input reader
            InputStream inputStream = socket.getInputStream();

            // attach the output stream to the BluetoothTransmission thread and start it
            bluetoothTransmission.attach(socket.getOutputStream());
            new Thread(bluetoothTransmission).start();

            // an object to make reading the stream easier
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));

            // keep trying to look for messages unless !exitCondition
            while (!exitCondition) {
                if (bufferedReader.ready()) {
                    // received a new message. Run the BluetoothAsyncTask Thread.
                    if ((message = bufferedReader.readLine()) != null) {
                        handleMessage(message);
                        setText("New message \"" + message + "\"", true);
                    }
                }
            }
            // once while loop exits, it means we've disconnected the device
            setText("Disconnected", false);
        } catch (Exception e) {
            // something went wrong
            exitCondition = true;
            setText(e.getMessage(), false);
            try {
                // close the socket
                socket.close();
            } catch (Exception e2) {
                setText(e2.getMessage(), false);
            }
        }
    }

    private void handleMessage(String message) {
        // basic error checking
        if (message == null || message.length() != 2) {
            return;
        }
        final int allyScore, aiScore;
        final String milliseconds; // the current date
        // convert it to integers
        allyScore = message.charAt(0) - '0';
        aiScore = message.charAt(1) - '0';
        milliseconds = String.valueOf(new Date().getTime());
        // basic error checking
        if (allyScore < 0 || allyScore > 9 || aiScore < 0 || aiScore > 9 || aiScore ==
allyScore) {
            return;
        }
        try {
            // write to the database
            ContentValues contentValues = new ContentValues();
            contentValues.put("p1_score", allyScore);
            contentValues.put("p2_score", aiScore);
            contentValues.put("date", milliseconds);

            SQLiteDatabase db = new DatabaseHelper(activity)
```

```
                    .getWritableDatabase();
            db.insert("matches", null, contentValues);
            activity.runOnUiThread(new Runnable() {
                public void run() {
                    // tell our recycler view that we have new data
                    long s = Long.parseLong(milliseconds);
                    adapter.addFirst(new MatchHistory(allyScore, aiScore, s));
                    adapter.notifyItemInserted(0);
                    // scroll up when new message is added
                    activity.recyclerView.smoothScrollToPosition(0);
                }
            });
        } catch (Exception e) {
            // something went wrong while trying to write to the database, send error message
back
            return;
        }
    }

    private BluetoothSocket createSocket(BluetoothDevice device) {
        setText("Initialising socket", false);
        // create the socket
        BluetoothSocket socket;
        try {
            // Get a BluetoothSocket to connect with the given BluetoothDevice.
            // MY_UUID is the app's UUID string, also used in the server code.
            // bluetooth serial port service
            UUID SERIAL_UUID = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");
            socket = device.createRfcommSocketToServiceRecord(SERIAL_UUID);

        } catch (IOException e) {
            // something went wrong with  creating the socket
            setText("Failed to create a socket " + e.getMessage(), false);
            return null;
        }
        setText("Successfully created socket!", false);
        try {
            // try connect to the bluetooth device
            setText("Trying to connect to bluetooth device.", false);
            socket.connect();
            setText("Connected!", true);
            return socket;

        } catch (IOException f) {
            // something went wrong whle trying to connect to the bluetooth device
            setText("Failed to connect! " + f.getMessage(), false);
            try {
                // try connecting in fallback mode
                setText("Trying fallback mode", false);
                socket = (BluetoothSocket) device.getClass().getMethod("createRfcommSocket",
new Class[]{int.class}).invoke(device, 1);
                socket.connect();
                setText("Connected!", true);
                return socket;
            } catch (Exception e2) {
                // something went wrong while trying to connect to the device
                setText(e2.getMessage(), false);
                try {
                    // close the socket
                    setText("Closing socket.", false);
                    socket.close();
                    return null;
```

```
                } catch (Exception e3) {
                    setText("Failed to close socket! " + e3.getMessage(), false);
                    return null;
                }
            }
        }
    }

    void setText(String s, boolean isConnected) {
        // change the text of the statusTextView
        textChanger.setText(s);
        textChanger.setConnected(isConnected);
        // update the UI
        this.activity.runOnUiThread(textChanger);
    }
}
```

6. TextChanger.java

```
package saleem.elec3607_pong;

import android.view.MenuItem;
import android.widget.TextView;

/**
 * Created by saleem on 21/05/17.
 * This class is used to modify the status text as well as enabling/disabling the connect
button
 */

public class TextChanger implements Runnable {
    private String text; // the text we're trying to change
    private TextView statusText; // the textView object that we're trying to check
    private MenuItem bluetoothItem; // the "connect" button
    private boolean isConnected; // true when we're connected to the slave device

    public TextChanger(TextView statusText, MenuItem bluetoothItem) {
        this.statusText = statusText;
        this.bluetoothItem = bluetoothItem;
        text = "";
    }

    public void setText(String s) {
        this.text = s;
    }

    public void setConnected(boolean isConnected) {
        this.isConnected = isConnected;
    }

    @Override
    public void run() {
        // set the text
        statusText.setText(text);
        // if we're connected to the slave, disable the "connect" button. otherwise, enable it
        bluetoothItem.setEnabled(!isConnected);
    }
}
```

7. SensorThread.java

```
package saleem.elec3607_pong;
```

```
import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.widget.TextView;

/**
 * Created by saleem on 24/05/17.
 */

public class SensorThread implements Runnable, SensorEventListener {
    private SensorManager sensorManager;
    private Sensor sensor;
    private Activity activity;
    private TextView statusText;
    private float sensorValue;

    public SensorThread(Activity activity){
        this.activity = activity;
        statusText = (TextView) activity.findViewById(R.id.status_text);
        sensorManager = (SensorManager) activity.getSystemService(Context.SENSOR_SERVICE);
        sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        if (sensor == null) {
            // device does not support sensor
            activity.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    statusText.setText("This device does not support accelerometer");
                }
            });
            return;
        }
    }

    @Override
    public void run(){
        registerSensor();
    }

    public void registerSensor(){
        sensorManager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_GAME);
    }


    @Override
    public void onSensorChanged(SensorEvent event) {
        synchronized (this){
            this.sensorValue = event.values[0];
        }
    }

    public float readSensorValue(){
        float t;
        synchronized (this){
            t = sensorValue;
        }
        return t;
    }
```

```
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }
}
```

8. MatchHistoryAdapter.java

```
package saleem.elec3607_pong;

import android.graphics.Color;
import android.support.v7.widget.CardView;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.ViewGroup;
import android.widget.TextView;
import java.util.LinkedList;

/**
 * Created by saleem on 19/05/17.
 */

public class MatchHistoryAdapter extends RecyclerView.Adapter<MatchHistoryAdapter.ViewHolder>{
    // this is the adapter that actually displays all the match histories
    private LinkedList<MatchHistory> data; // store all match histories in a linked list

    public MatchHistoryAdapter(){
        this.data = new LinkedList<>();
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {
        private CardView cardView;
        public ViewHolder(CardView v) {
            super(v);
            // the recycler view holds a card view
            cardView = v;
        }
    }

    @Override
    public MatchHistoryAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int
                                                             viewType){
        CardView cv = (CardView) LayoutInflater.from(parent.getContext())
                .inflate(R.layout.card_match_history, parent, false);
        // create a new cardview
        return new ViewHolder(cv);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position){
        // this will get called for every entry in the linked list "data"
        MatchHistory matchHistory = data.get(position); // get the match history class

        // extract the information
        int p1Score = matchHistory.getAllyScore();
        int p2Score = matchHistory.getEnemyScore();
        boolean win = p1Score > p2Score;
        CardView cardView = holder.cardView;

        // display the win or loss text
        TextView outcomeText = (TextView)cardView.findViewById(R.id.outcome_text);
        outcomeText.setText(win ? "WIN" : "LOSS");
        // append the scores
```

```
        outcomeText.setText(outcomeText.getText() + " " + p1Score + "-" + p2Score);
        // display the date
        TextView dateText = (TextView)cardView.findViewById(R.id.date_text);
        dateText.setText(matchHistory.getDate());

        // set the background colour to green or red depending on the outcome of the game
(green for win)
        cardView.setBackgroundColor(win ? Color.GREEN: Color.RED);

    }
    // basic methods to delete all items in the linked list, or add an item
    public void clear(){
        this.data.clear();
    }
    // add it to the beginning of the linked list
    public void addFirst(MatchHistory m){
        this.data.addFirst(m);
    }
    // add it to the end of the linked list
    public void add(MatchHistory m){
        this.data.add(m);
    }

    @Override
    public int getItemCount(){
        return this.data.size();
    }
}
```

9. MatchHistory.java

```
package saleem.elec3607_pong;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Created by saleem on 19/05/17.
 * Simple class to store the match histories
 */

public class MatchHistory {
    // ally (human) score
    private int allyScore;
    // computer score
    private int enemyScore;
    // date of the game
    private String date;


    public MatchHistory(int allyScore, int enemyScore, long milliseconds){
        this.allyScore = allyScore;
        this.enemyScore = enemyScore;
        // create and format the date
        Date date = new Date(milliseconds);
        SimpleDateFormat format = new SimpleDateFormat("d/M/y h:ma");
        this.date = format.format(date);
    }

    // setters and getters for our private methods

    public int getAllyScore() {
        return allyScore;
    }
```

```
    public void setAllyScore(int allyScore) {
        this.allyScore = allyScore;
    }

    public int getEnemyScore() {
        return enemyScore;
    }

    public void setEnemyScore(int enemyScore) {
        this.enemyScore = enemyScore;
    }

    public String getDate() {
        return date;
    }

    public void setDate(String date) {
        this.date = date;
    }
}
```

10. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="saleem.elec3607_pong">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/pong"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```