# Cover Page

CS1701

Group Project Lectures & Tutorials

CS1803 – Group Project Reflection

CS1809 – Software Design

CS1810 – Software Design

Software Design

**Task 2 – Traffic Lights**

**Group B33**

**Tutor: Prof. Theodora Koulouri**

***Nathan Singh - 2376781***

# Requirements Specification

## Functional Requirements:

1. The program must display a command prompt window which the user can interact with.
2. To allow for the Swift Bot to navigate its path using colour code input using the Swift Bot's onboard camera.
3. The Swift Bot must be able to respond to traffic lights when the lights are within a range of 20cm.
4. The program must be able to process the photo taken of the traffic light and determine the colour.
5. Once pressed; the ‘A’ button must initiate the program.
6. The Swift Bot should be able to toggle its LED under lights ON and OFF as well as switch between different colours.
7. The Swift Bot should have a predefined ‘low initial speed’ at which it travels at certain intervals.
8. The Swift Bot must be able to come to a halt at certain given points.
9. The Swift Bot must be able to rotate ‘x’ number of degrees in a fixed point to change direction.
10. The Swift Bot should be able to retrace its movement.
11. The Swift Bot must be able to come to a full stop.
12. The program must be stopped when the user presses the “X” button on the Swift Bot.
13. The program must be able to create, open, read to, write to and save a text file; where a display of the execution log will be present.
14. The program must be able to keep count of the types of colours and number of times it visits traffic lights as well as be able to calculate the most frequently visited of the same.
15. The Swift Bot and program code must keep track of time to plot and coordinate certain events during the program's running time.

## Non-Functional Requirements (in accordance with functional requirements above):

2) In order to navigate its path using colour code input, the Swift Bot must take a picture of its environment via the onboard camera and follow set protocol based on the input it detects.

4) One of the methods by which the picture taken (#2) by the Swift Bot can be processed is via a pixel matrix. After this is computed each pixel colour can be independently verified; after which the colour of the traffic light can be concluded and the following sequences can be executed.

7) A predefined low initial speed can be computed using simple mathematics and trial and error to determine a suitable speed. The challenge lies in optimizing the robot’s speed so as to achieve faster and more efficient travel while ensuring it can correctly identify traffic lights within 20cm of them.

14) In order to keep an efficient count of most frequently visited colours during the program runtime, each time a traffic light is passed a value must be added to a tally that can be accurately displayed at the end of the program.

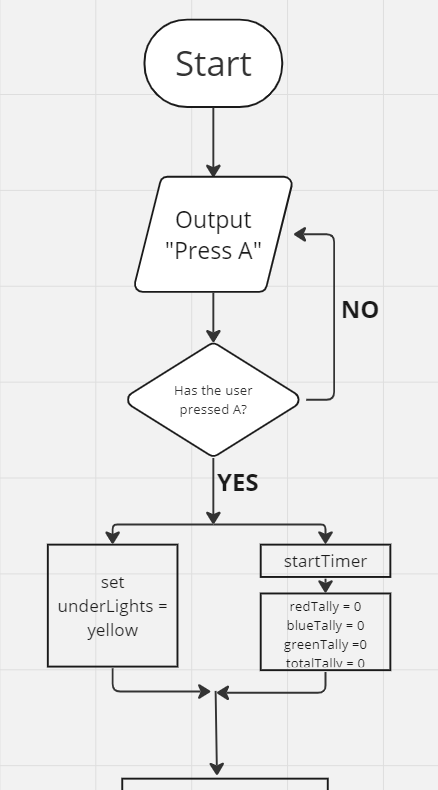
## Omitted Requirements (in accordance with functional requirements above):

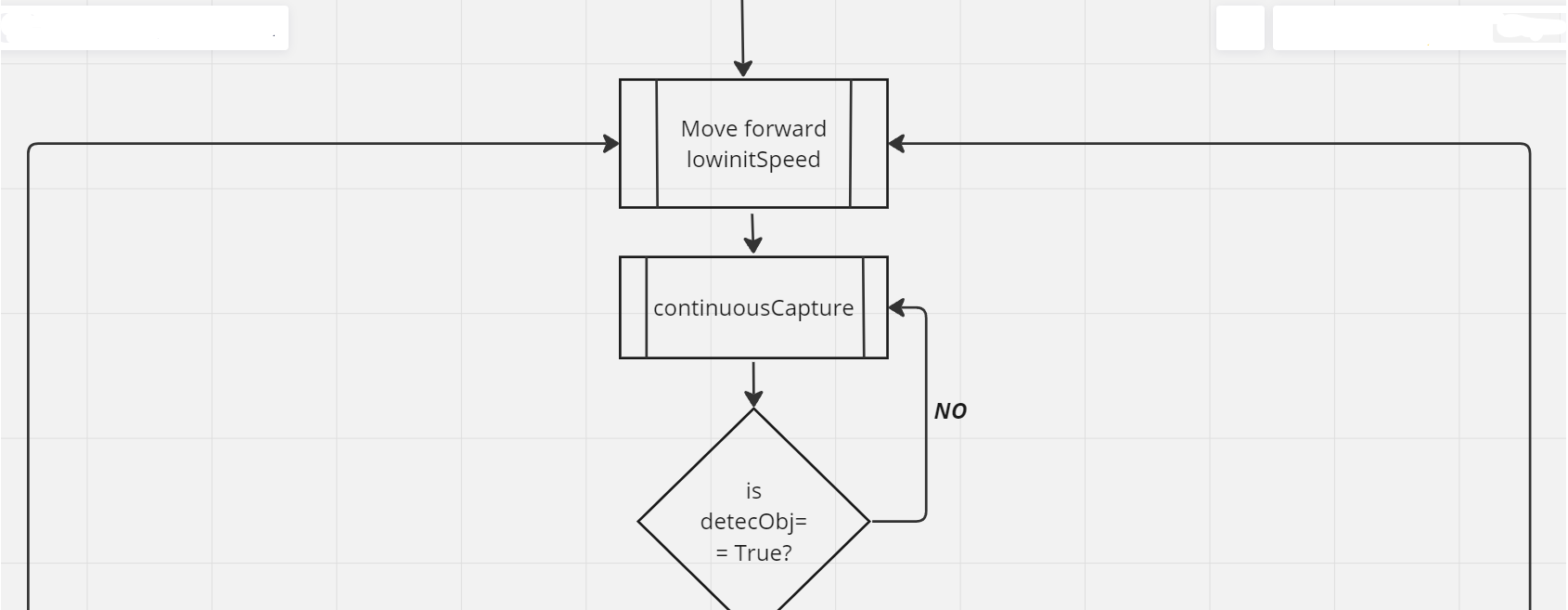
10) The program does not need a special algorithm/method by which it retraces its movement. This is for a couple reasons; namely it is only required in the singular instance when a blue colour traffic light is encountered. Hence, creating an algorithm that retraces the Swift Bot’s last steps is a waste of time and computational power. It makes more sense for the inverse of the movement to be executed.

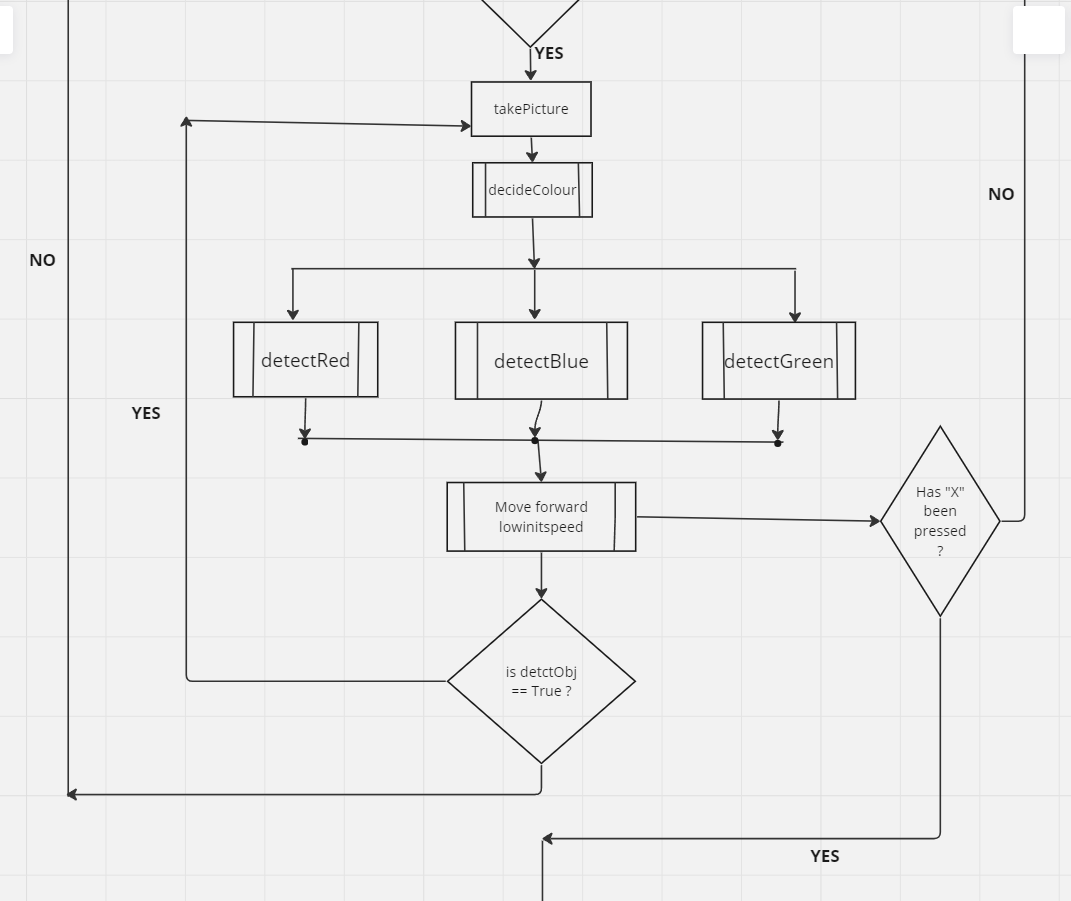
## Error Handling:

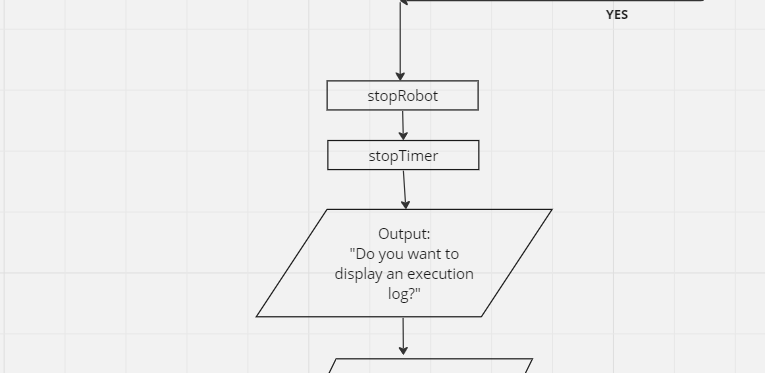
1. After the Swift Bot has identified a traffic colour using the decideColour(): sub procedure there is still room for error in the tally system that calculates the number of times a particular has been visited. The error may lie within the program's ability to be 100% sure that said colour is correct. It may incorrectly identify a traffic lights color due to external factors such as variance in lighting, a slightly dirty camera lens, etc. To make sure individual tally counts and the final execution log is accurate, a tally point is only added within specific colour sub procedures post being individually identified.
2. When prompted to enter either X or Y at the end of the flowchart to decide between displaying the execution log or not. If the user enters either A or B (buttons on the Swift Bot) an error message is displayed and prompts the user to enter X or Y again. This error handling is important to prevent unrecognized input from being fed into the system; hence possibly crashing the system.

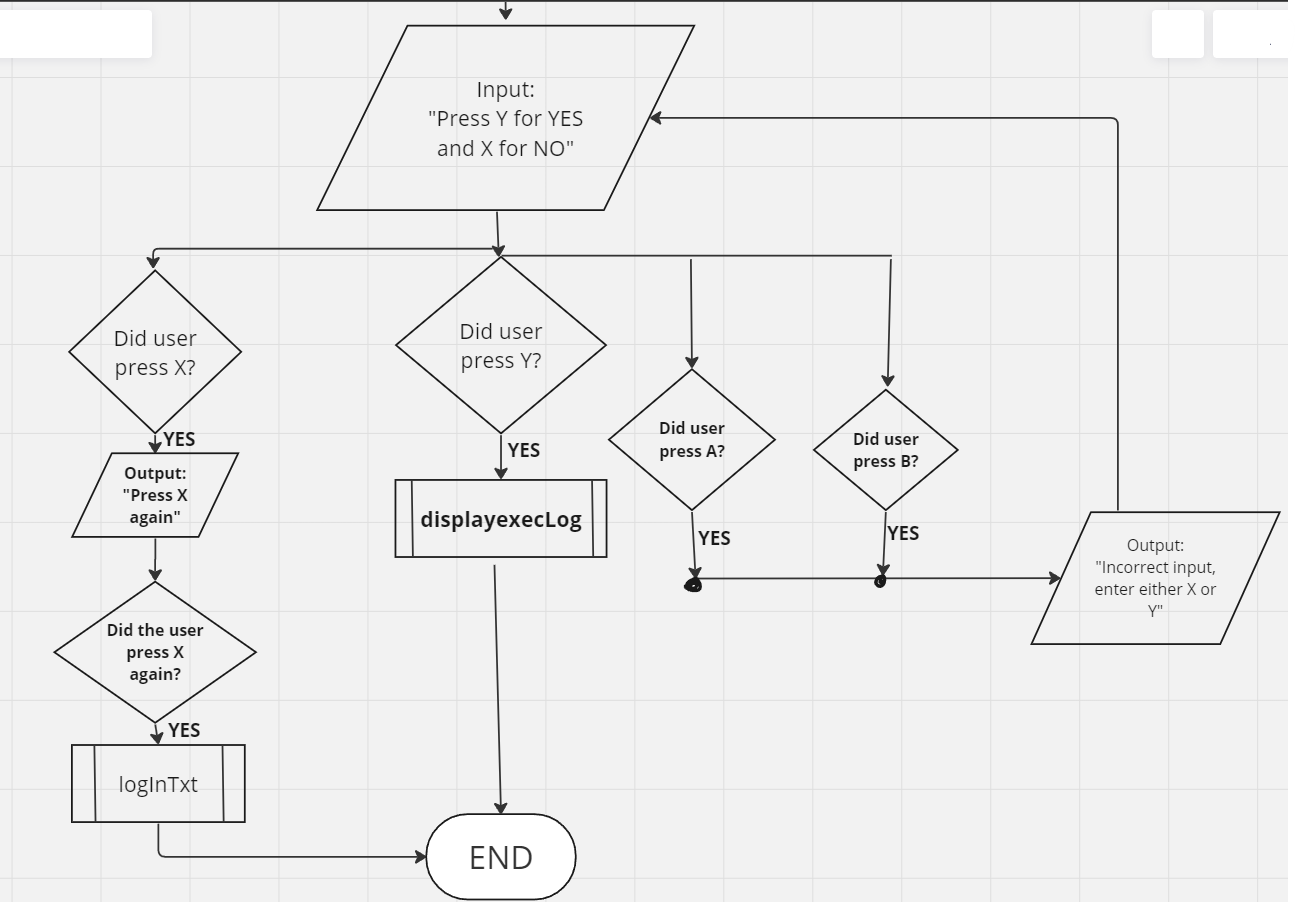
# Algorithm Design



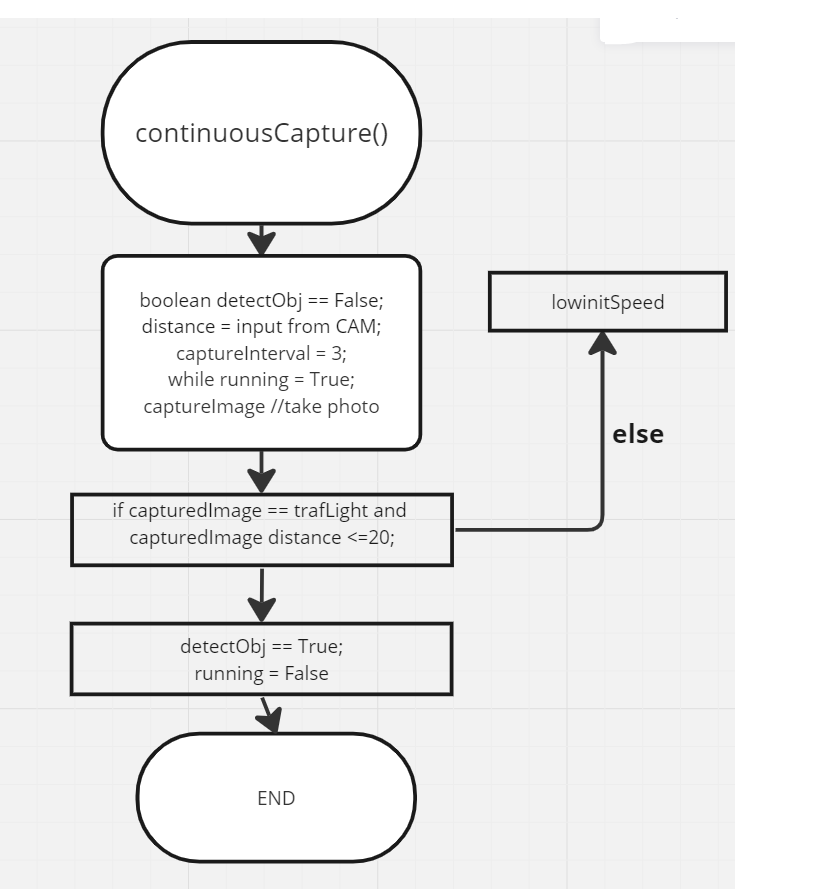


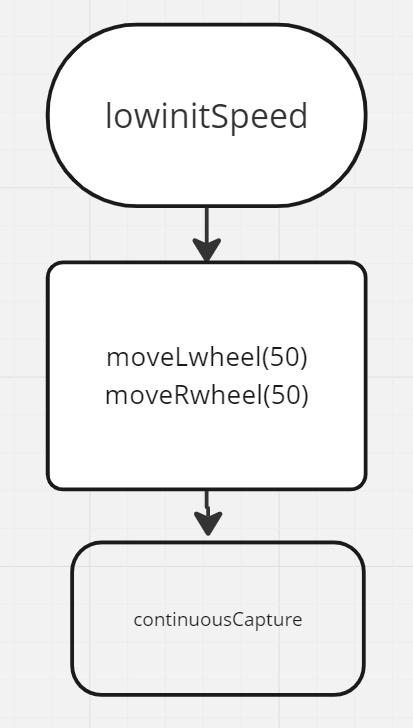


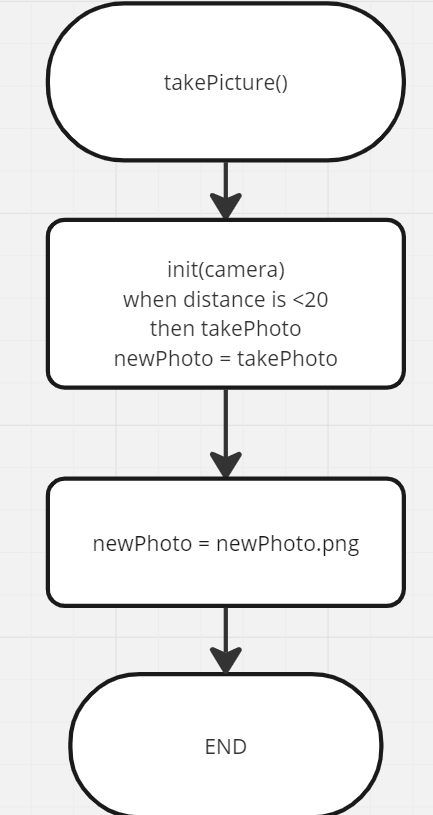


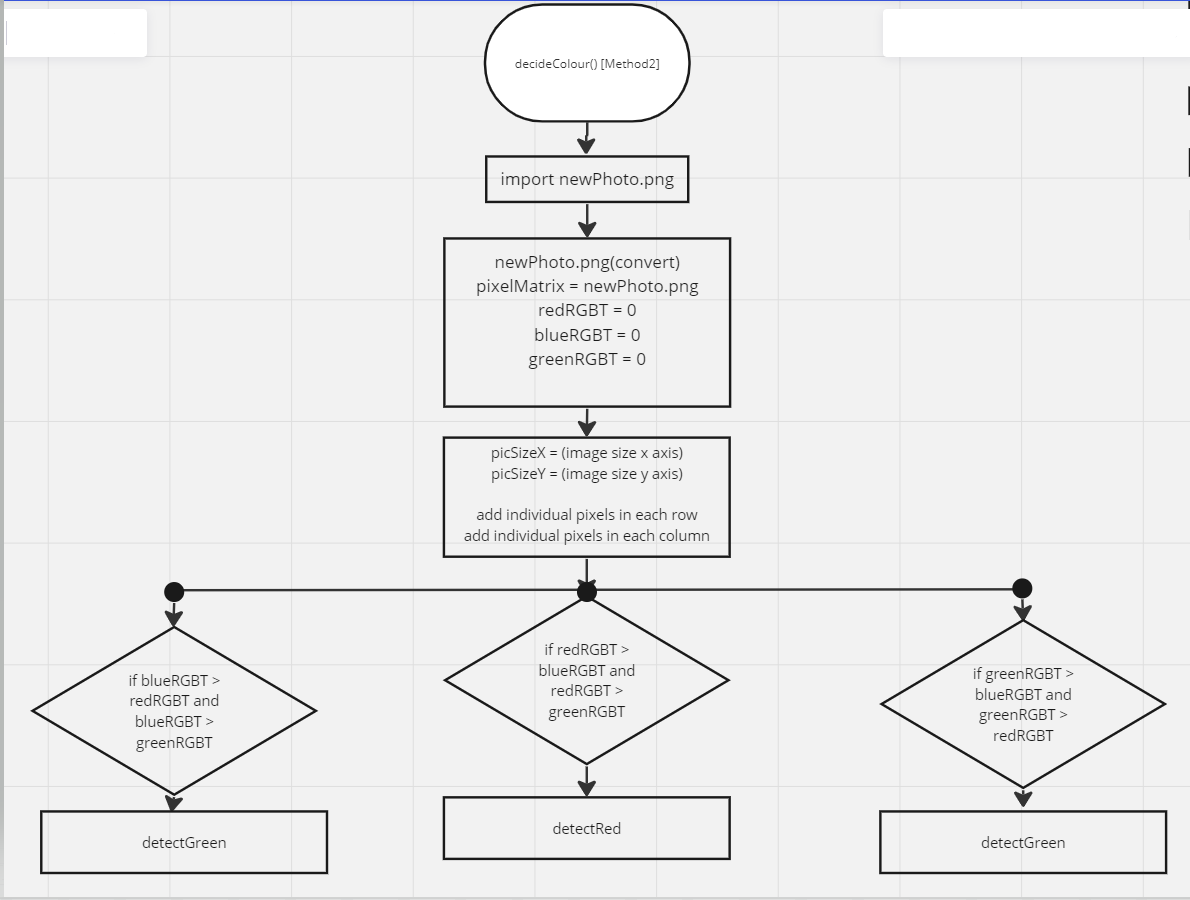


## Sub Procedures:

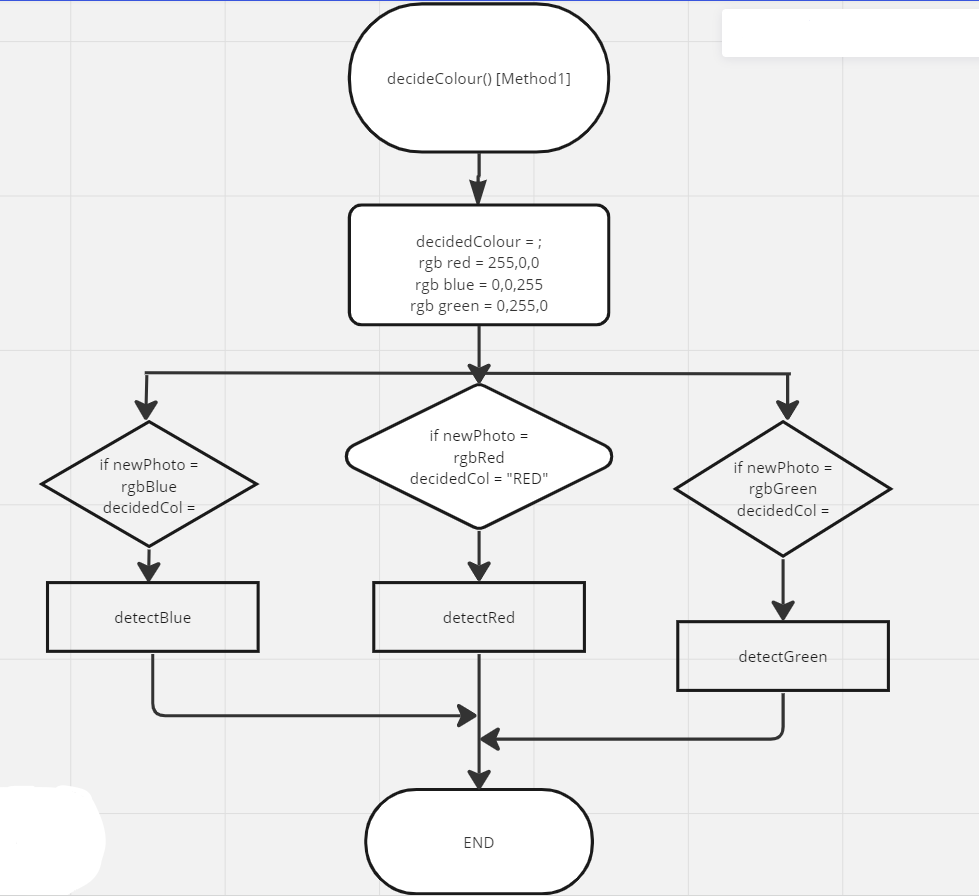






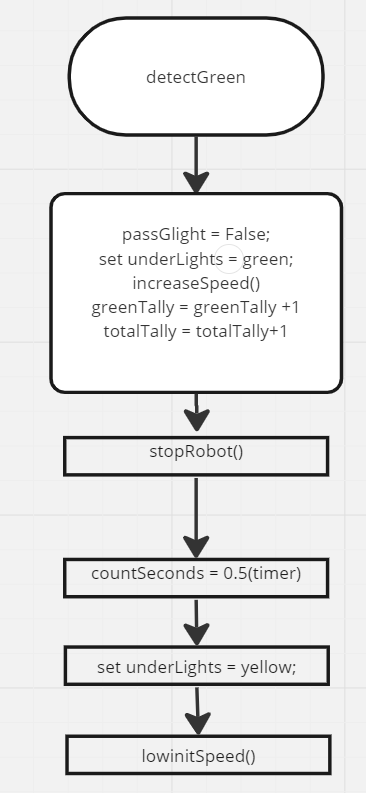


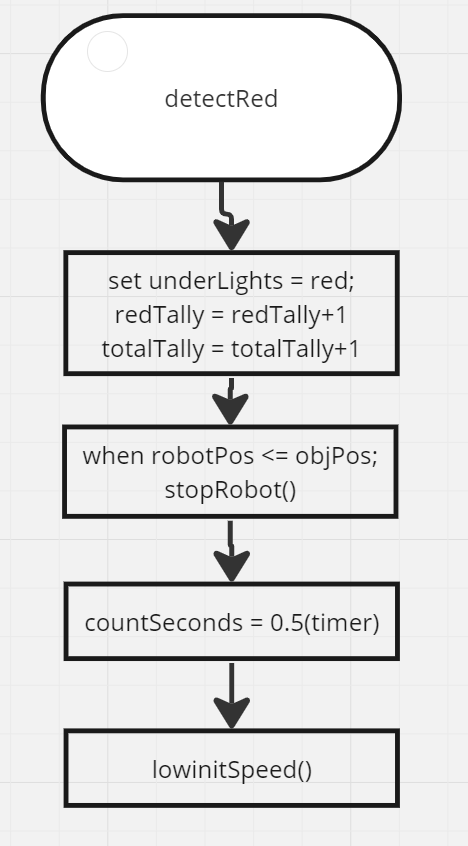
* This method is useful in unideal circumstances where picture quality may be grainy, and the traffic lights visibility is not perfect. It works by gathering image resolution and using the (.convert) command to create a pixel matrix. The pixels on the X and Y are tallied up and added to their individual tallies. The largest of the 3 is calculated and branches into respective sub processes.

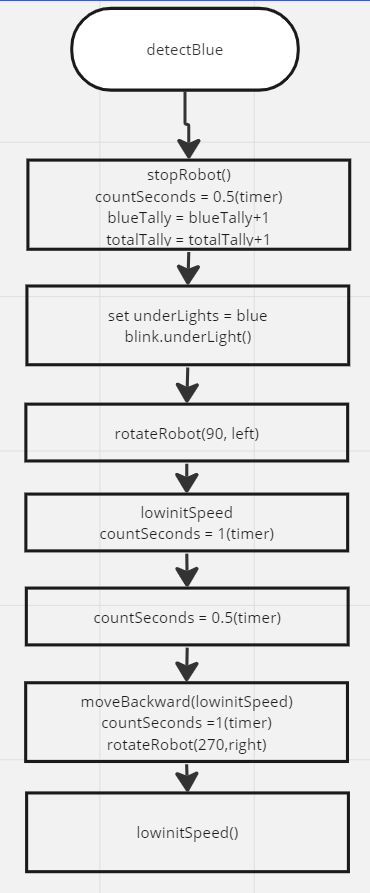


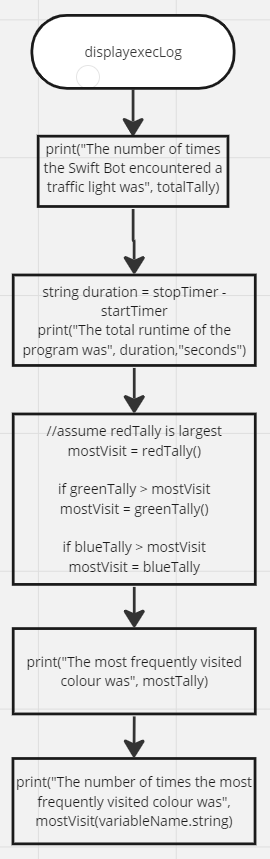
* This method of deciding colours works in an ideal scenario, where lighting is optimal and colour can be recognized and confirmed by simply matching it to its RGB code values. This branches into more sub procedures depending on respective decision by the program.

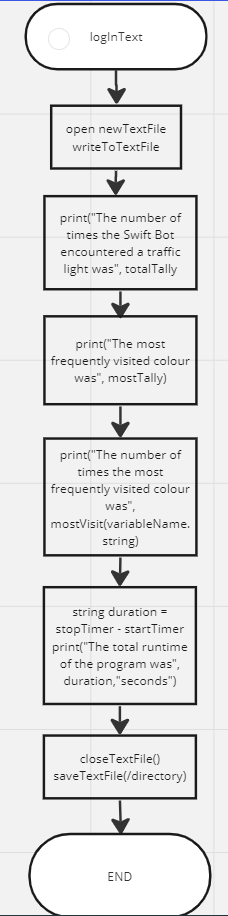
//





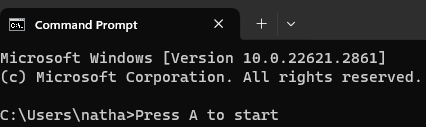






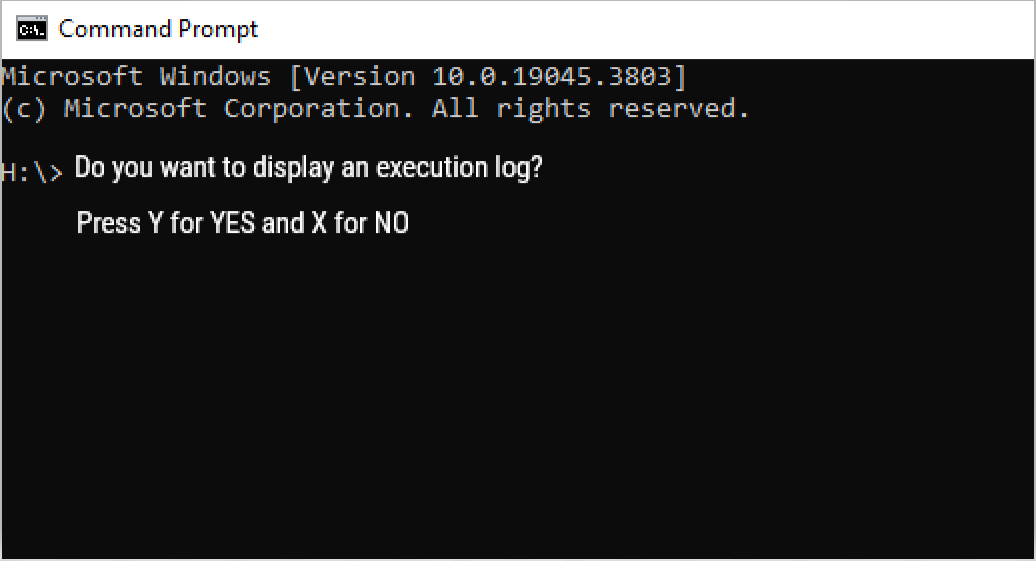
# User Interface Design

1. Initialization and introduction message of the program:



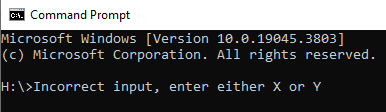
* The user is prompted to Press A on the Swift Bot to start the program process. The next steps are not displayed on the UI, they are internal processes that change the under light LEDs to yellow, set up several tallies as well as begin a runtime counter.

1. Asking the user whether they want to display a log of program execution:



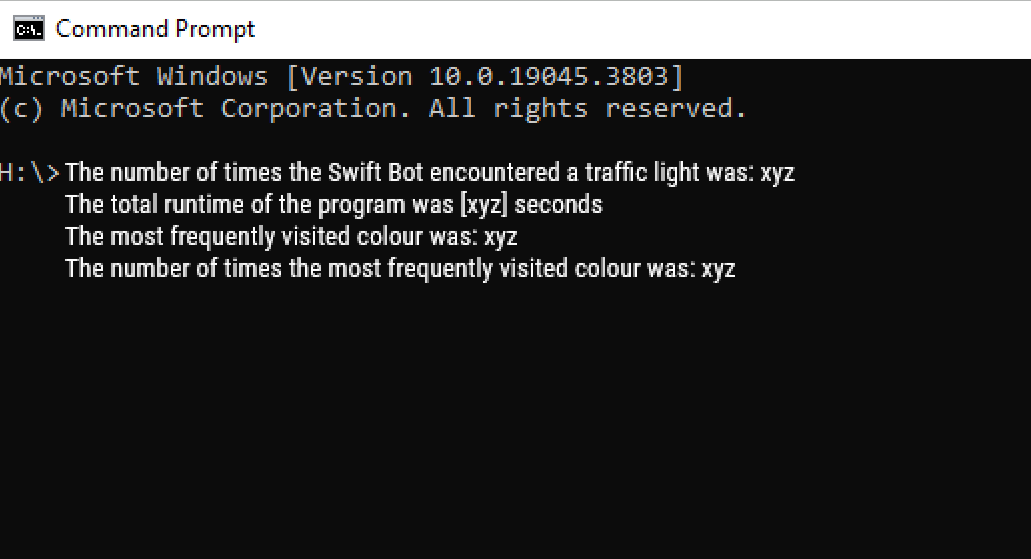
* The user is directed to press either Y to display the execution log or X again to exit the program and write the execution log to a text file.

1. Error handling for UI#2 (above):



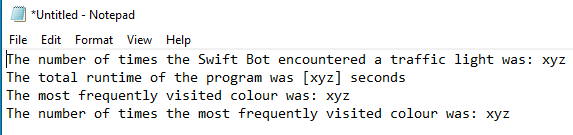
* Given the nature of the Swift Bot’s structure its components including physical buttons are inherently small. It is a plausible scenario in which a given user may press the wrong input buttons (either A or B). To avoid systems errors, the system must be able to recognize this error and go back to the prompt to enter either X or Y.

1. If user wants the execution log displayed:



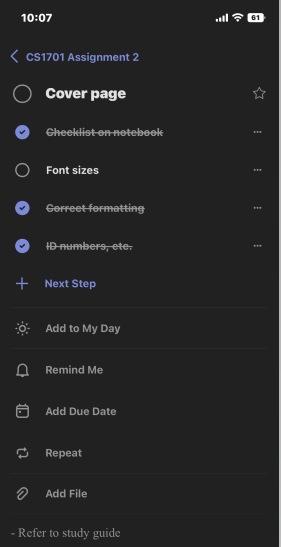
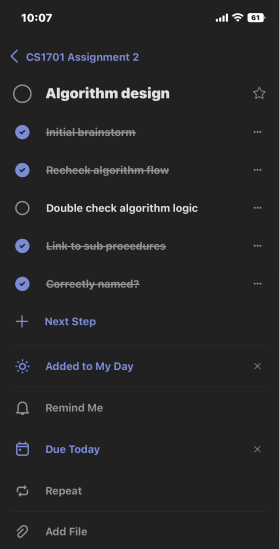
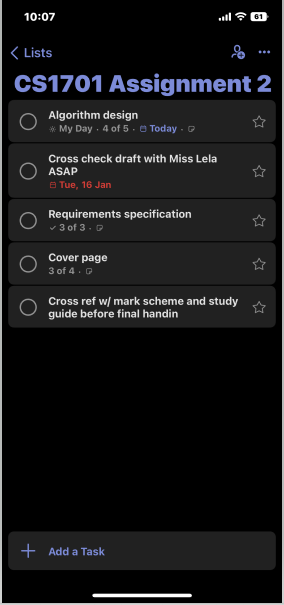
* The displayexecLog() sub process.

1. If the user clicks X again (UI#2):



* The execution log is written and saved to an external text file.

# Planning and Monitoring



* I used Microsoft To Do to organise a work involving the construction of robots and increase efficiency. Thanks to the easy-to-use interface of the application, I was able to quickly and easily generate a work list for the robot-building project. I made use of the ability to define internal deadlines for every activity, which helped me create a well-organized schedule for finishing the project. The reminder feature was quite helpful in alerting me to approaching deadlines and in motivating me to take timely action. Using subtasks within each primary task made it easier to break down complex components of the project into manageable segments, enabling a more organised and detailed approach to the construction process. Microsoft To Do was really helpful in keeping me focused and organised so that I could effectively complete each project's objectives.