ASSIGNMENT 3 – CS1810 – SOFTWARE IMPLEMENTATION - 2023/24

ROBOT TASK NO.2 – TRAFFIC LIGHT SYSTEM

STUDENT NAME: MOIZ DIN

STUDENT ID: 2221800

GROUP: B43

TUTOR: ROB MACREDIE

CONTENTS……PAGE NO

Implementation Summary

In this section, I'll go over how well the code meets the requirements set out in the design report as well as any changes made to the overall design. The Swiftbot is able to detect lights within 20cm or less, ultrasound is used to detect the Swiftbot and the light. Both these requirements were met. The swiftbot is able to take a picture using the camera and uses a matrix to calculate the light detect. This requirement was also met, try and catch methods were also used in case the camera fails. The user is able to start the program with button A. The additional requirement Idle state was also met, where the swiftbot returns to its idle state after every instance, to keep the flow of the program consistent. All colour detections scenarios (sub-requirements R1, R2, R3) were also met. The program should stop when the user enters "X" button. Technically this requirement was met but it was slightly altered with the additional requirement I specified, where the user will be prompted every 3 traffic light s(used a modulus function on the colour_array), to end the program, in which the user does press X to terminate the program. This gave the program a little more consistency rather than the user being able to press X at any time, this could have affected variables in the log had the swiftbot been stopped in the middle of a detection loop. All requirements within the log information were met as well as well as saving to a text file that stores all these variables. Ultimately, all the functional and non-functional requirements were met as well as the additional requirements.

A couple variable names were changed as you will see through my code, but the structure of the code matches that of the flowchart. The main things that will be noticeable will be the 1. The file path for the image, the one specified in the swiftbot and the one used in the code are different. 2. There were some errors with thread.sleep(500) being used with the swiftbot code, so swiftbot(0,0,500) was used instead. The RGB values for blue and green are swapped, turns out its RBG not RGB as I had assumed. This may have been an API issue however I'm not sure.

The Z Boolean loop used in the savelog function as per my flowchart isn't required as the buttons execute code specified under them so it is not needed. Buttons were swapped with A and B as using the same buttons twice kept coming up with double function errors. In my flowchart, there was also a mistake where in the Blue() function, the swiftbot movements end up turning 90 degrees right instead of left, this was changed in the code to meet the requirements as well.

In summary, I have met all the requirements listed out, the only thing I could have included even though I didn't specify it as a requirement, was saving the file a little neater. Opening it shows the information clunked together. Whilst this may be okay as it was not specified as a requirement by the assignment or myself, in industry, it would be important to consider how data is stored

Testing

Requirement	Input	Output	P/F (Pass/Fail?)	Comments?
Press A to start program	User presses A	Button A Press, swiftbot goes into idle state	P	
Swiftbot responds to green	Green light shown in front of camera	Swiftbot moves to the behavior of green, but shows blue light	F	Blue light shown instead of green
Swiftbot responds to Blue	Blue light shown in front of camera	Swiftbot moves to behaviour of blue, but blinks green light	F	Green light shown instead of green, swap the values
Swiftbot responds to green (repeat)	Green light shown in front of camera	Swiftbot now shows green light & behaviour	P	
Swiftbot responds to blue (repeat)	Blue light shown in front camera	Swiftbot now blinks blue light & behaviour	P	Problem fixed, colour matrix and detect colour works properly
Swiftbot responds to red	Red light shown in front of camera	Swiftbot shows red light and stops	P	
Idle state (additional requirement)		Yellow light shown and bot moves at a slow pace	P	

5cm Ultrasound test	Light shown within 5cm distance	Swiftbot responds to correct light	P	
10cm ultrasound test	Light shown within 10cm distance	Swiftbot responds to correct light	P	
15 ultrasound test	Light shown within 15cm distance	Swiftbot doesn't detect light properly	F	Detected Blue light (maybe from my wall), instead of green
15 ultrasound test (repeat)	Light shown within 15cm distance	Swiftbot respond to correct light	P	

20 ultrasound test	Light shown within 20cm distance	Swiftbot doesn't respond correctly to light	F	
20 ultrasound test (repeat)	Light shown within 20cm distance	Swiftbot doesn't respond correctly to light	F	
20 ultrasound test (repeat)	Light shown within 20cm distance	Swiftbot responds correctly to light	P	Ultrasound sensor is less consistent at longer distances?
5cm ultrasound test (repeat)	Light shown within 5cm distance	Swiftbot correctly responds to light	P	Better at shorter distances

Savelog method test	saveLog method	Error: Writing to file (repeated 100+ times due to recursion technique)	F	File path not found?
Savelog method test (repeat)	saveLog method	LOG SUCCESSFULLY SAVED TO SYSTEM, PROGRAM ENDED	P	File path was missing a \ at the start, problem fixed
Log method	Press B TO DISPLAY JOURNEY LOG	System_Log Number of times light was …encountered was …. Duration was … Most frequent light was… Number of occurrences for frequent light was….	P	System log outputted successfully
Camera method	Camera takes still colour picture	PICTURE TAKEN SUCCESSFULLY	P	

RUNBOTHATTHESAMETIME()	Ultrasound value from method	No output, but behaviour of swiftbot is successful	P	Use of threading was successful
END method PRESS B TO DISPLAY JOURNEY LOG	User presses B	LOG SUCCESSFULLY SAVED TO SYSTEM, PROGRAM ENDED		Log is showed, and the log is saved successfully.
END method PRESS A TO TERMINATE PROGRAM	User presses A	LOG SUCCESSFULLY SAVED TO SYSTEM, PROGRAM ENDED		Log isn't showed, log is saved successfully

Modulus function (additional requirement)	Behaviour: every 3 colour stops user is prompted to end program	Colour array % 3 == 0 code works, and code is rerouted appropriately	P	Structured prompts better for UI and variables than just pressing X at any given time

Monitoring

Date	**Task List**	**% Done**
15/01/24 –	Program Green, Blue, Red	100
22/01/24 –	Program Colour Matrix, Ultrasound,	66
29/01/24 –	Finish Ultrasound, Program Camera, Detect Colour	100

05/01/24 – Program Log, save_log 100

12/01/24 – Program Main, Idle state 33

19/01/24 – Program End, Finish Main, Finish Idle_state 100

26/01/24 – Debug & Check over comments 100

Submission: 01/03/24

Summary

Due to a late start date, due to some swiftbot issues concerning “magic values”, I had to press finishing some methods together within the same week. In week 1, I was able to program the green, blue and red functions successfully. In week 2, I struggled slightly with the ultrasound, mainly just due to it having multiple errors but was able to finish it in week 3. By week 5, I had all my side methods finished and it was time to piece together the code, I struggled due to not knowing how to run both ultrasound and moving the swiftbot at the same, but after learning about threading I was able to finish in week 6. In the final submission week 7, I just went over any logic errors, and checked over my comments and made sure my program was understandable.

Overall, I stuck to the timetable pretty well, and didn’t have to rush with anything. In the end my program was successful in achieving the requirements. Had I maybe started one week earlier, I could have had the chance to add some more additional requirements or do an environmental test of the swiftbot in different locations.

Source Code

```
import swiftbot.*; //list of all my imports

import javax.imageio.ImageIO;

import java.awt.image.BufferedImage;

import java.io.BufferedWriter;
```

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class DoesMySwiftBotWork {
   static SwiftBotAPI swiftBot;
   private static List<String> colour_array = new ArrayList<>(); //list of all my global variables
   private static String Traffic_Colour = "";
   private static long startTime = 0;
   private static double time = 0;
   private static String frequent = "";
   private static int num_times = 0;

   public static void main(String[] args) throws InterruptedException {
      swiftBot = new SwiftBotAPI();
      System.out.println("PRESS BUTTON A TO START PROGRAM");
      swiftBot.enableButton(Button.A, () -> {
         System.out.println("Button A HAS BEEN PRESSED");
         swiftBot.disableButton(Button.A);
         long startTime = System.currentTimeMillis();
         IDLE_STATE();
      });
   }
```

```
    public static void IDLE_STATE() { //idle state exactly shown in flowchart, pieces the code together
            swiftBot.disableAllButtons();
        System.out.println("SWIFTBOT IS SWITCHING TO IDLE STATE");
        int[] yellowLight = {255, 0, 255};
        swiftBot.fillUnderlights(yellowLight);
        RUNBOTHATTHESAMETIME();
        System.out.println("LIGHT DETECT");
        Camera();
        COLOUR_MATRIX();
        if (Traffic_Colour == "r") {
            Red();
            colour_array.add("r");
        } else if (Traffic_Colour == "b") {
            Blue();
            colour_array.add("b"); //adds detected colours to colour array
        } else {
            Green();
            colour_array.add("g");
        }
        int multiple = colour_array.size();
        System.out.println(multiple);
        if (multiple % 3 == 0) {
             System.out.println("Would you like to end the program (Y/X)?");
          swiftBot.enableButton(Button.Y, () -> {
            System.out.println("Button Y HAS BEEN PRESSED");
            swiftBot.disableButton(Button.Y);
            END();
```

```
        });
        swiftBot.enableButton(Button.X, () -> {
            System.out.println("Button X HAS BEEN PRESSED");
            swiftBot.disableButton(Button.X);
            IDLE_STATE();
        });
    } else {
        IDLE_STATE();
    }
}

public static void END() {
    long endTime = System.currentTimeMillis(); //end is made as a seperate method from main, due to button errors
    time = ((endTime - startTime)/1000);
    System.out.println("PRESS B TO DISPLAY THE JOURNEY LOG, PRESS A TO TERMINATE THE PROGRAM");
    swiftBot.enableButton(Button.B, () -> {
        System.out.println("Button B HAS BEEN PRESSED");
        swiftBot.disableButton(Button.B);
        LOG();
        SAVE_LOG(time);
        System.out.println("LOG SUCCESSFULLY SAVED TO SYSTEM, PROGRAM ENDED");
        System.exit(0);
    });
    swiftBot.enableButton(Button.A, () -> {
        System.out.println("Button A HAS BEEN PRESSED");
        swiftBot.disableButton(Button.A);
        SAVE_LOG(time);
```

```
        System.out.println("LOG SUCCESSFULLY SAVED TO SYSTEM, PROGRAM ENDED");
        System.exit(0);
    });
}


public static void RUNBOTHATTHESAMETIME() {
    Thread moveThread = new Thread(() -> swiftBot.startMove(100, 100));
    Thread ultrasoundThread = new Thread(() -> {
        Ultrasound();
    });

    moveThread.start();
    ultrasoundThread.start(); //runs ultrasound and swiftbot movement at same time

    try {
        moveThread.join();
        ultrasoundThread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}


public static void Ultrasound(){
         boolean shouldStop = false;
         while (shouldStop = false) {
        double ultrasoundValue = swiftBot.useUltrasound();
```

```
        if (ultrasoundValue <= 5) { //if ultrasound value is <5, light is detected
            shouldStop = true; //loop breaks and the swiftbot stops
            swiftBot.stopMove();
        }
        try {
            Thread.sleep(1000); // Adjust sleep time as needed
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public static void Camera(){
    try{
        BufferedImage image = swiftBot.takeStill(ImageSize.SQUARE_144x144);

        if(image == null){
            System.out.println("ERROR: Image is null");
            System.exit(5);
        }
        else{
            //saves coloured images to the swiftbot in a filepath
            ImageIO.write(image, "png", new File("/home/pi/colourImage.png"));

            System.out.println("PICTURE TAKEN SUCCESSFULLY");
        }
    }
    catch (Exception e){
```

```
            System.out.println("ERROR WHILE TAKING PICTURE...TRYING AGAIN");
            Camera(); //recrusion, itll call itself and try again
        }
    }

    public static void COLOUR_MATRIX() {
        String imagePath = "/home/pi/colourImage.png"; //path where the colored imaged gets saved
        try {
            File file = new File(imagePath); //gets the image from the filepath and stores it as file
            BufferedImage image = ImageIO.read(file); //read the file
            int[][][] rgbMatrix = new int[image.getWidth()][image.getHeight()][3]; //gets the height and width and stores it as rgb matrix
            for (int y = 0; y < image.getHeight(); y++) { //for each y row, get every x pixel
                for (int x = 0; x < image.getWidth(); x++) {
                    int rgb = image.getRGB(x, y); //determine the rgb value
                    rgbMatrix[x][y][0] = (rgb >> 16) & 0xFF; // red
                    rgbMatrix[x][y][1] = (rgb >> 8) & 0xFF;  // green
                    rgbMatrix[x][y][2] = rgb & 0xFF;        // bue
                }    //stores the rgb value at of each pixel in every row, until its done
            }
            Traffic_Colour = DETECT_COLOUR(rgbMatrix); //takes rgb matrix over to the detect colour method
        } catch (IOException e) {
            System.err.println("ERROR WHILE READING.TRYING AGAIN");
            COLOUR_MATRIX(); //recursion
        }
    }
```

```
    public static String DETECT_COLOUR(int[][][] colour_matrix) {
        int total_pixels = colour_matrix.length * colour_matrix[0].length;
        int redSum = 0, greenSum = 0, blueSum = 0; //counters for rgb
        for (int[][] row : colour_matrix) {
            for (int[] pixel : row) { //for each pixel in every row in the colour matrix
                redSum += pixel[0]; //count if its red green or blue
                greenSum += pixel[1];
                blueSum += pixel[2];
            }
        }
        int avgRed = redSum / total_pixels; //average number of every coloured pixel
        int avgGreen = greenSum / total_pixels;
        int avgBlue = blueSum / total_pixels;
        if (avgRed > avgGreen && avgRed > avgBlue) {  //same as the flowchart, if statement
that returns the value and stores it as traffic colour
            return "r";
        } else if (avgGreen > avgRed && avgGreen > avgBlue) {
            return "g";
        } else {
            return "b";
        }
    }


    public static void SAVE_LOG(double x) {
        String filePath = "/home/pi/Documents/log.txt"; //filepath of the log file
        int num1 = colour_array.size();
        double num2 = x; //time value
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
```

```
            writer.write(String.valueOf(num1)); //ADDS HOW MANY TRAFFIC LIGHT ENCOUNTRED
TO FILE
            writer.write(String.valueOf(num2)); //ADD TIME TO FILE
            writer.write(String.valueOf(frequent)); //adds most frequent color
            writer.write(String.valueOf(num_times)); //adds number of times
            System.out.println("Data has been written to the file.");
        } catch (IOException e) {
            System.err.println("Error writing to the file");
            SAVE_LOG(x);
        }
    }

    public static void LOG() {
        int len = colour_array.size();
        int r_count = 0;
        int g_count = 0;
        int b_count = 0;

        for (String x : colour_array) { //for each element in the colour_array
            if (x.equals("r")) {
                r_count++;
            } else if (x.equals("g")) { //counters for each light detected in the array
                g_count++;
            } else if (x.equals("b")) {
                b_count++;
            }
        }
```

```
      if (r_count > g_count && r_count > b_count) { //compares the count to find the most
frequent colour
         frequent = "red";
         num_times = r_count;
      } else if (g_count > r_count && g_count > b_count) {
         frequent = "green";
         num_times = g_count;
      } else if (b_count > r_count && b_count > g_count) {
         frequent = "blue";
         num_times = b_count;
      }

      System.out.println("System_Log"); //outputs the system log
      System.out.println("The number of times a light was encountered was " + len);
      System.out.println("\nThe duration was "+ time);
      System.out.println("\nMost frequent traffic light: " + frequent);
      System.out.println("\nNumber of occurrences for the most frequent light: " +
num_times);
   }

   public static void Green() {
      System.out.println("GREEN LIGHT DETECTED");
      int[] greenLight = {0, 0, 255}; //NOTE THE VALUES FOR GREEN AND BLUE ARE RBG NOT
RGB
      swiftBot.fillUnderlights(greenLight);
      swiftBot.move(50, 50, 2000);
      swiftBot.stopMove();
   }
```

```
public static void Red() {
    System.out.println("RED LIGHT DETECTED");
    int[] redLight = {255, 0, 0};
    swiftBot.fillUnderlights(redLight);
    swiftBot.move(0, 0, 500);
}

public static void Blue() {
    System.out.println("BLUE LIGHT DETECTED");
    int[] blueLight = {0, 255, 0}; //value is rbg not rgb
    swiftBot.move(0, 0, 500);
    swiftBot.fillUnderlights(blueLight);
    try {
       Thread.sleep(500);
    } catch (InterruptedException e) {
        System.out.println("ERROR, retrying");
        Blue();
    }
    swiftBot.disableUnderlights();
    swiftBot.move(100, 0, 2000); //turns left, moves for bit then retraces backwards
    swiftBot.move(50, 50, 2000);
    int leftwheelv = -50;
    int rightwheelv = -50;
    swiftBot.move(leftwheelv, rightwheelv, 2000);
    leftwheelv = -100;
    rightwheelv = 0;
    swiftBot.move(leftwheelv, rightwheelv, 2000);
}
```

```
}
```