CS1704 GROUP PROJECT

CS1814 SOFTWARE Implementation ASSIGNMENT

3

TASK 6: DETECT OBJECT

NAME: ATHAR HUSSAIN

STUDENT iD: 2456634

Tutor: DR. LORRAINE AYAD

GROUP B38

Key:

Red – To be changed

Green - Additions

Blue – To be omitted

Functional requirements:

1. General program functions

Startup & base functioning

1. The program should display the following message in the console when started:

`"Hi there Welcome to Swiftbot Detect Object!"`

2. The program should allow the user to select a mode by scanning a corresponding text based QR codes to enter one of the following modes: Curious SwiftBot mode, Scaredy SwiftBot mode, Dubious SwiftBot mode

3. After a QR code is scanned and validated, the program should notify the user which mode they are using on the console window with the message

“You are in mode {MODE_NAME].”

4. The program regardless of the mode when ‘wandering’ (has not detected an object) should have the underlights set to blue

5. When replacing the object at a new distance from the SwiftBot, the User should ensure that it is placed either forward or backwards from the original position and that the new position is still in the possible line of sight of the Swiftbot i.e. not behind or to the right or the left of the SwiftBot

5. The SwiftBot will only detect objects that are infront of it being unable to detect objects behind or to the side of it

Termination & Execution Log

6. The user should be able to terminate the program by pressing the 'X' button on the SwiftBot Regardless of the mode

7. Before termination, the program should ask the User via the console window:

"Would you like to view the execution log? Press the Y button for Yes and the X button for No"

8. If the User presses the 'Y' button, the console should display:

- The mode used

- Duration of execution of program

- Number of times the Swiftbot encountered an object
- File path where images taken were saved

- File path where log file is saved.

8. If the user presses 'X'. The program should then only display the file path where the log file will be saved and then save the execution log to that location

Mode- Specific Functional requirements

Curious SwiftBot Mode

9. When an object is detected, the underlights should turn green

10. The SwifBot should maintain a 30 cm buffer zone between itself and the object, known as the 'Buffer zone'

11. If the object is exactly 30cm away, the SwiftBot should do the following things in the order they are mentioned:

- Switfbot should stop moving

- blink the underlights green for 3 seconds

- Turn off the underlights

- Remain stationary

12. If the object is placed inside the buffer zone the SwiftBot should:

- Move backwards until the Buffer zone is reformed

- Then perform requirement 11

13. If the object is placed outside the buffer zone the SwiftBot should:

- Move forwards until the buffer zone is formed

- Then perform requirement 11

14. Once the bufferzone is formed the SwiftBot should:

- Take an image of the object

- Save the image under the appropriate file path

- It should then wait for 5 seconds and check if the object has moved positions via image size comparison

15. If after five seconds, the object has not moved the program should reorientate the swiftbot 20 degrees towards the right

- Then move forward at a speed of 40 for 30 cm until it encounters another object or 5 seconds has passed
- If it has not encountered an object for 5 seconds it should remain stationary for a second and again reorient itself 20 degrees towards the right and move in that direction continuing this loop until it encounters an object

Scaredy SwiftBot Mode

16. The Swiftbot should aim to maintain a distance of 50 cm between itself and an object

17. When an object is detected, the SwifBot should turn the underlights red

18. When the object is within fifty cm, the SwiftBot should:

- first take an image of the object and save it

- then blink the underlights of the SwiftBot red

- finally reverse for 3 seconds and then turn 180 degrees to the right and then set the underlight colour back to blue and move in the direction it is now placed for at a speed of 40 to try to encounter another object

- Reverse for 1 second

- turn approximately 180 degrees to the right

- Move forward for 3 seconds

- set the underlight colour back to blue

19. A loop should occur where If after five seconds an object has not been encountered, the SwiftBot should:

- remain stationary for one second

- then reorientate 20 degrees to the right

- Move in that direction for 50 cm for until it encounters an object breaking the loop at a speed of 40 or 5 seconds pass whereby the loop should restart

Dubious SwiftBot Mode

20. When this mode is selected the program should:

- Randomly select and enter either the 'Curious' or 'Scaredy' SwiftBot mode

- Execute the selected mode's logic

Non-functional requirements:

Performance:

21. The Swift Bot should be able to process and detect objects within 2 seconds to promote real time engagement

22. The underlight should change colours within 2 seconds of an object being detected

23. In Curious SwiftBot mode bufferzone calculations should be executed within 3 seconds

24. In Scaredy SwiftBot mode the 180-degree turn should be performed within 3 seconds within 5 seconds

Usability:

25. The system should attempt to scan a QR code up to 3 times before displaying an error message

26. The program should clearly indicate which mode is in use via the colour of the underlights:

- Red for Scaredy SwifBot
- Green for Curious SwiftBot

27. Messages from the console should be clearly visible to the User

28. The program termination process (including the Execution log prompts) should require no more than two buttons

Reliability and Maintainability:

29. The system should be able to detect and handle common errors that may cause unexpected termination and display an appropriate message in the console window to notify the User that an error has occurred

30. The Execution log data should be stored in an existing file path so that it may be easily accessed

Portability:

31. The program should be compatible with all SwiftBot models supporting these functionalities

Security:

32. All Execution Log data should be stored in a secure file path

33. If the program randomly restarts, a new log file should be created to prevent overwriting the previous log file

Scalability:

34. The program should be optimised to keep object detection under 2 seconds in standard conditions

Availability:

35. If the program does not detect an object within 2 minutes, it should notify the User with a message in the console

Error Handling Requirements

36. If the camera cannot capture an image after 5 attempts it should prompt the User to reboot the program by rescanning a QR code

37. If the system fails to detect an object within 2 minutes it should display the following message in the console window:

"No object detected. Please scan and re-adjust environment"

38. If the system encounters an unrecoverable error, it should safely terminate the program

Additional requirements

39. The User should be able to enter a preferred name at the start of the program. The SwiftBot should use this name in console messages when interacting wi given the option to create an in-game name that the SwiftBot should address the User by this name in console messages.

40. The User should have an option to set a maximum number of objects to be detected followed by program termination

Changes made Summary

- Requirement 5 has been changed as its previous iteration highlighted user constraints and so it has been reformatted so that it focuses more on what the system itself is capable of to be in line with system requirements
- Part of requirement 14 has been omitted as it has been brought to light that the system itself uses a sensor to detect the object distance and not image comparison
- Requirements 15 and 19 have been changed from wandering for 30cm to 5 seconds and mention of a loop to be accurate to the original task description provided in the assessment guide
- Requirements 18 has been adjusted from reversing for 3 seconds to first turning 180 degrees and then moving forward for 3 seconds to be accurate to the original task description provided in the assessment guide

Testing

Test Case ID	Requirements Covered	Test Case Description	Test data	Expected Outcome/Behavior	Observed value/behaviour	Status (Pass/Fail)	Comments
1	1	Startup the program with no input	No input provided	Console displays ascii and print message "Hi there Welcome to Swiftbot Detect Object!"	Desired mode logic executed	pass	

2	2	Selecting a mode via the QR code	QR code: “Curious SwiftBot”, QR code: “Scaredy SwiftBot”	Program correctly identifies mode desired and executes program	Desired mode logic executed	pass	QR code had black bakground which prevented detection once changed was easily recognised
3	6	Terminating the program mid execution	Press 'X' button during execution of Curious mode, Press 'X' button during execution of Scaredy mode,Press 'X' button during execution of Dubious mode	Program proceeds to terminate at different parts of the program	Program terminates	pass	
4	4	Enter wandering mode upon starting a mode	Enter Curious mode logic,Enter Scaredy mode logic	Blue underlights visable and object detection occurs	Blue under lights are visable + object detection	pass	
5	12 – 15	Peform curious mode logic	Use an object when in Curious mode	Bufferzone logic is followed	Bufferzone logic occurred	pass	
6	16-18	Perform Scaredy mode logic	Use an object when in Scaredy mode	Underlights turn red,image is taken,swiftbot reverses and turns in opposite direction	Underlights turn red,image is taken,swiftbot reverses and turns in opposite direction	pass	

Changes to algorithm and User interface

Change:

The logic of using image comparison specified in flowchart 1.6 to detect distance of objects has been omitted

Justification:

SwiftBot has an ultrasound to detects distance not image comparison

Change:

Wandering logic that is not previously mentioned in the flow charts has been added to the scaredy and curious modes

Justification:

To allow the program to meet requirement 4 in the software requirement specification

Change:

An image is only taken when the 'bufferzone' has been formed as opposed to calling the saveimage method regardless of the distance mentioned algorithm 1.3 "Curious SwiftBot"

Justification:

For ease of implementation and to allow logic to flow more seamlessly where bufferzone can be quickly formed rather than images constantly being taken after every movement

Planning and monitoring

Task/Milestone	Deliverables	Challenges faced	Solutions and adjustments	Actual Timeline
1. Writing initial code	- Added core logic for each mode - preliminary console I/O flow	-Initial Struggle to convert mode logic into code	-Broke down the logic into smaller problems and then matched programming principles I learnt to these smaller sections of logic	Feb 23 – Mar 10
2. Adding OOP principles	-Encapsulated logic into helper method and modes class to declutter and refine main class	- Attempt to Introduce different classes also introduced a high	-Learnt about basic OOP principles first then came back and continued working on code	Mar 11 – Mar 20

		level of challenge relative to current coding ability		
3. Writing implement ation summary	-Draft implementation summary	-Difficulty Clearly demonstrating what changes were made	-Utilised colour coding key to differentiate new iterations to specification	Mar 21- Mar 22
4. Refining code	-Refined syntax and code structure for all classes	-Braces were not indented cleanly causing unreachable errors	-Analysed code line by line to properly indent braces	Mar 23 – Mar 25
5. Testing	Black box test plan	-including only relevant information in testing table	-Prioritised key features to be included into table	Mar 26 – Mar 27

```
mport java.io.FileWriter;


import java.io.IOException;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.concurrent.atomic.AtomicBoolean;


import swiftbot.Button;

import swiftbot.SwiftBotAPI;

public class Maincode {


public static void main (String []args)

{

System.out.println("\r\n"

+ " ___ __ __ ____ __ _ __ \r\n"

+ " / _ \\___ / /____ ____/ /_ / __ \\/ / (_)__ ____/ /_\r\n"

+ " / // / -_) __/ -_) __/ __/ / /_/ / _ \\ / / -_) __/ __/\r\n"

+ "/____/\\__/\\__/\\__/\\__/\\__/ \\____/_.__/_/ /\\__/\\__/\\__/ \r\n"

+ " |___/ \r\n"

+ "");
```

```
System.out.println("Hi there Welcome to Swiftbot Detect Object!");
System.out.println("Please scan one of the following modes: Curious SwiftBot , Scaredy SwiftBot or Dubious SwiftBot");

// Use try block to prevent code from crashing from unhandled exceptions
try
{
// Call the api
SwiftBotAPI api = new SwiftBotAPI();
// Create object from Mode class to access methods within it
Modes mode = new Modes(api);

// call the method that allows activation of the x button to quit the program at any time
mode.TerminateX();


// Create a string for the purpose of storing the result of reading the QR code
while (true)
{
String chosenmode = mode.QRcode();

// exit if no mode chosen
if (chosenmode == null || chosenmode.isEmpty())
{

System.out.println ("No mode chosen. Please select a mode");
continue;
}
```

```
// create switch block to account for the different scenarios where each of the modes is chosen
switch (chosenmode.toLowerCase())
{ // change the string to lowercase so the switch block can reliably call the right method

case "curious swiftbot":
 mode.modeCuriousSwiftbot();

break;
case "scaredy swiftbot":
mode.modeScaredySwiftbot();
break;
case "dubious swiftbot":
mode.modeDubiousSwiftBot();
break;

// create a default case to notify the user the QR code is not recognisable

default:
System.out.println("Unrecognisable QR code");
}
break;
}
viewExecutionLog(mode);

 }
```

```
catch (Exception e )
{
e.printStackTrace();
}

}

// method asking if user wants to view execution log or not
public static void viewExecutionLog( Modes mode) {

// Call the api
SwiftBotAPI api = mode.getApi();
// Create object from Mode class to access methods within it
// disable buttons first
api.disableButton(Button.X);
api.disableButton(Button.Y);

System.out.println("Would you like to view the execution log? Press Y for 'Yes' and X for 'No'. You have 20 seconds to decide before program termination occurs");

// use AtomicBoolean class to create a boolean flag that we can change the value of within lambda expressions to determine what to do when the User presses either the X and Y buttons

final AtomicBoolean Yes = new AtomicBoolean(false);
final AtomicBoolean No = new AtomicBoolean(false);
```

```
// Enable the Y button for User
api.enableButton(Button.Y, () -> { //Lambda expression
Yes.set(true); // if the user presses Y the bollean flag becomes true
System.out.println("You have selected 'Yes'. Retrieving Log..."); // Console I/O flow
});

// Enable the X button for User
api.enableButton(Button.X, () -> {
No.set(true);
System.out.println("You have selected 'No'. Saving Log and displaying file path only... "); // Console I/O flow
api.disableButton(Button.X);

});
while (true) {
// if statement to handle scenario where User presses Y
if (Yes.get()) {
// call a method that will display the log File and store it in a string and then print the contents
String pressedY = mode.logInfo();
System.out.println(pressedY);
 // shutdown program with status code 0 indicating a regular shutdown
break;

}
// if statement to handle scenario if User presses X
if (No.get()) {
System.out.println("Skipping the log file displaying filepath where logfile will be stored...");
```

```
System.out.println("data/home/pi/logs/executionLog.txt");
break;



}
try {
Thread.sleep(100);

} catch (InterruptedException e) {
e.printStackTrace();
}

}
String logInfo = mode.logInfo();
String filePath = "data/home/pi/logs/executionLog.txt";
makeLogFile(logInfo, filePath);
System.exit(0);




}
private static void makeLogFile(String logInfo, String filePath) {
// make timestamp to act as a identifier/"foreign key" for individual log files
LocalDateTime time = LocalDateTime.now();

DateTimeFormatter format = DateTimeFormatter.ofPattern("yyyyMMdd_HHmmss");
String bookmark = time.format(format);
```

```
String fileName = filePath.replace(".txt", "_" + bookmark + ".txt");
try (FileWriter file = new FileWriter(fileName)) {
file.write(logInfo);
System.out.println("Log file saved to: " + fileName);
} catch (IOException e) {
System.err.println("Error creating Log File: " + e.getMessage());
}

}




}

import swiftbot.SwiftBotAPI;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class HelperMethods {
private SwiftBotAPI api;

public HelperMethods(SwiftBotAPI api) {
this.api = api;
}

public void allUnderLights(int r, int g, int b) {
```

```
int [] rgb = {r,g,b};
api.fillUnderlights(rgb);
}

// helper method for curious mode to blink underlights green for 3 seconds
public void blinkGreenUnderlights() throws InterruptedException {
for (int i = 0; i <3 ; i++) {
//turn the underlights green for half a second
allUnderLights(0,255,0);
Thread.sleep(500);

//turn the underlights off for half a second
api.disableUnderlights();
Thread.sleep(500);
}
}
// helper method for Scaredy mode to blink underlights red
public void blinkRedUnderlights() throws InterruptedException {
for (int i = 0; i <3 ; i++) {
//turn the underlights green for half a second
allUnderLights(255,0,0);
Thread.sleep(500);

//turn the underlights off for half a second
api.disableUnderlights();
Thread.sleep(500);
}
}
```

```
public void saveImage(BufferedImage image, String Filename) {
try {
File file = new File(Filename);
ImageIO.write(image, "jpg", file);
System.out.println("image saved to" + file.getAbsolutePath());

} catch (IOException e) {
System.err.println("error while saving image" + e.getMessage());
}
}
public void rightTurn() throws InterruptedException {
api.move(40, -40, 2000);
api.stopMove();
}

public void moveForward() throws InterruptedException {
api.move(40, 40, 5000);
api.stopMove();
}
}

import swiftbot.*;

import java.awt.image.BufferedImage;
import java.util.Random;
```

```
public class Modes {

// call the api as a class field

private SwiftBotAPI api;


// set a boolean flag that exits out of loops in the mode methods when the user presses x and use volatile to ensure loops in different threads see an updated value

private volatile boolean logLoop = true;


public boolean islogLoop() {

return logLoop;

}

public void logLoop(boolean value) {

logLoop = value;




}


//Declare class fields to be used later by log file


private String userMode;

 private int objectEncounters;

 private long durationExecution;

 private String imageFilePath;

 private String logFilePath;

// create the constructor and the paramater so that we can interact with API

public Modes(SwiftBotAPI api) {

this.api = api;

this.userMode = "";
```

```
this.objectEncounters = 0;

this.durationExecution = 0;

this.imageFilePath = "";

this.logFilePath = "";


}


public SwiftBotAPI getApi() {

return this.api;

}

// method to read QR code

public String QRcode() {

//Declare and Initialise a local variable that will store QR code

String QRMode = "";

try {

Thread.sleep(10000);

System.out.println("Scanning QR Code..."); // I/O flow

BufferedImage QRImage = api.getQRImage(); // create an object from Bufferedimage class and then call method from APi that retrieves the QR image and then store it in the object

System.out.println("get qrImage() method called successfully");

if (QRImage!=null) {

QRMode = api.decodeQRImage(QRImage); // Call the api method to decode QR image and then store the result in the local variable

}

System.out.println("QR code: " +QRMode); // User interface

} catch (Exception e) { // Error handling

System.err.println("Error while reading QR code: " + e.getMessage()); // Error handling

}
```

```
return QRMode.trim();
}
private double objectDetection() {
//create local variable and make fall back distance incase reading fails
double distance = 9999;
try {
// call the ultrasound method
distance = api.useUltrasound();

} catch (Exception e) {
e.printStackTrace();
}
// return the value given by the ultrasound method
return distance;
}

public void wandering()
{
HelperMethods helperMethods = new HelperMethods(api);
// Fulfil requirement of having the underlights set to blue when the robot is 'wandering'
helperMethods.allUnderLights(0, 0, 255); // call the underlights method

boolean wandering = true;
while (wandering)
{

double objectdetection = objectDetection();
```

```
if (objectdetection < 100)
{
System.out.println("Breaking from wander mode!");
wandering = false;
break;
}
else

{
api.move(40, 40, 2000);
}

}

}
public void modeCuriousSwiftbot() {

System.out.println("Welcome to Curious SwiftBot mode!"); // User interface
// update class field for mode being used

// create start time for mode
long start = System.currentTimeMillis();

System.out.println("The robot will begin wandering in 5 seconds please move your phone out the way");
try {
Thread.sleep(5000);
wandering();
```

```
} catch (InterruptedException e) {

e.printStackTrace();
}



// delcare the variables for the Log file
 userMode = "Curious SwiftBot";
 objectEncounters = 0;
 logFilePath = "date.apiaddress/curious_log.txt";
 imageFilePath = "";

// Use try block to prevent code from crashing from unhandled exceptions
try {

HelperMethods helperMethods = new HelperMethods(api);
// Fulfil requirement of having the underlights set to blue when the robot is 'wandering'
helperMethods.allUnderLights(0, 0, 255); // call Underlight method and set the blue to maximum



// create a loop that keeps going so that we can look for objects until the user terminates the program
while (true)
{



// First create a local variable to hold result of objectDetection
```

```
double objectDistance = objectDetection();
System.out.println("Distance from object is: " + objectDistance + "cm"); //User Interface

// IF the object is within a range of 200 cm count this as an object detected

if (objectDistance < 200)
{
objectEncounters++;
}
//
// update classfield for objectEncounter if an object has been detected

// While loop to hold bufferzone scenarios
boolean bufferZone = true;
while (bufferZone)
{
objectDistance = objectDetection();
System.out.println("Distance from object is: " + objectDistance + "cm");
 // Scenario if Swiftbot is roughly at the bufferzone of 30 cm give or take 2 cm

if (Math.abs(objectDistance - 30) < 10)
 {
api.stopMove();
helperMethods.blinkGreenUnderlights();
BufferedImage image = api.takeStill(ImageSize.SQUARE_480x480);
helperMethods.saveImage(image, "CuriousSwiftbotimage.jpg");
bufferZone = false;
break;
```

```
}

// Scenario if SwiftBot is within bufferzone but is greater than 5cm from the object
 else if (objectDistance < 30 && objectDistance > 5)
{
helperMethods.allUnderLights(0, 255, 0);
api.move(-40, -40, 2000);
api.stopMove();

objectDistance = objectDetection();
continue;

}
// Scenario if SwiftBot is outside bufferzone
 else if (objectDistance > 30 && objectDistance < 200)
 {
api.move(40, 40, 2000);
api.stopMove();

objectDistance = objectDetection();
continue;
 }

}
```

```
// pause for 5 seconds to see if object is moved
Thread.sleep(5000);

// check if object has moved
double newDistance = objectDetection();
System.out.println("Distance from object is: " +        objectDistance + "cm");

// if object has "not moved" from the bufferzone then change direction and wander until new object is found
if (Math.abs(newDistance - objectDistance) <= 3)
{
System.out.println("Object has not moved. Changing direction and beginning wandering");
helperMethods.rightTurn();
helperMethods.moveForward();
wandering();
continue;



}
// scenario if object is no longer at bufferzone then reform bufferzone by going back to beginning of While(true) loop
else
{

continue;
}

}
```

```
// scenario where no object is detected within 200 cm



} //error handling
catch (Exception e)
{
e.printStackTrace();
}
finally
{
// track the total time for the program running




// Turn the lights off
api.disableUnderlights();
long end = System.currentTimeMillis();
 durationExecution = end - start;
}



}
```

```
public void modeScaredySwiftbot()
{

System.out.println("Welcome to Scaredy SwiftBot mode!");
// create start time for mode
long start = System.currentTimeMillis();
// delcare the variables for the Log file
userMode = "Scaredy SwiftBot";
objectEncounters = 0;
logFilePath = "date.apiaddress/scaredy_log.txt";
imageFilePath = "";
// Fulfil requirement of underlights set to blue when the robot is 'wandering'

// Use try block to prevent code from crashing from unhandled exceptions
try
{

//Create object from HelperMethod class to access its methods
HelperMethods helperMethods = new HelperMethods(api);
```

```
// use a while (true) loop so that object detection can continue going until the user wants to terminate the program with the x button

while (true)

{

// Fulfil requirement of underlights set to blue when the robot is 'wandering' by calling wandering method

wandering();

//Call the object detection method to get the object distance

double objectDistance = objectDetection();

System.out.println("Distance from object is: " + objectDistance + "cm");



// if the object is within 200 cm count this as an object encounter and turn red

if (objectDistance < 200)

{

objectEncounters++;

helperMethods.allUnderLights(255,0,0);

}

// while loop to deal with scenario where swiftbot is more then 50 cm from the object

boolean scaredyObject = true;

while (scaredyObject)

{

// if statement to fulfill requirement when Swiftbot is more than 50 cm from the object
```

```
if (objectDistance > 50 && objectDistance < 200)
{
System.out.println("Object is " + objectDistance + "cm");
api.move(40, 40, 2000);
api.stopMove();
continue;




}


// if statement for scenario where swifbot is within 50cm of object
else if (objectDistance < 50 && objectDistance > 5)
{


// first take an image of the object
BufferedImage image = api.takeStill(ImageSize.SQUARE_240x240);
helperMethods.saveImage(image, "CuriousSwiftbotimage.jpg");
// then blink the under lights red
helperMethods.blinkRedUnderlights();

// finally reverse away from the object
api.move(-40, -40, 2000);
```

```
// spin in the opposite direction
api.move(40, -40, 1000);

// finally move away for 3 seconds
api.move(40, 40, 3000);
// continue scanning for objects
scaredyObject = false;
break;

}
}

// scenario if no object has been detected within 5 seconds
long noDetection = System.currentTimeMillis();
// create a flag to come out of a while loop if an object is found within 5 seconds

while ((System.currentTimeMillis() - noDetection <5000))
{
objectDistance = objectDetection();
if (objectDistance < 200)
{
System.out.println ("Object found");
break;

}
// short sleep to avoid hammering
Thread.sleep(500);
// if statement to handle scenario if no object is detected within 5 seconds
```

```
 if ((System.currentTimeMillis() - noDetection <5000))
{
System.out.println("No object detected. Reorientating to a new direction!");
helperMethods.rightTurn();
helperMethods.moveForward();
break;
}
}
continue;




}
}
// short break before continuing loop


catch (Exception e) {
e.printStackTrace();

}
api.disableUnderlights();
long end = System.currentTimeMillis();
durationExecution = end - start;
}
```

```
public void modeDubiousSwiftBot() {
System.out.println("Welcome to Dubious SwiftBot!");
Random chanceMode = new Random();
Boolean curious = chanceMode.nextBoolean();



if (curious) {
System.out.println("Curious mode has been selected! Loading program..."); // UI and I/O flow
modeCuriousSwiftbot();
return;

}
else {
System.out.println("Scaredy mode has been selected! Loading program...");
modeScaredySwiftbot();
return;

}
```

```
}
```

```
public void TerminateX()

{

api.enableButton(Button.X, () -> { // Lambda expression which tells the program what to do when the button X is pressed

System.out.println("Terminating program..."); // console I/O flow

logLoop(false);

Maincode.viewExecutionLog(this);

api.disableButton(Button.X);


});


}

// multiline string at end of program to retrieve all the logfile info

public String logInfo() {


StringBuilder log = new StringBuilder();

log.append("Mode used: ").append(userMode).append("\n");

log.append("Objects encountered: ").append(objectEncounters).append("\n");

log.append("Length of execution: ").append(durationExecution).append("\n");

log.append("log file path: ").append( logFilePath).append("\n");

log.append("Image file path: ").append( imageFilePath).append("\n");

return log.toString();   }


}
```