

American University of Sharjah

COE 312 - Software Design for Engineers

Phase V – Jumping Jackpot (Virtual Jump Rope)

Spring 2019

Submitted to: Imran A. Zualkernan

April 22, 2019

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student Name** | **ID** | **Contribution** | **Overall Contribution** | **Grade** |
| **Amal**  **Suleiman** | 68541 | * Hardware Construction * Developing the Java Code * Developing the Arduino code * Writing and editing the report * Filming and editing the video | 33% |  |
| **Brylle**  **Gomez** | 67871 | * Hardware Construction * Developing the Java Code * Developing the Arduino code * Writing and editing the report * Filming and editing the video | 33% |  |
| **Mueez**  **Khan** | 68255 | * Hardware Construction * Developing the Java Code * Developing the Arduino code * Writing and editing the report * Filming and editing the video | 33% |  |

1. **Hardware design**
2. **ACM Hardware Components**

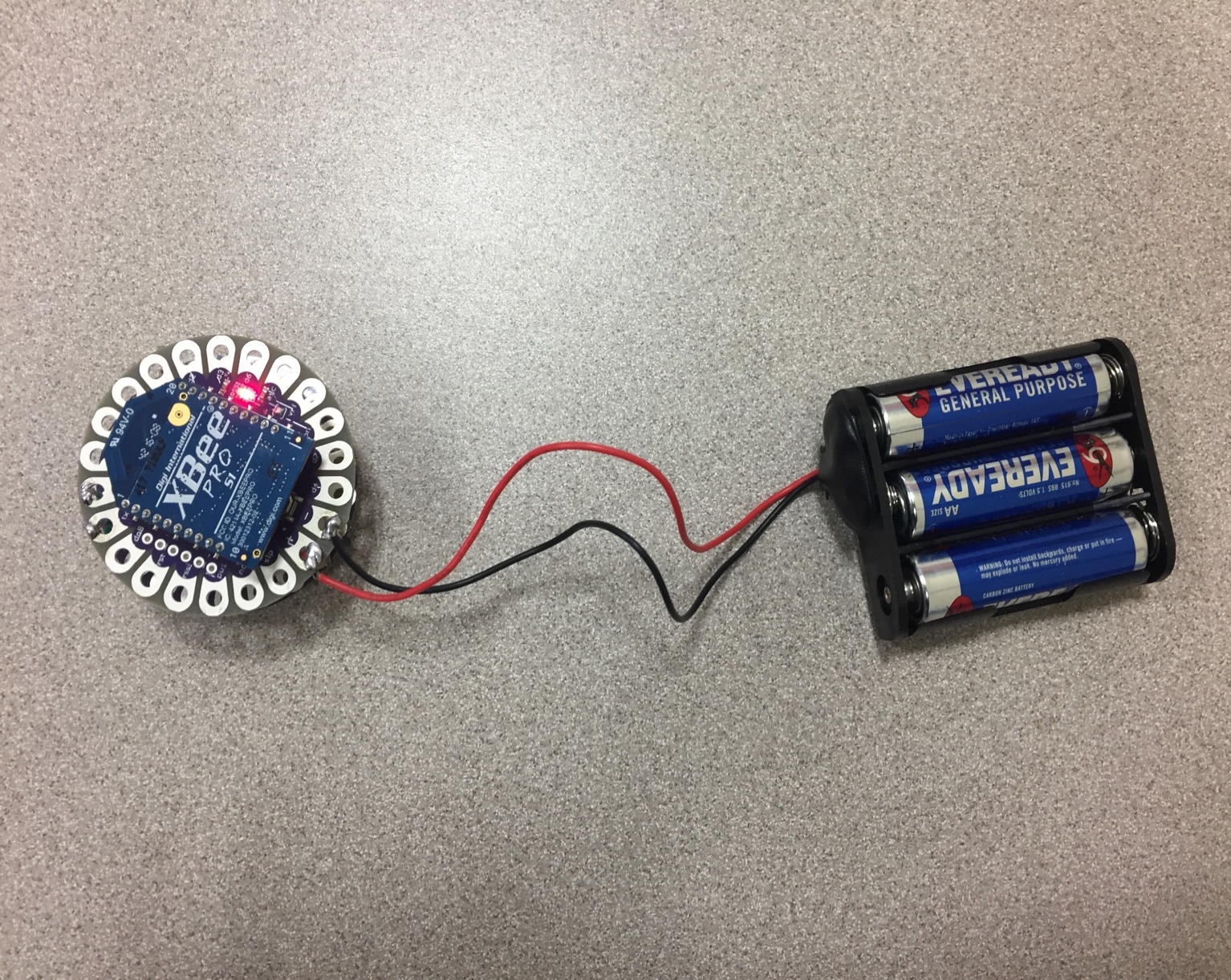
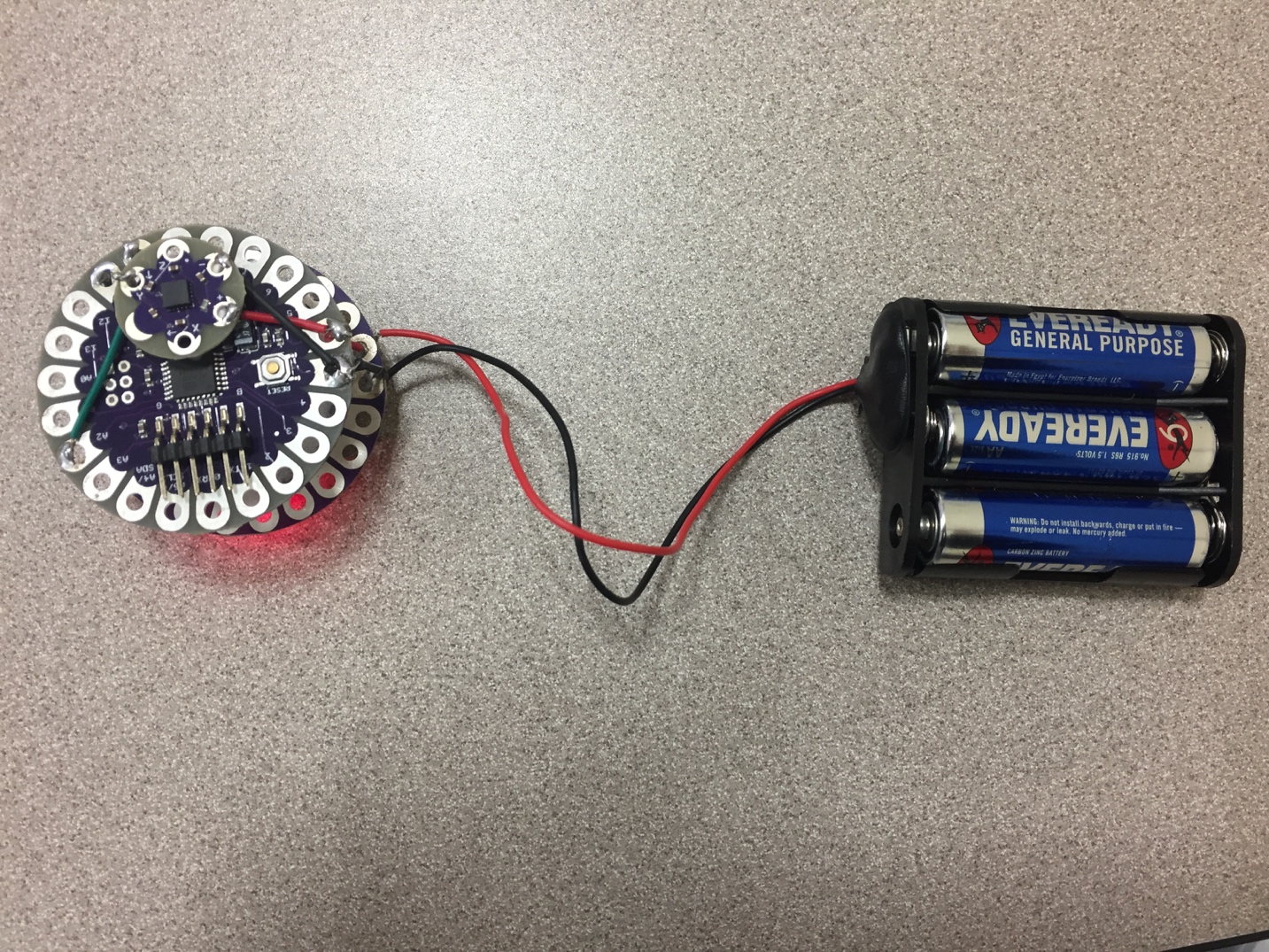


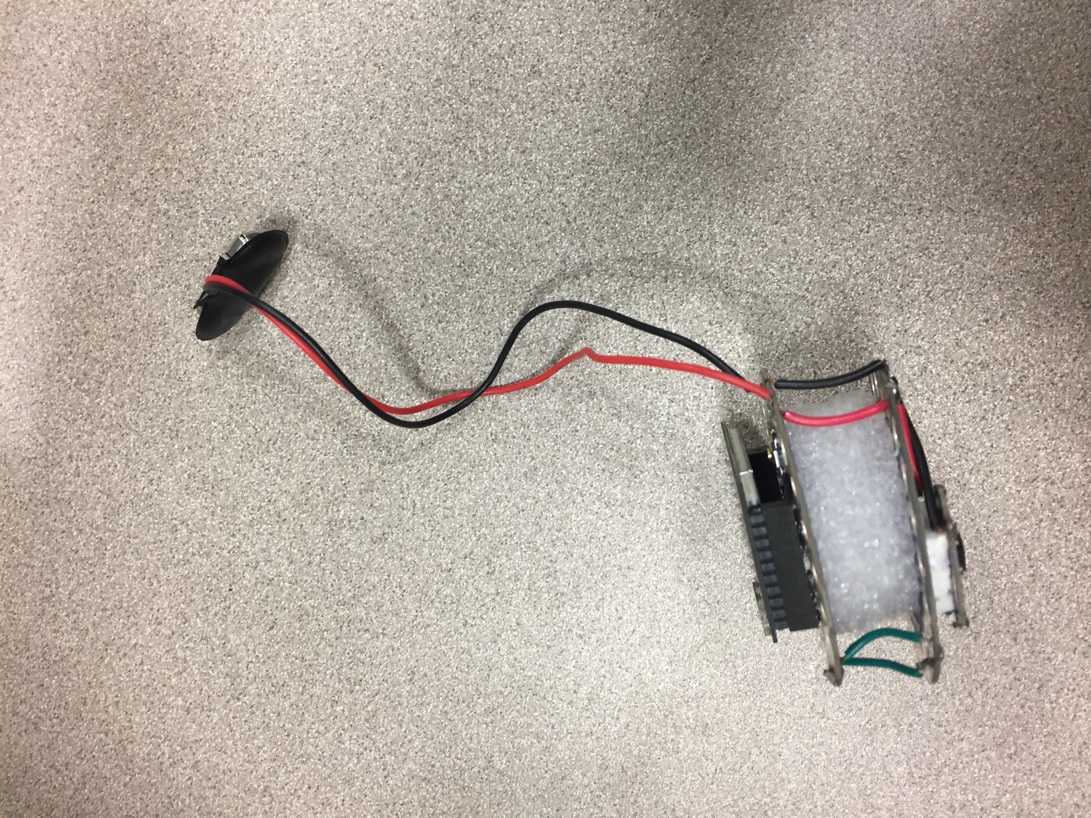
Figure 1 – ACM hardware ****



Figure 2 – Waist belt to hold ACM components

****

Figure 3 – ACM belt which contains the components for ease of jumping

1. **LED Hardware Components**

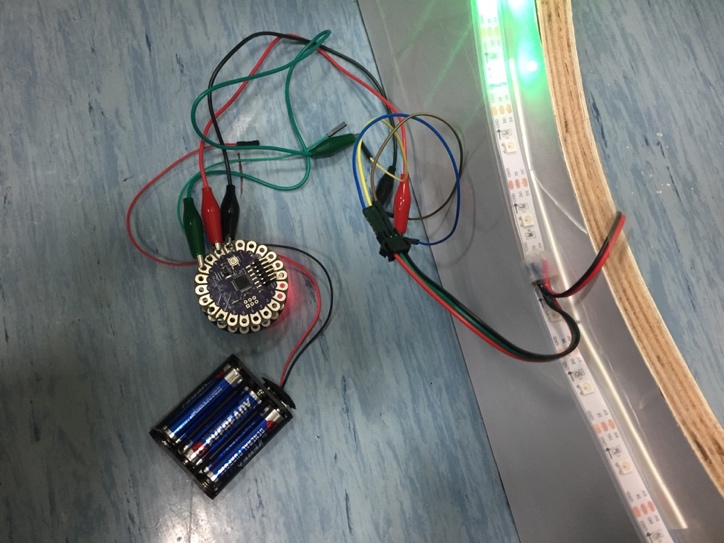
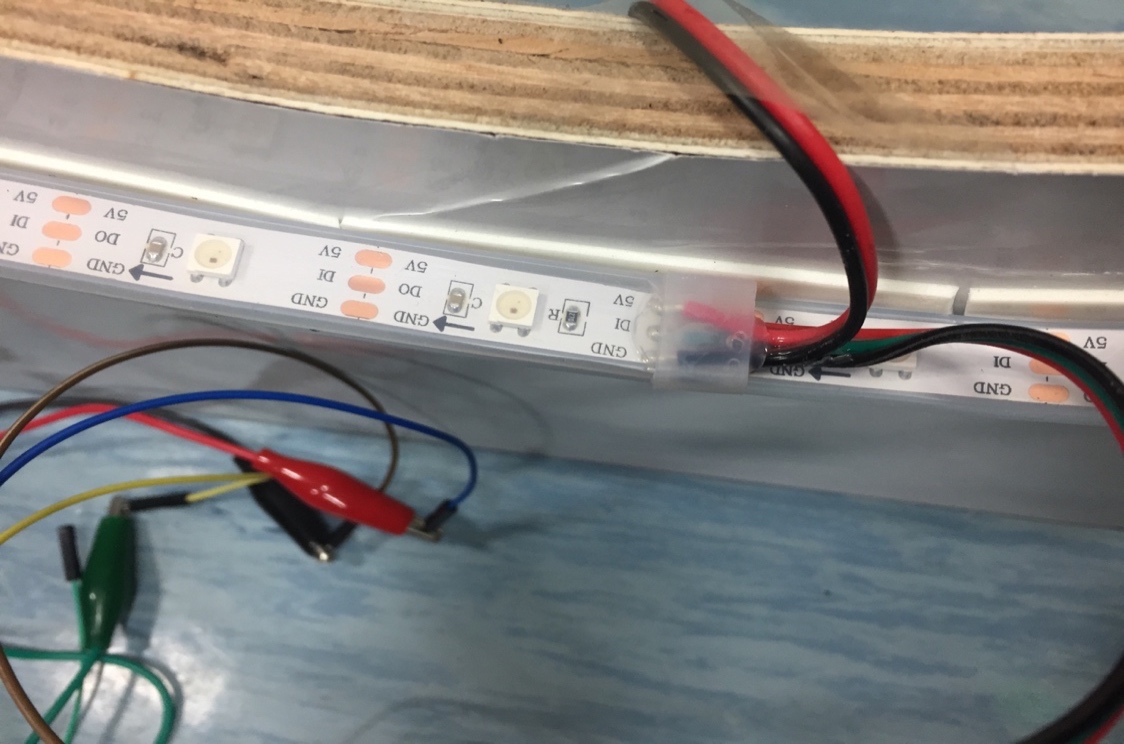


Figure 4 – LED hardware components

1. **Hardware Description**

The game has two end devices, one is connected to the WS2812B Adressable LED Strip ALITOVE WS2812B Addressable RBG 150 LED Strip which is in the form of a ring and the other to the accelerometer which is attached to a belt. Each end device consists of one Lilypad XBee and one Lilypad Arduino Micro. The Lilypad Arduino Micro is connected to a 4.5V Battery Pack which powers the Lilypad Arduino as well as the XBee. The Lilypad Xbee is used to send data wirelessly to the Lilypad XBee Coordinator which is connected to a computer. The LED ring is connected to a pair of Arduino Micro and XBee which sends and receives data from the coordinator. The coordinator is used to start/stop the LED and receive the number of the LED which is turn on.

The Lilypad accelerometer is connected to another pair of Arduino Micro and XBee which sends the accelerometer data to the coordinator. We use the Y-axis of the accelerometer to record data and this data is sent to through the Arduiono+XBee to the coordinator. The data is then used by the Java code, which contains the game logic, to detect a jump correctly and increment the score if a jump is successful.

1. **Commented Arduino Code**
2. **ACM Code**

#include <SoftwareSerial.h>  
#define   ID        1   
#define   JUMPED    1  
#define   LANDED    2  
  
SoftwareSerial mySerial(10, 11);     // RX, TX  
const int ypin = A2;                 // y-axis  
int acmVal;                          // stores ypin sensor values  
  
void setup() {  
  
  mySerial.begin(9600);              // initialize the serial communications:  
  
}  
  
void loop() {  
  
  acmVal = analogRead(ypin);  
  //mySerial.println(acmVal);  
  //Serial.println(acmVal);

if (acmVal > 750) {               //acmVal > 700 indicates a "jump"  
    sendMessage(ID, JUMPED);  
    delay(325);  
    while(acmVal > 610 && acmVal < 604);  
    delay(325);  
    while(acmVal > 610 && acmVal < 604);  
    sendMessage(ID, LANDED);  
  
  }  
  
  delay(25);  
}  
  
void sendMessage(byte id, byte payload) {  
  byte m = (id << 6) | payload;  
  mySerial.println(m);  
}

1. **LED Code**

#ifdef \_\_AVR\_\_  
  #include <avr/power.h>  
#endif  
#include <SoftwareSerial.h>  
#include <Adafruit\_NeoPixel.h>  
#define PIN       5  
#define numPX     150 //number of LEDs in the entire strip  
#define startPX   0   // LED # where you want the LED loop to begin  
#define endPX     127 // LED # where you want the LED loop to end  
#define bandSize  5   // how many LEDs   
#define bottomLED1 121  
#define bottomLED2 127  
  
#define   ID          0  
  
#define   BOTLED1     1  
#define   BOTLED2     2  
#define   GAMEOVER    3  
  
Adafruit\_NeoPixel pixels = Adafruit\_NeoPixel(numPX, PIN, NEO\_GRB + NEO\_KHZ800);   
SoftwareSerial mySerial(10, 11);     // RX, TX  
int speedx = 10;  
void setup() {  
  pixels.begin();                     // This initializes the NeoPixel library.  
  Serial.begin(9600);  
  mySerial.begin(9600);              // initialize the serial communications:  
  // This initializes all LEDs to LOW (off)  
  for (int i = 0; i < numPX; i++) {  
    pixels.setPixelColor(0, pixels.Color(0,0,0));   
    pixels.show();  
  }  
  
}  
void loop() {  
  // pixels.Color takes RGB values, from 0,0,0 up to 255,255,255  
  // turns on LED lights and makes them go around the ring  
  
if (mySerial.available())  
{  
  int n = mySerial.read() - 48;  
  if (n == 4)  
  {  
    for(int j = 0; j<5 ; j++){  
  
      for (int i = startPX; i <= endPX; i++) {  
        if (mySerial.available()) {  
          int x = mySerial.read() - 48;  
          switch(x){  
            case 1:  
              speedx = 10; //slow (for easy mode)  
              break;  
            case 2:  
              speedx = 1;   //fast (for both modes)  
              break;  
            case 4:  
              speedx = 5;   //slow2 (for hard mode)  
              break;  
            default:  
              ;  
          }  
        }  
  
        // set color for LED  
        pixels.setPixelColor((i % (endPX-startPX+1)), pixels.Color(255,50,50));   
        pixels.setPixelColor(((i+1) % (endPX-startPX+1)), pixels.Color(255,50,50));  
        pixels.setPixelColor(((i+2) % (endPX-startPX+1)), pixels.Color(255,50,50));  
        pixels.setPixelColor(((i+3) % (endPX-startPX+1)), pixels.Color(255,50,50));  
        pixels.show(); // Show the changes made to the LED colors (above)  
        delay(speedx); // dictates speed of LED rotation around the ring  
        pixels.setPixelColor((i % (endPX-startPX+1)), pixels.Color(0,0,0));   
        pixels.setPixelColor(((i+1) % (endPX-startPX+1)), pixels.Color(0,0,0));  
        pixels.setPixelColor(((i+2) % (endPX-startPX+1)), pixels.Color(0,0,0));  
        pixels.setPixelColor(((i+3) % (endPX-startPX+1)), pixels.Color(0,0,0));  
        pixels.show(); // Turn the LEDs back off  
        if (i == bottomLED1) {               // bottomLED indicates the LED # that, once lit up, the player has to jump over  
          sendMessage(ID, BOTLED1);              // send 00000001 if bottom LEDs are on (ID 00, instruction 000001)  
        }  
        if (i == bottomLED2) {  
          sendMessage(ID, BOTLED2);             // send 00000010 if bottom LEDs are on (ID 00, instruction 000010)  
        }  
  
      }  
    }  
         sendMessage(ID, GAMEOVER);  
        for (int i = startPX; i <= endPX; i++)  
        {  
        pixels.setPixelColor((i % (endPX-startPX+1)), pixels.Color(255,0,0));   
        pixels.setPixelColor(((i+1) % (endPX-startPX+1)), pixels.Color(255,0,0));  
        pixels.setPixelColor(((i+2) % (endPX-startPX+1)), pixels.Color(255,0,0));  
        pixels.setPixelColor(((i+3) % (endPX-startPX+1)), pixels.Color(255,0,0));  
        pixels.show(); // Show the changes made to the LED colors (above)   
        }  
        for (int i = startPX; i <= endPX; i++)  
        {  
        pixels.setPixelColor((i % (endPX-startPX+1)), pixels.Color(0,0,0));   
        pixels.setPixelColor(((i+1) % (endPX-startPX+1)), pixels.Color(0,0,0));  
        pixels.setPixelColor(((i+2) % (endPX-startPX+1)), pixels.Color(0,0,0));  
        pixels.setPixelColor(((i+3) % (endPX-startPX+1)), pixels.Color(0,0,0));  
        pixels.show(); // Show the changes made to the LED colors (above)   
  }  
  
   }  
}  
}  
  
void sendMessage(byte id, byte payload) {  
  byte m = (id << 6) | payload;  
  mySerial.println(m);  
}

1. **Arduino Code Explanation**

…

if (acmVal > 750) {               //acmVal > 700 indicates a "jump"  
    sendMessage(ID, JUMPED);  
    delay(325);

…

Figure 5 – ACM Arduino

If the acmVal is greater than 750 then the sendMessage function will send the ID (indicating the end device) and the signal Jumped. It will delay for 325ms as that is usually the range of time needed to complete a full jump.

…

while(acmVal > 610 && acmVal < 604);  
    delay(325);

while(acmVal > 610 && acmVal < 604);  
    sendMessage(ID, LANDED);

…

Figure 6 – Configure the LED

If the acmVal is between 610 and 604 then this will detect a Jump landed and will send this signal along with the ID.

…

Adafruit\_NeoPixel pixels = Adafruit\_NeoPixel(numPX, PIN, NEO\_GRB + NEO\_KHZ800);

…

Figure 7 – Configure the LED

We used the Adafruit NeoPixel library to configure the LED Strip and program it. In Figure

4, we configure the LED using the Adafruit\_NeoPixel function and we pass the value of number of LED’s and the pin number to which the LED strip is connected. The NEO\_GRB + NEO\_KHZ800 is a default value.

…

pixels.setPixelColor((i % endPX), pixels.Color(0,255,0));

…

Figure 8 - Setting the pixel color

In the Figure 5, we set the color of the individual LEDs using setPixelColor(). The first parameter sets the location of the LED and the latter selects the color desired by the user.

1. **Commented Java Code**
2. **Proxy.java**

**import** java.io.\*;

**public** **abstract** **class** Proxy **implements** Runnable, Observer {

String name;

*//InputStream in;*

*//OutputStream out;*

**int** id;

Dispatcher d;

Game g;

*// Proxy object constructor*

Proxy(String name, **int** i, Dispatcher d, Game g) **throws** IOException{

*// initialize data members*

**this**.name = name;

*//this.in = new BufferedInputStream(in);*

*//this.out = new BufferedOutputStream(out);*

**this**.id = i;

**this**.d = d;

**this**.g = g;

d.register(**this**); *// register yourself as a proxy with dispatcher d*

**new** Thread(**this**).start(); *// start the thread*

}

*// This is how the proxy passes messages to the dispatcher*

**void** send\_msg(msg m) **throws** IOException {

Debug.trace(**this**.name+": Sending "+m.getPayload()+" to the dispatcher");

d.send\_msg(**this**, m); *// send to the dispatcher a pointer to the sending proxy object along with the message*

}

*// This is how the proxy receives messages from the dispatcher*

*// This is the callback function that the dispatcher calls*

*// Proxy run method*

**public** **void** run() {

**while**(**true**);

} *//end run method*

**public** **abstract** **void** process\_message(msg m) **throws** IOException;

}

1. **Dispatcher.java**

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** jssc.SerialPort;

**import** jssc.SerialPortException;

*////import java.util.Random; // only for testing*

**public** **class** Dispatcher **implements** Runnable, Subject {

ArrayList<Proxy> proxies;

SerialPortHandle sph;

*//// Random rand = new Random(); // only for testing*

*// Dispatcher constructor*

Dispatcher(SerialPortHandle sph) {

**this**.sph = sph;

proxies = **new** ArrayList<Proxy>(); *// create an array list to remember proxies*

**new** Thread(**this**).start(); *// start the thread*

}

*// Add a new proxy to the list of proxies known by the dispatcher*

**public** **void** register(Proxy proxy) **throws** IOException {

Debug.trace("Adding " + proxy.name + " as ED" + proxies.size() + " to list of proxies known to " + **this**);

proxies.add(proxy); *// adds the proxy object to proxies list*

}

**public** **void** remove(Proxy proxy) {

**int** i = proxies.indexOf(proxy);

**if** (i >= 0) proxies.remove(i);

}

**public** **void** notifyObserver(**byte** id, **byte** pl) **throws** IOException {

**if** (proxies.size() > 0) *// make sure there is a proxy registered in the dispatcher*

proxies.get(id).process\_message(**new** msg(pl)); *// pass the payload to the proxy with the specified ID (through callback method)*

}

*// This is how the dispatcher sends messages through the (ZigBee) Serial port (in 4 steps)*

**public** **void** send\_msg(Proxy proxy, msg m) **throws** IOException {

*// (1) Check if there are any proxies registered in the dispatcher table*

**if** (proxies.size() == 0) {

Debug.trace("Dispatcher ERROR: Message m="+m.payload+" received from "+proxy.name+", no existing ED to send to!");

**return**;

}

*// (2) Check if the proxy trying to send a message is actually in the dispatcher list of proxies*

**int** index;

**boolean** found = **false**;

**for** (index = 0; index < proxies.size(); index++) {

**if** (proxies.get(index) == proxy) {

found = **true**; *// If it is, retrieve its associated end device ID (ED#) and store in index*

**break**;

}

}

**if** (!found) {

Debug.trace("Dispatcher ERROR: Message m="+m.payload+" received from "+proxy.name+", sending proxy not registered in dispatcher!");

**return**; *// If not, ignore the message send request*

}

*// (3) Embed the ID onto the message and create the "byte version" of the message*

m.setId((**byte**)index);

**byte** x = m.getByte();

*// (4) Send the byte version of the message through Serial port*

String s = Byte.toString(x); *// Convert this byte into string*

sph.printLine(s); *// Because the SerialPort readLine and printLine methods only take in strings*

*// to be changed later on because this conversion is redundant*

Debug.trace("Dispatcher: Message m="+m.payload+" received from "+proxy.name+", sending to ED"+m.id);

}

*// Dispatcher run method*

@Override

**public** **void** run() {

**try** { *//try block that catches any exception*

String buf = "";

**while**(**true**) { *// The run method of the dispatcher simply watches the serial port for any incoming bytes*

*// and processes these bytes before passing them to the proxy (through callback method)*

*/\**

*// This block of code introduces delay to the random byte generation*

*// Only for testing*

*try {*

*Thread.sleep(1000);*

*} catch (InterruptedException e) {*

*// TODO Auto-generated catch block*

*e.printStackTrace();*

*}*

*\*/*

*// Wait for incoming bytes, read if any*

buf = sph.readLine(); *// watch the serial port for any incoming input*

*////byte r = (byte)(Math.abs(rand.nextInt() % 128)); // generates a random byte r from 0 to 127 (inclusive)*

**if** (buf != **null**) {

*// Split received byte (message) into ID and payload components*

**byte** r = Byte.parseByte(buf); *// convert input, if any, from string to byte (needed because readLine() method returns strings)*

*//Debug.trace("Incoming message = "+r);*

**byte** r\_id = (**byte**)((r >> 6) & 3); *// retrieve ID by shifting the byte to the right 6 times (retrieve 2 leftmost bits) and AND with 00000011 to get rid of sign*

**byte** r\_payload = (**byte**)(r & 63); *// retrieve payload by ANDing byte with 00111111 (retrieve 6 rightmost bits)*

notifyObserver(r\_id, r\_payload);

} **else** Debug.trace("Incoming message discarded: no proxies registered in the dispatcher!");

}

} **catch** (Exception e) { *//if any exception is caught, terminate program*

e.printStackTrace();

System.exit(1);

}

} *//end run method*

}

*// Serial Port Handle Class*

*/\**

*Author: Mr. Suresh Radder*

*Date: 14 Oct, 2017*

*\*/*

**class** SerialPortHandle {

SerialPort sp;

String path;

**public** SerialPortHandle(String path) {

**super**();

**this**.sp = **new** SerialPort(path);

;

**this**.path = path;

**try** {

sp.openPort();

sp.setParams(9600, 8, 1, 0);

} **catch** (SerialPortException e) {

*// TODO Auto-generated catch block*

e.printStackTrace();

}*// Open serial port*

}

**public** String readLine() {

StringBuffer string = **new** StringBuffer();

**boolean** quit = **false**;

**while** (!quit) {

**byte**[] buffer;

**try** {

buffer = sp.readBytes(1);

*// Read 1 bytes from serial port*

**if** (buffer[0] != 13) {

string.append((**char**) (buffer[0]));

}

**if** (buffer[0] == 13) {

*// Read the following 10 character*

sp.readBytes(1);

quit = **true**;

}

} **catch** (SerialPortException e1) {

*// TODO Auto-generated catch block*

e1.printStackTrace();

}

}

**return** string.toString();

}

**public** **void** printLine(String s) {

**byte** byteArray[] = s.getBytes();

**try** {

sp.writeBytes(byteArray);

sp.writeByte((**byte**) '\n');

} **catch** (SerialPortException e1) {

*// TODO Auto-generated catch block*

e1.printStackTrace();

}

}

}

1. **Msg.java**

**import** java.io.IOException;

*// msg "Message" objects contain ID (ranging from 0 to 3) and payload (ranging from 0 to 63)*

**public** **class** msg {

**byte** id; *// contains ID of sender/recipient end device (0 to 3, inclusive)*

**byte** payload; *// contains payload of message (0 to 63, inclusive)*

msg(**byte** p) **throws** IOException{ *// msg constructor w/ just the payload (instruction), ID initialized to 0*

**this**.id = 0; *// ID defaulted to 0*

**if** (0 <= p && p <= 63) **this**.payload = p; *// check if specified payload is between 0 and 63*

**else** {

Debug.trace("Exception in msg constructor: Specified payload out of range!");

**throw** **new** IllegalArgumentException(); *// notify calling thread about exception*

}

}

msg(**byte** i, **byte** p) **throws** IOException{ *// msg constructor w/ both ID and payload*

**if** (0 <= i && i <= 3) **this**.id = i; *// check if specified id is between 0 and 3*

**else** {

Debug.trace("Exception in msg constructor: Specified ID out of range!");

**throw** **new** IllegalArgumentException(); *// notify calling thread about exception*

}

**if** (0 <= p && p <= 63) **this**.payload = p; *// check if specified payload is between 0 and 63*

**else** {

Debug.trace("Exception in msg constructor: Specified payload out of range!");

**throw** **new** IllegalArgumentException(); *// notify calling thread about exception*

}

}

**public** **byte** getId() { *// ID getter*

**return** id;

}

**public** **void** setId(**byte** id) **throws** IOException{ *// ID setter*

**if** (0 <= id && id <= 3) **this**.id = id; *// check if specified id is between 0 and 3*

**else** {

Debug.trace("Exception in msg id setter: Specified ID out of range!");

**throw** **new** IllegalArgumentException(); *// notify calling thread about exception*

}

}

**public** **byte** getPayload() { *// Payload getter*

**return** payload;

}

**public** **void** setPayload(**byte** pl) **throws** IOException{ *// Payload setter*

**if** (0 <= pl && pl <= 63) **this**.payload = pl; *// check if specified payload is between 0 and 63*

**else** {

Debug.trace("Exception in msg payload setter: Specified payload out of range!");

**throw** **new** IllegalArgumentException(); *// notify calling thread about exception*

}

}

**public** **byte** getByte() { *// Returns the byte version of the message (combines id and payload into one byte)*

**return** (**byte**)((id << 6) | payload); *// Bitwise leftshift the ID 6 times and OR the result with payload*

}

}

1. **Game.java**

**import** java.io.IOException;

**abstract** **class** Game {

**int** score = 0;

*//boolean scored = false;*

**boolean** jumped = **false**;

**boolean** bottomLED = **false**;

**boolean** stop = **false**;

**void** jumped() **throws** IOException { *// called by acm proxy*

jumped = **true**;

}

**void** landed() **throws** IOException { *// called by acm proxy*

jumped = **false**;

}

**void** bottomLED1() **throws** IOException { *// called by led proxy*

bottomLED = **true**;

}

**void** bottomLED2() **throws** IOException { *// called by led proxy*

bottomLED = **false**;

}

**void** stop() **throws** IOException {

stop = **true**;

}

**void** start(ProxyLED p) **throws** IOException

{

p.send\_msg(**new** msg((**byte**) 4));

}

**void** flashLED3(ProxyLED p) **throws** IOException {

p.send\_msg(**new** msg((**byte**) 3));

}

**abstract** **void** setspeed(ProxyLED p) **throws** IOException;

**public** **void** run()

{

**try** { *// try block catches any exception*

Debug.tracefile("debugtrace.txt"); *// Save console messages to a file (located in project directory)*

*// Announce ED numbers*

Debug.trace("LED is ED0!");

Debug.trace("ACM is ED1!");

SerialPortHandle sph = **new** SerialPortHandle("COM7"); *// Initiate serial port for MS Windows*

*//SerialPortHandle sph = new SerialPortHandle("/dev/cu.usbserial-A901JY5U"); // Initiate serial port for Mac OS*

Dispatcher d = **new** Dispatcher(sph); *// Create a dispatcher*

*// Create and start proxy objects*

ProxyLED led\_proxy = **new** ProxyLED("led\_proxy", 0, d, **this**); *// Proxy object for the LED strip (end device 0)*

ProxyACM acm\_proxy = **new** ProxyACM("acm\_proxy", 1, d, **this**); *// Proxy object for the accelerometer (end device 1)*

**this**.flashLED3(led\_proxy); *// make LED flash 3 times to indicate game start*

**this**.setspeed(led\_proxy); *// set initial LED speed*

**this**.start(led\_proxy); *// start LED light sequence*

**while**(**true**) {

**if** (stop)

{

System.out.println("Game Over! Final score = " + score);

**break**;

}

*//Debug.trace("Jumped = " + this.jumped + " Bottomled = " + this.bottomLED);*

**if** (**this**.bottomLED && **this**.jumped) { *// if LEDs are at the bottom and jump is detected*

score += 10; *// increment score*

System.out.println("You scored! Score = " + score);

**this**.bottomLED = **false**; *// reset LED flag*

**this**.jumped = **false**; *// reset jump flag*

}

**if** (score > 50)

led\_proxy.setSpeed(**new** FastSpeed()); *//if score reaches a threshold, make the game harder*

}

} **catch** (Exception e) { *// terminate the program if exception is caught*

e.printStackTrace();

System.exit(1);

}

}

}

1. **HardGame.java**

**import** java.io.IOException;

**public** **class** HardGame **extends** Game **implements** Runnable{

**void** setspeed(ProxyLED p) **throws** IOException

{

p.setSpeed(**new** SlowSpeed2());

p.performSpeed();

Debug.trace("LED speed set to SLOW");

}

}

1. **EasyGame.java**

**import** java.io.IOException;

**public** **class** EasyGame **extends** Game **implements** Runnable{

**void** setspeed(ProxyLED p) **throws** IOException

{

p.setSpeed(**new** SlowSpeed());

p.performSpeed();

Debug.trace("LED speed set to SLOW");

}

}

1. **Driver.java**

**import** java.io.IOException;

**public** **class** Driver {

**public** **static** **void** main(String[] args) **throws** IOException, InterruptedException {

System.out.println("Welcome to Jumping Jackpot!");

System.out.println("Select Game Mode: ");

System.out.println("1 - Easy");

System.out.println("2 - Hard");

**int** x = System.in.read(); *// user input for game difficulty*

x = x - 48; *// convert from char to int*

System.out.println("Your input:" + x);

**switch** (x) { *// selects game difficulty based on user input*

**case** 1:

System.out.println("Game difficulty set to easy!");

EasyGame easy = **new** EasyGame();

Thread t1 = **new** Thread(easy);

t1.start();

**break**;

**case** 2:

HardGame hard = **new** HardGame();

Thread t2 = **new** Thread(hard);

t2.start();

**break**;

**default**:

}}}

1. **FastSpeed**

**import** java.io.IOException;

**public** **class** FastSpeed **implements** Speed{

**public** **void** speed(ProxyLED proxyled)

{

**try** {

Debug.trace("Sending 2 for fast speed...");

proxyled.send\_msg(**new** msg((**byte**)2)); *// 2 is fast speed for LED end device*

} **catch** (IOException e) {

*// TODO Auto-generated catch block*

e.printStackTrace();

}

}

}

1. **Observer.java**

**import** java.io.IOException;

**public** **interface** Observer {

**public** **void** process\_message(msg m) **throws** IOException;

}

1. **ProxyACM.java**

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**public** **class** ProxyACM **extends** Proxy {

ProxyACM(String name, **int** i, Dispatcher d, Game g) **throws** IOException {

**super**(name, i, d, g);

*// TODO Auto-generated constructor stub*

}

**public** **void** process\_message(msg m) **throws** IOException {

*// In this case "processing" is simply printing out message*

Debug.trace(**this**.name+": Message "+m.payload+" received from dispatcher");

**switch**(m.payload) {

**case** 1:

g.jumped();

**break**;

**case** 2:

g.landed();

**break**;

**default**:

;

}

}

}

1. **ProxyLED.java**

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**public** **class** ProxyLED **extends** Proxy {

Speed speed;

ProxyLED(String name, **int** i, Dispatcher d, Game g) **throws** IOException {

**super**(name, i, d, g);

*// TODO Auto-generated constructor stub*

}

**public** **void** setSpeed(Speed sp)

{

**this**.speed = sp;

}

**public** **void** performSpeed()

{

**try** {

Debug.trace("Performing speed!");

} **catch** (IOException e) {

*// TODO Auto-generated catch block*

e.printStackTrace();

}

speed.speed(**this**);

}

**public** **void** process\_message(msg m) **throws** IOException {

*// In this case "processing" is simply printing out message*

Debug.trace(**this**.name+": Message "+m.payload+" received from dispatcher");

**switch**(m.payload) {

**case** 1:

g.bottomLED1();

**break**;

**case** 2:

g.bottomLED2();

**break**;

**case** 3:

System.out.println("GOT MESSAGE FOR GAME OVER!");

g.stop();

**default**:

;

}

}

}

1. **SlowSpeed.java**

**import** java.io.IOException;

**public** **class** SlowSpeed **implements** Speed {

**public** **void** speed(ProxyLED proxyled)

{

**try** {

Debug.trace("Sending 1 for slow speed...");

proxyled.send\_msg(**new** msg((**byte**)(1))); *// instruction 1 is slow speed for arduino*

} **catch** (IOException e) {

*// TODO Auto-generated catch block*

e.printStackTrace();

}

}

}

1. **SlowSpeed2.java**

**import** java.io.IOException;

**public** **class** SlowSpeed2 **implements** Speed {

**public** **void** speed(ProxyLED proxyled)

{

**try** {

Debug.trace("Sending 4 for slow speed...");

proxyled.send\_msg(**new** msg((**byte**)(4))); *// instruction 4 is slow speed for LED arduino*

} **catch** (IOException e) {

*// TODO Auto-generated catch block*

e.printStackTrace();

}

}

}

1. **Speed.java**

**public** **interface** Speed {

**public** **void** speed(ProxyLED proxyLED);

}

1. **Subject.java**

**import** java.io.IOException;

**public** **interface** Subject {

**public** **void** register(Proxy proxy) **throws** IOException;

**public** **void** remove(Proxy proxy);

**public** **void** notifyObserver(**byte** id, **byte** pl) **throws** IOException;

}

1. **Java Code Explanation**

…

**if** (**this**.bottomLED && **this**.jumped) {

score += 10;

System.out.println("You scored! Score = " + score);

**this**.bottomLED = **false**;

**this**.jumped = **false**;

}

…

Figure 9 – Game Thread

BottomLED is defined when the LEDs are on the bottom of the ring and Jumped is defined to be true when the individual jumps correctly. When both of these variables are true then the score increments by 10. It outputs “You scored! Score =” and the player’s current score. Both the variables are set to false to start a new loop of the game.

…

d.register(this);

…

Figure 10 - Proxy Constructor

As the figure above shows, in the constructor, the proxy registers itself into a dispatcher.

…

Proxy(String name, InputStream in, OutputStream out, Dispatcher d) **throws** IOException

…

Figure 11 - Proxy Constructor

The proxy constructor takes in 4 arguments: the string name of the proxy, an input stream, an output stream, and Dispatcher object. This allows us to create a proxy in order to be able to register our end devices and keep track of messages.

…

d.send\_msg(**this**,**new** msg(b));

…

Figure 12 - Proxy Send Message

“b” is a data variable which reads in the bytes from a buffered input stream. The send\_msg method takes in two arguments: a proxy object and message m. This message block combines the data bytes “msg” that you obtain from the game thread to with the proxy that received it and sends a pointer to the dispatcher and awaits for the call-back method “**public** **void** process\_message(msg m)”.

…

**public** **void** process\_message(msg m) **throws** IOException

…

Figure 13 - Dispatcher Process Message Method

The process message method simply awaits a call from the dispatcher for any messages. This is the call-back function of the proxy that the dispatcher interrupts in order to interact and send back messages to the proxy.

…

**public** **void** register(Proxy proxy) **throws** IOException

…

Figure 14 - Dispatcher Register Method

The register method takes in a Proxy object as an argument and registers new entries of proxies in the array list. This allows the dispatcher to understand how many end devices are connected and where the data message should go to during call-back.

…

**public** **void** send\_msg(Proxy proxy, msg m) **throws** IOException

...

**if** (proxies.size() == 0)

…

**for** (index = 0; index < proxies.size(); index++)

…

m.setId((byte)index);

byte x = m.getByte();

…

proxies.get(r\_id).process\_message(new msg(r\_payload));

…

Figure 15 - Dispatcher steps to send message

In the dispatcher class, we begin to receive data bytes from the hardware and now we need to perform a call-back. Firstly, we need to check for the proxy (a) ensure that there are proxies registered in the dispatcher table (b) check that the proxy that is attempting to send a message is actually in the dispatcher array list of proxies. If the proxy is not found on the list, we will ignore the message send request; otherwise, we will take the message “generated data of the hardware” and embed it within the ID to create the byte version of the message. Finally, we will send the byte version of the message through the serial port.

…

msg(**byte** i, **byte** p) **throws** IOException

…

Figure 16 - msg constructor

The message constructor consists of two arguments: the ID number “i” required in order for the dispatcher to map the pointers and find the correct proxy and the payload “p” which is the instruction. ID represents the 2 leftmost bits of the byte; hence, can be 0 (00) or 1 (01) or 2 (10) or 3 (11). Payload is set to be 6 digits ranging from 000000 (instruction 0) to 111111 (instruction 63).

…

**public** **byte** getByte()

…

Figure 17 - msg getByte() method

The method getByte() simply combines the ID and payload and returns one byte. For example, if the end device ID is 1 (01) and instruction (payload) is 43 (101011) then getByte() will return 107 (01101011).

Table 1: Table of Instructions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of Instruction** | **LED to Game Thread** | **ACM to Game Thread** | **Game Thread to LED** | **Game Thread to ACM** |
| 0 |  |  |  |  |
| 1 | BottomLED1 | Jumped | Slow |  |
| 2 | BottomLED2 | Landed | Fast |  |
| 3 | Game Over |  | Flash LED x 3?? |  |
| 4 |  |  | Game Start | Game Start |
| 5 |  |  | MaxSpeed |  |

A close up of a white wall

Description automatically generated

Figure 18: Observer Pattern

A close up of text on a white surface

Description automatically generated

Figure 19: Proxy Subclass

A picture containing text, building

Description automatically generated

Figure 20: Strategy Pattern

A close up of text on a white background

Description automatically generated

Figure 21: Template Pattern

1. **Java Design Patterns**
2. **Template-Method Pattern**

The template pattern was implemented to set the difficulty level of the game. The base class of the template pattern is the Game.java. The recipe of the template method is:

* **this**.flashLED3(led\_proxy);
* **this**.setspeed(led\_proxy);
* **this**.start(led\_proxy);

The flashLED3 method will allow all the LEDs to flash three times to signal the end of the game and the start method will indicate the beginning of the game which is a behavior the EasyGame and HardGame have in common. However the setspeed is a method that is implemented in the EasyGame and HardGame but with different implementations. In EasyGame, we will set the speed to slow while the HardGame sets it to Fast.

1. **Strategy Pattern**

The strategy pattern is implemented for the various speeds of the LED lights – Slow speed and Fast speed. Speed is defined as a common interface to all supported concrete classes – Slow Speed and Fast Speed. These concrete classes each define a specific behavior. The SlowSpeed slows down the speed of the LED lights and FastSpeed makes the LEDs go faster.

1. **Observer Pattern**

The behavioral design patter, Observer, contains the subject and the observer. The subject will notify the observers about the change in its state. In our case, the Dispatcher.java is the subject while the Proxy.java is the observer. The dispatcher notifies the observer about the changes through the calling of the process\_message(msg m) method which is in the observer class. The subject class has three main methods:

* **public** **void** register(Proxy proxy) **throws** IOException; // Adds an observer to the internal list of observers.
* **public** **void** remove(Proxy proxy); // Deletes an observer from the internal list of observers.
* **public** **void** notifyObserver(**byte** id, **byte** pl) **throws** IOException; // Checks the internal flag to see if the observable has changed state and notifies all observers.

1. **YouTube Video Link**

<https://www.youtube.com/watch?v=TXHsaJ-ttBE>