TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

186.A25 Building Interaction Interfaces

Project 2A - “Arduino meets Processing”

Banafsheh Alinezhad - 1229630

Miguel Brito - 1428574

Pedro Santos - 1428569

Introduction

This report focuses on presenting and documenting the project developed on the scope of the Building Interaction Interfaces course. This project had as main goals to get students familiarized with Arduino, Processing and how they can interact with each other. The following chapters describe in a detailed way the main aspects of this project.

Project description

This project has two main components that interact with each other:

- Arduino - On this component, a circuit with some components was assembled and some event handlers were added to send messages to the Serial port.
- Processing - This component is responsible for the graphical part of the application. There is a listener for the Serial port that reads the incoming messages, parses them and triggers the actions corresponding to those messages.

They interact with each other via the serial port, to where the arduino sends some messages when some type of events happen, and then on processing they are parsed and the corresponding actions are started.

Now, regarding the description of the project itself, it is a multiplayer game, where one player plays the game against another player. The game is like a sports track where some hurdles are appearing randomly separated and the players have to jump over these hurdles so they don't start hitting them and start losing their position. The objective of the game is to surpass all obstacles (hurdles) and when the game time comes to an end, be in front of the other player.

When the game starts, both players are stopped and they only start moving when they're "activated" by the RFID tag. The hurdles start appearing when both players are activated. Then, when they're both running, if the front one hits a hurdle the other approaches. If the second one hits a hurdle, he'll increase the distance to the front player.

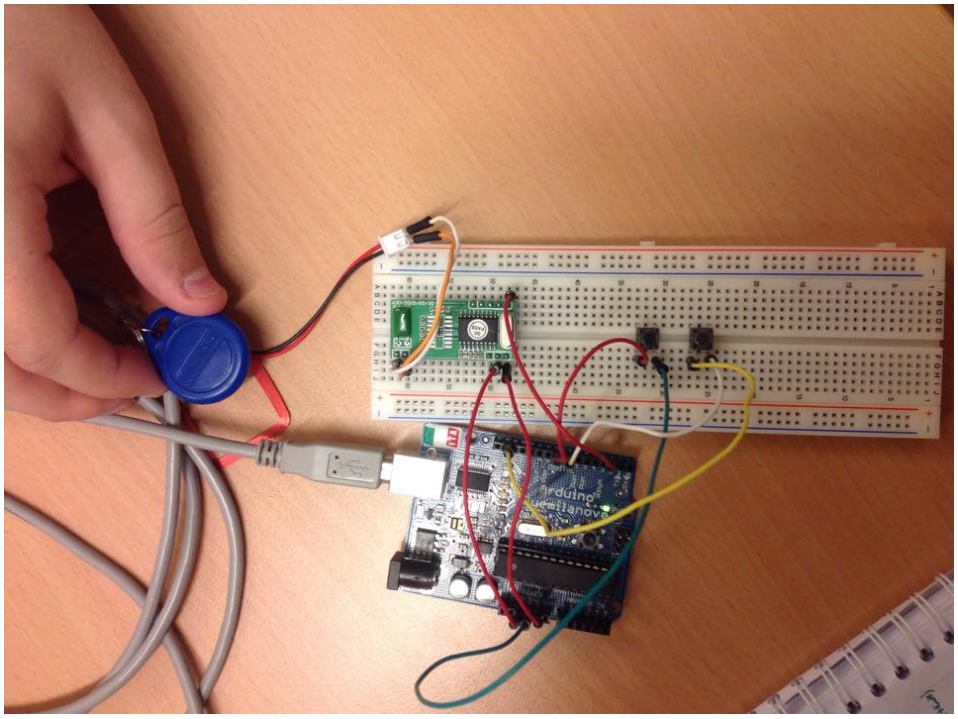
Arduino components

The components (and their use) were used to create the Arduino "scenario" are the following:

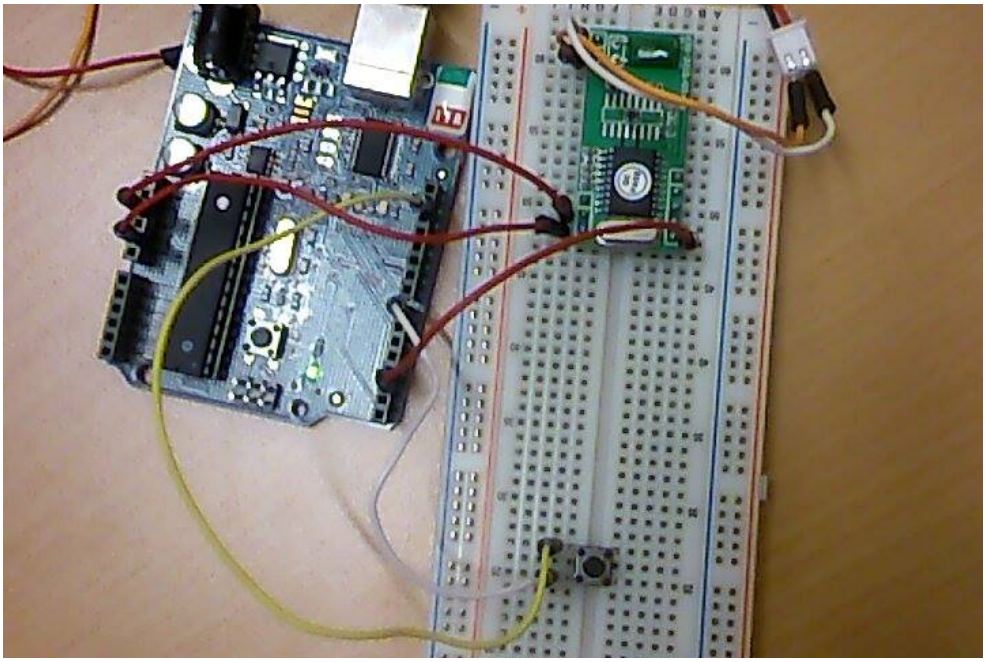
- 1 Arduino Board - This Arduino board is responsible to connect all the components and sending the signals to the serial port;
- 1 IR Sensor - The IR sensor is the responsible to read the sign of the RFID tags;
- 2 RFID tags - Each of the RFID tags (red and blue) are assigned to each player so they can "login" into the game;

- 1 RFID Reader - The RFID decodes the value read from the RFID tags through the sensor;
- 2 Buttons - Each button is responsible to control one player on the game. Pressing a button means “telling” the player to jump;

With this component, we’ve assembled the circuit as the next pictures illustrate:



Img. 1 - The circuit assembled



Img. 2 - A closer view of the circuit

Tutorials used

To start this project we checked a few tutorials to understand better how to work with some components of the system.

- How to work with RFID - http://www.instructables.com/id/Arduino-Tutorials-RFID/?ALLSTEPS;
- A little introduction to Processing - http://arduino.cc/en/Tutorial/Graph;
- Also the tutor showed us an example of how to work the serial port in Processing which unfortunately we don't have the reference, but it's worth mentioning;

Arduino Code

The arduino code basically reads the values of the RFID sensor and the buttons and sends data to the Processing component. When an interruption occurs, the Arduino interprets what button is clicked or what is the value of the RFID tag that triggered the sensor. Based on this, Arduino sends a message to Processing.

```
#include <SoftwareSerial.h>


//Initialization of global variables and constants
SoftwareSerial RFID(2, 3); // RX and TX

int data1 = 0;
int readTag = -1;
int BLUE = 10;
int RED = 11;

int buttonPressed = 0;
int buttonUnpressed = 1;
int redButtonState = buttonUnpressed;
int blueButtonState = buttonUnpressed;

int bluePlayerButton = 7;
int redPlayerButton = 8;

int redTag[14] = {2,51,68,48,48,65,57,49,56,52,69,67,50,3}; //Identifier of the red RFID tag
int blueTag[14] = {2,48,51,48,48,65,53,50,49,52,52,67,51,3}; // Identifier of the blue RFID tag
int newtag[14] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // used for read comparisons


void setup()
{
  RFID.begin(9600); // start serial to RFID reader
  Serial.begin(9600); // start serial to PC

  //Button setup
  pinMode(bluePlayerButton,INPUT);
  digitalWrite(bluePlayerButton,HIGH);
```

```
  pinMode(redPlayerButton,INPUT);
  digitalWrite(redPlayerButton,HIGH);
}

//Comparing two tags (arrays of values)
boolean comparetag(int aa[14], int bb[14])
{
  boolean ff = false;
  int fg = 0;
  for (int cc = 0 ; cc < 14 ; cc++){
    if (aa[cc] == bb[cc]){
      fg++;
    }
  }
  if (fg == 14){
    ff = true;
  }
  return ff;
}

//Check if a read tag is one of ours and return its value.
void checkmytags(){

  if (comparetag(newtag, redTag) == true){
    readTag = RED;
  }
  if (comparetag(newtag, blueTag) == true){
    readTag = BLUE;
  }
}

//Function to read RFID tags from the sensor
void readTags()
```

```
{
  readTag = -1;
  if (RFID.available() > 0) {//checks if there is any value to read on the buffer.
    // read tag numbers
    delay(100); // needed to allow time for the data to come in from the serial buffer.
    for (int z = 0 ; z < 14 ; z++){ // read the rest of the tag
      data1 = RFID.read();
      newtag[z] = data1;
    }
    RFID.flush(); // stops multiple reads
    // Compare the read value to check if it is a valid one
    checkmytags();
  }
  // Based on the read value, write to the buffer the correspondent command
  if (readTag == BLUE){ // if we had a match
    Serial.println("RFID_BLUE");
  }else if (readTag == RED){ // if we didn't have a match
    Serial.println("RFID_RED");
  }
  readTag = -1;
}

//Read the value of the buttons - if it is the red or the blue one or even if it is a press or a release from that button.
void readButtons(){

  if(digitalRead(bluePlayerButton) == buttonPressed){
    if(blueButtonState == buttonUnpressed){
      blueButtonState = buttonPressed;
      Serial.println("BLUE_PRESS");
    }
  }else{
    if(blueButtonState == buttonPressed){
```

```
      blueButtonState = buttonUnpressed;
      Serial.println("BLUE_RELEASE");
    }
  }

  if(digitalRead(redPlayerButton) == buttonPressed){
    if(redButtonState == buttonUnpressed){
      redButtonState = buttonPressed;
      Serial.println("RED_PRESS");
    }
  }else if (digitalRead(redPlayerButton) == buttonUnpressed){
    if(redButtonState == buttonPressed){
      redButtonState = buttonUnpressed;
      Serial.println("RED_RELEASE");
    }
  }
}

//In the main loop, we are reading the buttons and RFID tags to send commands to processing to notify it from the user's actions, and then these commands are parsed on processing
void loop()
{
  readTags();
  readButtons();
}
```

Processing Code

The Processing component does all the game logic and the drawing. It controls the game state and the players state based on the users input in the arduino. Most of the code is responsible for the management of the game state, hurdles, and winning conditions. The last functions represent the drawing of everything, including small animations and also the jumping physics.

```
import processing.serial.*;
import java.util.Random;

Serial myPort;
String readValue;
GameState state = GameState.START;
PlayerState blueState = PlayerState.SET;
PlayerState redState = PlayerState.SET;

//constant y coordinate of players
final int blueScreenY = 200;
final int redScreenY = 325;
//variable x coordinate of players
int blueX = 25;
int redX = 25;
//auxiliar y coordinate used to simulate jump
int blueY = 0;
int redY = 0;
int animation = 0;

//Constants used
int MAX_RAND = 225;
int MIN_RAND = 125;
int MAX_HEIGHT = -50;
int MIN_HEIGHT = 0;
int MAX_GAME_OVER_TIMER = 500;
int MAX_HURT_TIMER = 50;
int MAX_GAME_TIME = 5000;

//Timers to control several aspects of the game.
int gameTimer = 0;
```

```
int gameOverTimer = 0;
int mutexTimer = 0;
int hurdlesTimer = 0;
int blueTimer = 0;
int redTimer = 0;

PFont f;

int randomNum = 0; //used to create hurdles at a random timestamp

ArrayList<Integer> hurdles = new ArrayList<Integer>();
void setup(){

  f = createFont("Arial",28,true); // Arial, 16 point, anti-aliasing on

  //Initializing Serial Port for comunication and screen.
  String portName = Serial.list()[0];
  myPort = new Serial (this, portName, 9600);
  size(400, 400);
  background(0, 0, 0);

  GameState state = GameState.START;
  blueState = PlayerState.SET;
  redState = PlayerState.SET;

  //Initializing timers to 0 at start
  mutexTimer = 0;
  blueTimer = 0;
  redTimer = 0;
  gameOverTimer = 0;
  gameTimer = 0;
  hurdlesTimer = 0;
```

```
  //Initializing positions
  blueX = 25;
  redX = 25;
  blueY = 0;
  redY = 0;
  hurdles = new ArrayList<Integer>();

}
void draw(){

  //Check if it Game is over
  //Three conditions - Either the time has passed, or one of the players got out of the screen or
one player has reached 200 on X coordinate without the other starting
  //The player who is in front wins.
  if(gameTimer >= MAX_GAME_TIME || blueX < 0 || redX < 0
          || (state == GameState.REDONLY && redX >= 200)
          || (state == GameState.BLUEONLY && blueX >= 200)){
    textFont(f,38);
    fill(255);
    if(blueX > redX){
      text("Blue player Wins!",50,200);
    }else{
      text("Red player Wins!",50,200);
    }

   gameOverTimer++;
   //Check if the Game Over message can be discarted and the game can be restarted
   if(gameOverTimer >= MAX_GAME_OVER_TIMER){
      myPort.stop();
      setup();
      state = GameState.START;
   }else
     return;
```

```
  }

  //cleans the last frame
  background(0, 0, 0);
  //variable to check if the "camera" moves
  boolean bothMoving = true;

  //makes the red lights blink
  if(state == GameState.START){
    if(mutexTimer < 50 || (mutexTimer < 100 && mutexTimer > 60)){
      stroke(#FF0000);
      fill(#FF0000);
      ellipse(200, 75, 75, 75);
    }

    mutexTimer++;

    if(mutexTimer > 110)
      state = GameState.READY;
  }

  //makes the green light appear
  if(state == GameState.READY || state == GameState.REDONLY || state ==
GameState.BLUEONLY){
    stroke(#00FF00);
    fill(#00FF00);
    ellipse(200, 75, 75, 75);
  }

  //IF the blue player is the only one that started running
  //OR the red player hits a hurdle and is ahead of the player
  //the camera "stops"
```

```
if(state == GameState.BLUEONLY || (redState == PlayerState.HURTING && redX > blueX)){
  blueX++;
  bothMoving = false;
}

//Same as the last if, but for the red player
if(state == GameState.REDONLY || (blueState== PlayerState.HURTING && blueX > redX)){
  redX++;
  bothMoving = false;
}

//If the red player hits a hurdle and is behind
//the camera does not stop and he stays behind
if(redState == PlayerState.HURTING && redX <= blueX){
  redX--;
}

//Same as last if, but for the blue player
if(blueState == PlayerState.HURTING && blueX <= redX){
  blueX--;
}

//Player moves in the beggining until they reach the middle of the screen
if(state == GameState.NOHURDLES){
  blueX++;
  redX++;

  if(blueX >= 200 || redX >= 200){
    state = GameState.BOTH;
    randomNum = 0;
  }
}
```

```
  //Camera "moves" and makes the hurdle closer
  if(bothMoving){
    for(Integer i = 0; i < hurdles.size(); i++){
      hurdles.set(i, hurdles.get(i) -1);
      gameTimer++;
    }
  }

  //Hurdles creation
  if(state == GameState.BOTH && bothMoving){

    if(hurdlesTimer >= randomNum){
      hurdles.add(400);
      hurdlesTimer = 0;
      Random rand = new Random();
      randomNum = rand.nextInt((MAX_RAND - MIN_RAND) + 1) + MIN_RAND; //Generate a random number between MIN_RAND and MAX_RAND so that the next hurdle is created at a random distance from the previous

    }
    hurdlesTimer++;
  }

  // Snippet for the communication.
  if(myPort.available() > 0){

    readValue = myPort.readStringUntil('\n'); // Read line by line what's in the buffer.

    //Parsing the received messages
    if(readValue != null){
      if(state == GameState.READY && readValue.contains("RFID_RED")){
        state = GameState.REDONLY;
```

```
        redState = PlayerState.RUNNING;
      }
      if(state == GameState.BLUEONLY && readValue.contains("RFID_RED")){
        state = GameState.NOHURDLES;
        redState = PlayerState.RUNNING;
      }
      if(state == GameState.READY && readValue.contains("RFID_BLUE")){
        state = GameState.BLUEONLY;
        blueState = PlayerState.RUNNING;
      }
      if(state == GameState.REDONLY && readValue.contains("RFID_BLUE")){
        state = GameState.NOHURDLES;
        blueState = PlayerState.RUNNING;
      }
      if(readValue.contains("BLUE_PRESS") && blueState == PlayerState.RUNNING){
        blueState = PlayerState.JUMPING;
      }
      if(readValue.contains("RED_PRESS")  && redState == PlayerState.RUNNING){
       redState = PlayerState.JUMPING;
      }
    }
  }

  drawPlayers();
  drawLines();
  drawHurdles();
  checkColisions();

}
```

//Function to check collisions between players. This checking is done by comparing if a hurdle's X position is the same as the player's X and if this is not jumping (A bit naive algorithm, does not fully check the collision)

```
void checkColisions(){
 for(int i = 0; i < hurdles.size(); i++){
    if(blueX == hurdles.get(i) && (blueState == PlayerState.RUNNING)){
        blueState = PlayerState.HURTING;
        blueX += 2;
    }

    if(redX == hurdles.get(i) && (redState == PlayerState.RUNNING)){
       redState = PlayerState.HURTING;
       redX += 2;
    }

 }

}

//Players are being drawn here. And the jumping control is also done here. When the player jumps, it decreases its Y position (because (0,0) is on the top left of the screen) until 50 px.
//Then, when the players reaches that altitude, starts falling again until he reaches the original position.
void drawPlayers(){
  //BluePlayer
  if(blueState == PlayerState.JUMPING && blueY >= MAX_HEIGHT){
    if(blueY == MAX_HEIGHT){
      blueState = PlayerState.FALLING;
    }else
      blueY--;
  }else if(blueState == PlayerState.FALLING  && blueY <= MIN_HEIGHT){
    if(blueY == MIN_HEIGHT){
      blueState = PlayerState.RUNNING;
     }else
       blueY++;
```

```
}
if(blueState == PlayerState.HURTING){
  if(blueTimer < MAX_HURT_TIMER){
   blueTimer++;
  }else{
    blueState = PlayerState.RUNNING;
    blueTimer = 0;
  }
}

//RedPlayer
if(redState == PlayerState.JUMPING && redY >= MAX_HEIGHT){
  if(redY == MAX_HEIGHT){
    redState = PlayerState.FALLING;
   }else
    redY--;
}else if(redState == PlayerState.FALLING  && redY <= MIN_HEIGHT){
  if(redY == MIN_HEIGHT){
    redState = PlayerState.RUNNING;
   }else
     redY++;
}

if(redState == PlayerState.HURTING){
  if(redTimer < MAX_HURT_TIMER){
   redTimer++;
  }else{
    redState = PlayerState.RUNNING;
    redTimer = 0;
  }
}

//BluePlayer
```

```
stroke(#0000FF);
fill(#0000FF);
if(blueState == PlayerState.SET){
  ellipse(blueX+20,blueScreenY + blueY - 5, 10, 10);
  line(blueX,blueScreenY + blueY + 10,blueX + 4,blueScreenY + blueY + 2);
  line(blueX + 4,blueScreenY + blueY + 2,blueX+20,blueScreenY + blueY - 5);
  line(blueX+10,blueScreenY + blueY + 1,blueX + 13,blueScreenY + blueY +10);
}
if(blueState == PlayerState.RUNNING){
  if(animation < 10){
    ellipse(blueX+5,blueScreenY + blueY - 10, 10, 10);
    line(blueX+5,blueScreenY + blueY - 10, blueX+5, blueScreenY + blueY+2);
    line(blueX+5,blueScreenY + blueY + 2, blueX, blueScreenY + blueY + 10);
    line(blueX+5,blueScreenY + blueY + 2, blueX+10, blueScreenY + blueY + 10);
    line(blueX+5,blueScreenY + blueY, blueX, blueScreenY + blueY + 3);
    line(blueX+5,blueScreenY + blueY, blueX+10, blueScreenY + blueY - 3);
  } else {
    ellipse(blueX+5,blueScreenY + blueY - 10, 10, 10);
    line(blueX+5,blueScreenY + blueY - 10, blueX+5, blueScreenY + blueY+2);
    line(blueX+5,blueScreenY + blueY + 2, blueX+2, blueScreenY + blueY + 10);
    line(blueX+5,blueScreenY + blueY + 2, blueX+8, blueScreenY + blueY + 10);
    line(blueX+5,blueScreenY + blueY, blueX+2, blueScreenY + blueY + 1);
    line(blueX+5,blueScreenY + blueY, blueX+8, blueScreenY + blueY - 1);
  }
}
if(blueState == PlayerState.JUMPING || blueState == PlayerState.FALLING){
  ellipse(blueX+5,blueScreenY + blueY - 10, 10, 10);
  line(blueX+5,blueScreenY + blueY - 10, blueX+5, blueScreenY + blueY+2);
  line(blueX+5,blueScreenY + blueY + 2, blueX-2, blueScreenY + blueY + 8);
  line(blueX+5,blueScreenY + blueY + 2, blueX+12, blueScreenY + blueY + 8);
  line(blueX+5,blueScreenY + blueY, blueX, blueScreenY + blueY - 3);
  line(blueX+5,blueScreenY + blueY, blueX+10, blueScreenY + blueY - 3);
}
```

```
if(blueState == PlayerState.HURTING){
  if(animation < 10){
    ellipse(blueX+20,blueScreenY + blueY - 5, 10, 10);
    line(blueX,blueScreenY + blueY + 10,blueX + 4,blueScreenY + blueY + 2);
    line(blueX + 4,blueScreenY + blueY + 2,blueX+20,blueScreenY + blueY - 5);
    line(blueX+10,blueScreenY + blueY + 1,blueX + 13,blueScreenY + blueY +10);
  }
}

//RedPlayer
stroke(#FF0000);
fill(#FF0000);
if(redState == PlayerState.SET){
  ellipse(redX+20,redScreenY + redY - 5, 10, 10);
  line(redX,redScreenY + redY + 10,redX + 4,redScreenY + redY + 2);
  line(redX + 4,redScreenY + redY + 2,redX+20,redScreenY + redY - 5);
  line(redX+10,redScreenY + redY + 1,redX + 13,redScreenY + redY +10);
}
if(redState == PlayerState.RUNNING){
  if(animation < 10){
    ellipse(redX+5,redScreenY + redY - 10, 10, 10);
    line(redX+5,redScreenY + redY - 10, redX+5, redScreenY + redY+2);
    line(redX+5,redScreenY + redY + 2, redX, redScreenY + redY + 10);
    line(redX+5,redScreenY + redY + 2, redX+10, redScreenY + redY + 10);
    line(redX+5,redScreenY + redY, redX, redScreenY + redY + 3);
    line(redX+5,redScreenY + redY, redX+10, redScreenY + redY - 3);
  } else {
   ellipse(redX+5,redScreenY + redY - 10, 10, 10);
    line(redX+5,redScreenY + redY - 10, redX+5, redScreenY + redY+2);
    line(redX+5,redScreenY + redY + 2, redX+2, redScreenY + redY + 10);
    line(redX+5,redScreenY + redY + 2, redX+8, redScreenY + redY + 10);
    line(redX+5,redScreenY + redY, redX+2, redScreenY + redY + 1);
    line(redX+5,redScreenY + redY, redX+8, redScreenY + redY - 1);
```

```
    }
  }
  if(redState == PlayerState.JUMPING || redState == PlayerState.FALLING){
    ellipse(redX+5,redScreenY + redY - 10, 10, 10);
    line(redX+5,redScreenY + redY - 10, redX+5, redScreenY + redY+2);
    line(redX+5,redScreenY + redY + 2, redX-2, redScreenY + redY + 8);
    line(redX+5,redScreenY + redY + 2, redX+12, redScreenY + redY + 8);
    line(redX+5,redScreenY + redY, redX, redScreenY + redY - 3);
    line(redX+5,redScreenY + redY, redX+10, redScreenY + redY - 3);
  }
  if(redState == PlayerState.HURTING){
    if(animation < 10){
      ellipse(redX+20,redScreenY + redY - 5, 10, 10);
      line(redX,redScreenY + redY + 10,redX + 4,redScreenY + redY + 2);
      line(redX + 4,redScreenY + redY + 2,redX+20,redScreenY + redY - 5);
      line(redX+10,redScreenY + redY + 1,redX + 13,redScreenY + redY +10);
    }
  }

  animation++;
  if(animation >= 20){
   animation = 0;
  }
}
void drawLines(){
  stroke(#FFFFFF);
  line(0,175,400,175);
  line(0,225,400,225);
  line(0,300,400,300);
  line(0,350,400,350);
}
void drawHurdles(){
  stroke(#FFFFFF);
```

```
  for(Integer i: hurdles){
    line(i, 185, i + 25, 200);
    line(i, 185, i, 200);
    line(i + 25, 200, i + 25, 215);
    line(i, 200, i + 2, 198);
    line(i + 25, 215, i + 27, 213);

    line(i, 310, i + 25, 325);
    line(i, 310, i, 325);
    line(i + 25, 325, i + 25, 340);
    line(i, 325, i + 2, 323);
    line(i + 25, 340, i + 27, 338);
  }
}
```