Contents

Software Design

Requirements Specification

Introduction:

The purpose of this program is to navigate the finch robot using a computer. The program provides a simple and easy to use graphical user interface with a variety of commands to direct the finch.

Functional Requirements:

1) *The system should have an introductory message at the start, which informs the user of the implemented commands that the program can execute and how to properly enter a command.*
2) *The system shall successfully implement these commands:*
    - *F (for Forward movement)*
    - *B (for Backward movement)*
    - *R (for Right turn)*
    - *L (for Left turn)*
    - *T (for retracing previous movement(s))*
    - *W (for Writing the log of the commands to a text file)*
    - *X (for executing commands from a text file)*
    - *Q (to Quit the program)*
3) *The system shall present the user with an error if any other letter that does not correspond to a command is entered and ask them to re-enter their command.*
4) *No movement should last more than 6 seconds, the speed should not exceed 200 neither should accept a negative value for speed and duration.*
5) *The system shall ensure that the input values are within the set valid range; for example, duration should be entered having a value between 1 to 6 seconds and speed should be entered having a value between 0 to 200, thus an input of 10 seconds or 250 (for speed) will result in an error message requesting the user to re-enter their command.*
6) *The system shall only accept two integer values as inputs for the commands forward('F') and backward("B"). The first value should set the duration of the command (in seconds) and the second value should set the speed of the finch; at which it should execute the command. For example, 'F 3 150' should be interpreted as a command to move the finch forward for 3 seconds at a speed of 150.*
7) *The system shall only accept two integer values for the commands right('R') and left('L'). The first value sets the duration of the turn in seconds and the second value should set the speed of the finch after the turn is made.*
    a) *Each turn should not last more than 6 seconds and speed should not exceed 200.*
    b) *The turns should always be orthogonal to the current course and finch shall continue moving forward at the given duration and speed after the turn is made.*
8) *The system shall only accept one integer value for the command retrace('T'). This value should determine how many previous movements the finch needs to retrace. For example,*

if the finch has executed the following commands 'L', 'R', B', 'F', 'B', 'R', thus when 'T4' is entered, the finch should execute again 'R', then 'B' then 'F' and lastly 'B' (i.e. the previous 4 movements starting from the last one).

   a) *If the number of movements to retrace exceeds the number of movements the finch has executed then the system must display an error message and request the user to re-enter their command.*
   b) *The system shall retrace only previous movement commands and not any previous 'T' commands.*

9) *For the command 'W', the program should write the current local time in HH: MM: SS format and all the commands (including the 'T' commands) that the finch has executed to a text file.*
   a) *If no text file is found then a new text file shall be created and written to.*
10) *For command 'X', the program should open and read an existing file that must contain at least three commands (i.e. 'B 1 100', 'F 3 200', 'L 4 150', 'R 6 170') and have the finch execute them one by one.*
    a) *Either if no text file is found or the text consists of fewer than three commands then, the system shall display an error message informing the user should check a file exist in the specified directory and that the file consists of at least 3 executable commands.*
    b) *The system shall also execute any 'T' commands found in the text file.*
11) *The system shall terminate the program if the command 'Q' is entered.*

Non-functional Requirements:

1) The system should provide a simple and easy to use graphical user interface for the user to interact (GUI) with the program (usability).

2) The program should be reliable and maintainable. Furthermore, it should aim to have low coupling and high cohesion between methods.

3) The system should attempt to implement object-oriented (oo) methods such as encapsulation to ensure that the program is as secure as possible.

4) The system may have an additional log display in the graphical user interface (GUI), where it displays previously executed commands and error messages that do occur during the execution of commands.

Additional Requirements:

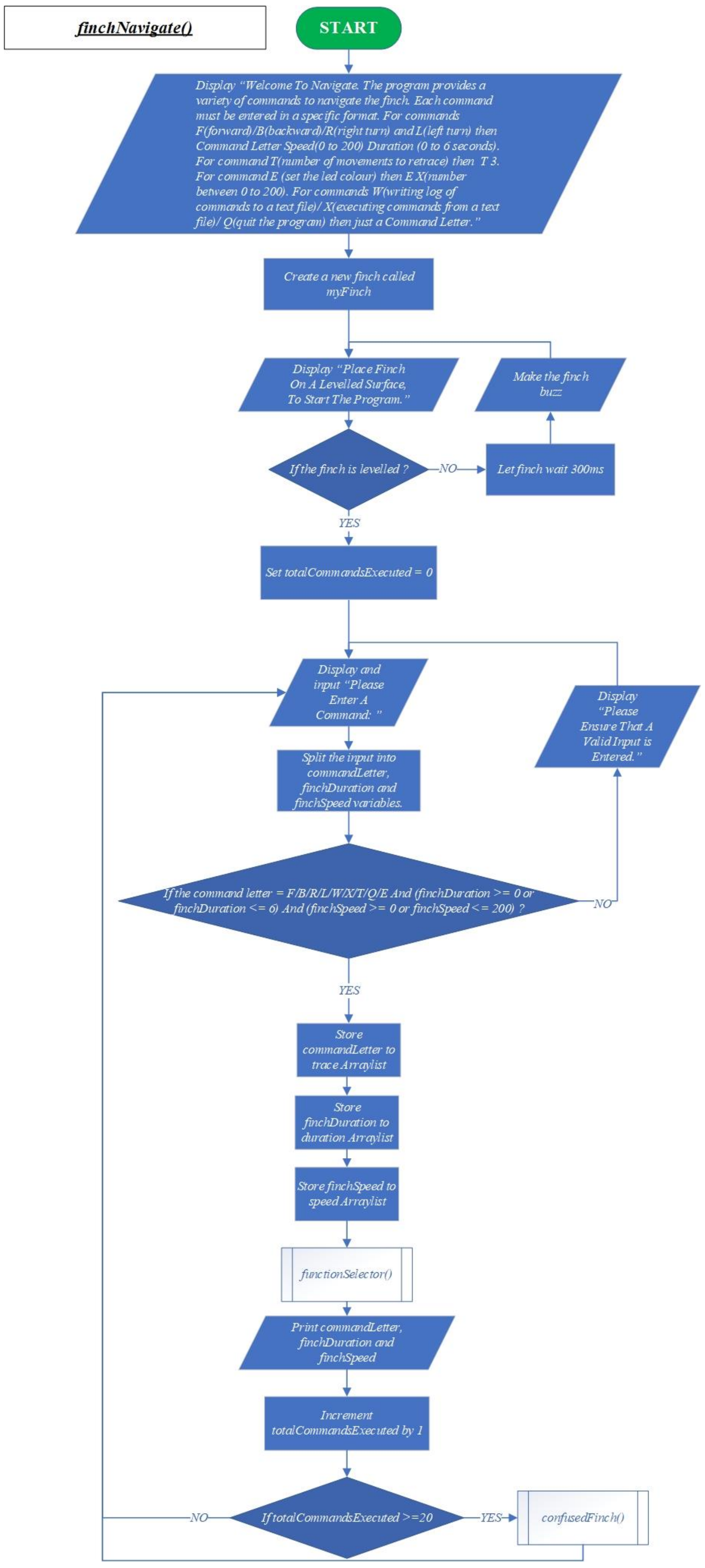
1) The system may have an additional mode ("Confused Finch") where if twenty or more commands are executed then it results in the finch randomly executing 5 previously executed commands from the trace array. Furthermore, the finch may choose to forget any previous commands executed (each ArrayList is cleared) if the confused finch mode occurs.

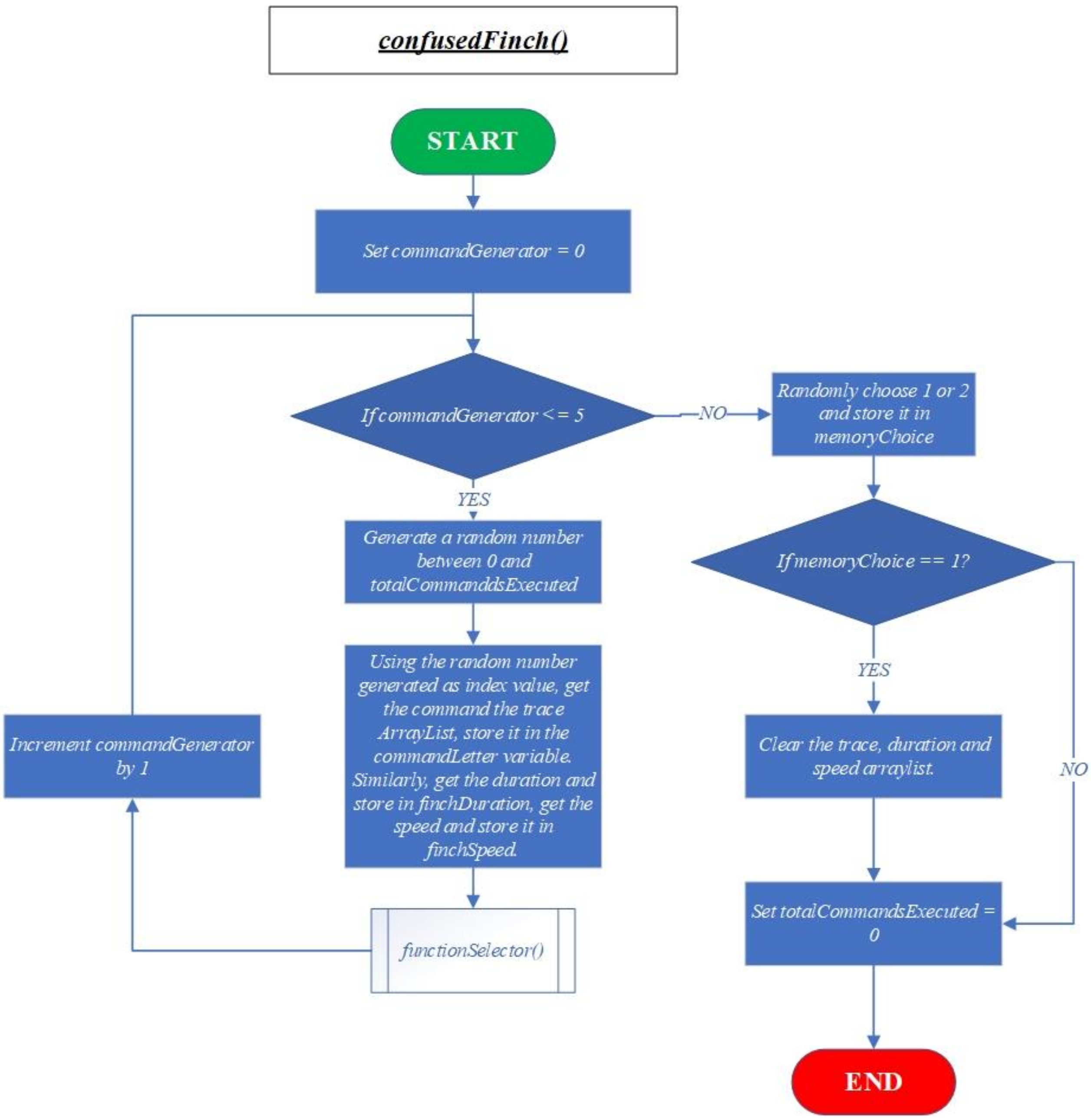
2) The system may have an additional function that checks the level of the finch relative to the ground. If the finch is on a levelled ground, then the program will continue else, the system shall display an error message and request the user to place the finch on a levelled surface. Furthermore, the system shall make the finch wait 300ms and make a buzz sound.

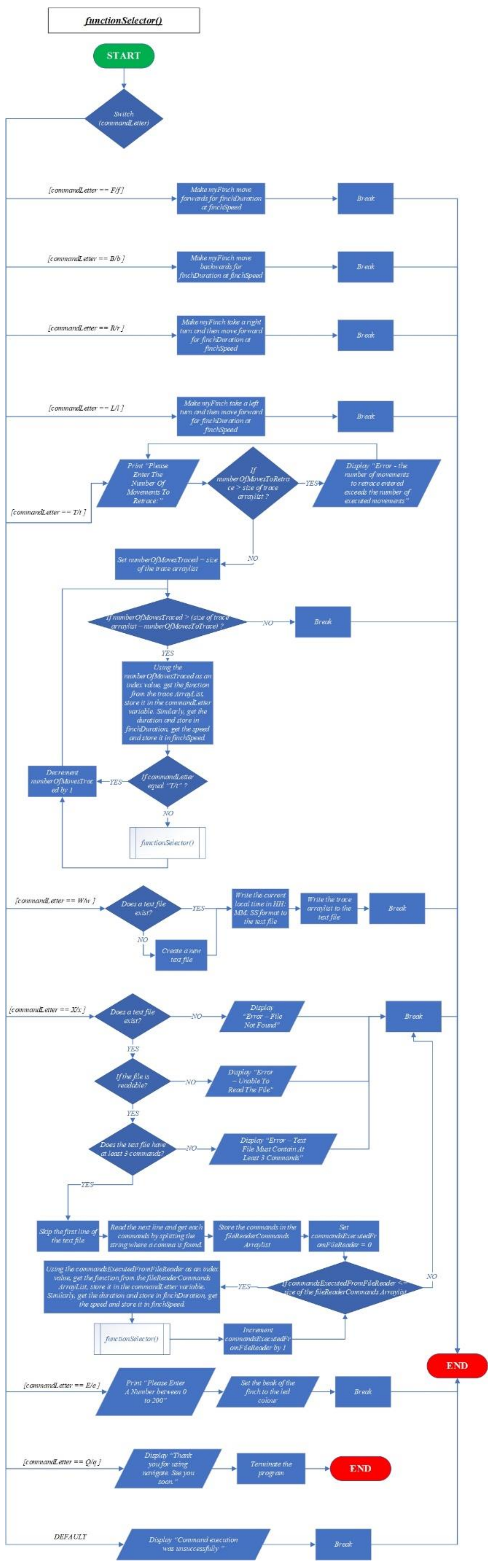
3) An additional command, 'E (for Led colour)' may be implemented, where the system shall only accept one integer value between 0 and 200. This value sets the beak of the finch to a random colour for 3 seconds. If a value greater less than 0 or than 200 is entered, then the user must be informed using an error message and request the user to re-enter their command.

Conclusion: The design of the algorithm includes all the required functionalities, the non-functional requirements and the additional requirements. Many of the additional functionalities are added, provide a variety and commands for the user to execute and also the confusedFinch mode, makes the program more interesting and complete.

Flowchart:


finchNavigate()
START
Display "Welcome To Navigate. The program provides a variety of commands to navigate the finch. Each command must be entered in a specific format. For commands F(forward)/B(backward)/R(right turn) and L(left turn) then Command Letter Speed(0 to 200) Duration (0 to 6 seconds). For command T(number of movements to retrace) then T 3. For command E (set the led colour) then E X(number between 0 to 200). For commands W(writing log of commands to a text file)/ X(executing commands from a text file)/ Q(quit the program) then just a Command Letter."
Create a new finch called myFinch
Display "Place Finch On A Levelled Surface, To Start The Program."
Make the finch buzz
If the finch is levelled ?
NO
Let finch wait 300ms
YES
Set totalCommandsExecuted = 0
Display and input "Please Enter A Command: "
Display "Please Ensure That A Valid Input is Entered."
Split the input into commandLetter, finchDuration and finchSpeed variables.
If the command letter = F/B/R/L/W/X/T/Q/E And (finchDuration >= 0 or finchDuration <= 6) And (finchSpeed >= 0 or finchSpeed <= 200) ?
NO
YES
Store commandLetter to trace Arraylist
Store finchDuration to duration Arraylist
Store finchSpeed to speed Arraylist
functionSelector()
Print commandLetter, finchDuration and finchSpeed
Increment totalCommandsExecuted by 1
NO
If totalCommandsExecuted >=20
YES
confusedFinch()


confusedFinch()
START
Set commandGenerator = 0
If commandGenerator <= 5
NO
Randomly choose 1 or 2 and store it in memoryChoice
YES
Generate a random number between 0 and totalCommanddsExecuted
Using the random number generated as index value, get the command the trace ArrayList, store it in the commandLetter variable. Similarly, get the duration and store in finchDuration, get the speed and store it in finchSpeed.
functionSelector()
Increment commandGenerator by 1
If memoryChoice == 1?
YES
NO
Clear the trace, duration and speed arraylist.
Set totalCommandsExecuted = 0
END



functionSelector()
START
Switch (commandLetter)
[commandLetter == F/f]
Make myFinch move forwards for finchDuration at finchSpeed
Break
[commandLetter == B/b]
Make myFinch move backwards for finchDuration at finchSpeed
Break
[commandLetter == R/r]
Make myFinch take a right turn and then move forward for finchDuration at finchSpeed
Break
[commandLetter == L/l]
Make myFinch take a left turn and then move forward for finchDuration at finchSpeed
Break
[commandLetter == T/t]
Print "Please Enter The Number Of Movements To Retrace:"
If numberOfMovesToRetrace > size of trace arraylist ?
YES
Display "Error - the number of movements to retrace entered exceeds the number of executed movements"
NO
Set numberOfMovesTraced = size of the trace arraylist
If numberOfMovesTraced > (size of trace arraylist – numberOfMovesToTrace) ?
NO
Break
YES
Using the numberOfMovesTraced as an index value, get the function from the trace ArrayList, store it in the commandLetter variable. Similarly, get the duration and store in finchDuration, get the speed and store it in finchSpeed
If commandLetter equal "T/t" ?
YES
Decrement numberOfMovesTraced by 1
NO
functionSelector()
[commandLetter == W/w]
Does a text file exist?
YES
NO
Create a new text file
Write the current local time in HH: MM: SS format to the text file
Write the trace arraylist to the text file
Break
[commandLetter == X/x]
Does a text file exist?
NO
Display "Error – File Not Found"
Break
YES
If the file is readable?
NO
Display "Error – Unable To Read The File"
YES
Does the text file have at least 3 commands?
NO
Display "Error – Text File Must Contain At Least 3 Commands"
YES
Skip the first line of the text file
Read the next line and get each commands by splitting the string where a comma is found.
Store the commands in the fileReaderCommands Arraylist
Set commandsExecutedFromFileReader = 0
If commandsExecutedFromFileReader < size of the fileReaderCommands Arraylist
NO
YES
Using the commandsExecutedFromFileReader as an index value, get the function from the fileReaderCommands ArrayList, store it in the commandLetter variable. Similarly, get the duration and store in finchDuration, get the speed and store it in finchSpeed.
functionSelector()
Increment commandsExecutedFromFileReader by 1
END
[commandLetter == E/e]
Print "Please Enter A Number between 0 to 200"
Set the beak of the finch to the led colour
Break
[commandLetter == Q/q]
Display "Thank you for using navigate. See you soon."
Terminate the program
END
DEFAULT
Display "Command execution was unsuccessfully "
Break

Pseudocode

Algorithm 1: finchNavigate (myFinch, User Input)

/ This algorithm takes in commands from the graphical user interface (GUI), validates if the command (function, duration and speed) entered are within the set valid range and finally stores the executed commands to an ArrayLists. */*

Input: User inputs are string based which includes the function, duration and the speed. myFinch is a finch. Finch also provides an internal input to check if its levelled – where it returns a Boolean value.

```
Set finchDuration = 0, finchSpeed = 0, totalCommandsExecuted = 0
Initialise inputCommand, commandLetter
Create an ArrayList called trace
Create an ArrayList called duration
Create an ArrayList called speed

Print "Welcome To Navigate. The program provides a variety of commands to navigate the finch. Each command must be entered in a specific format. For commands F(forward)/B(backward)/R(right turn) and L(left turn) then Command Letter Speed(0 to 200) Duration (0 to 6 seconds)…"

/* Prints out an introductory message informing the user about the program. Creates a new finch and it make finch wait for 300ms, make the finch buzzes to inform the user to place finch on levelled surface. */

Create a new finch
Repeat
    Let myFinch wait for 300ms
    myFinch buzz
    Print "Place Finch On A Levelled Surface, To Start The Program."
Until myFinch is on a levelled surface
```

/ This while loop ensure repeats itself until the user enters a valid command which satisfies all conditions in the if statement. Furthermore, it stores function into the trace ArrayList, finchDuration in the duration ArrayLists and finchSpeed into the speed ArrayList. Then the functionSelector method is called. If the totalCommandsExecuted is greater than 20 then it calls the confusedFinch method. */*

```
While true
    Set inputCommand = Input("Please Enter A Command:"")
    Split the inputCommand into commandLetter, finchDuration and finchSpeed
    If the commandLetter is equal to (F or B or R or L or W or X or T or Q or E) And
    (finchDuration >= 0 or finchDuration <= 6) And (finchSpeed >= 0 or finchSpeed <=
    200) then
        Add commandLetter to the trace ArrayList
        Add finchDuration to the duration ArrayList
        Add finchSpeed to the speed ArrayList

        Call functionSelector()
        Print "commandLetter, finchDuration and finchSpeed"
        Increment totalCommandsExecuted by 1
        If totalCommandsExecuted  >= 20 then
            Call confusedFinch()
        Break
        End If
    Else
        Print "Please Ensure That A Valid Input is Entered."
    End If
End While
```

Output: Buzz sound from the finch, totalCommandsExecuted, commandLetter, finchDuration, finchSpeed, trace, duration and speed.

Algorithm 2: confusedFinch (myFinch, totalCommandsExecuted, commandLetter, finchDuration, finchSpeed, trace, duration and speed)

/ This algorithm makes the finch randomly select 5 previously executed commands and adds a random option which can clear any previously stored commands. */*

Input: totalCommandsExecuted is an integer, which indicates the total number of commands so far. commandLetter is a character where it stores the function. finchDuration and finchSpeed are integer values. Trace is a character ArrayList consisting commandLetters. Both duration and speed are integer based ArrayList.

/ When curiousFinch method is called, it randomly selects 5 previously executed commands from the 3 ArrayLists and executes using the functionSelector method. Lastly, towards the end it randomly picks a number, which decides if the previous command log remain or not. If 1 is chosen then all 3 ArrayLists are cleared. */*

```
Set commandGenerator = 0
Initialise indexValue, memoryChoice
If commandGenerator <= 5 then
    indexValue = Generate a random number between 0 and totalCommandsExecuted
    Set commandLetter = Get trace[indexValue]
    Set finchDuration = Get duration[indexValue]
    Set finchSpeed = Get speed[indexValue]
    Call functionSelector()
    Increment commandGenerator by 1
Else
    Set memoryChoice = generate a random number (1, 2)
    If memoryChoice is equal to 1 then
        Clear trace ArrayList
        Clear duration ArrayList
        Clear speed ArrayList
        Set totalCommandsExecuted = 0
    Else
```

```
        Set totalCommandsExecuted = 0
    End If
End If
```

Output: totalCommandsExecuted, commandLetter, finchDuration, finchSpeed, trace, duration and speed

Algorithm 3: functionSelector (myFinch, text file, commandLetter, finchDuration, finchSpeed, trace, duration and speed)

/ This algorithm is a switch case, which takes in the commandLetter as input and compared it against each individual case. */*

Input: The text file consists of string commands separated by commas. commandLetter is a type of string variable which stores the function. finchDuration and finchSpeed are integer values. Trace, duration and speed are types of arrays. myFinch is a finch.

```
Initialise numberOfMovesToRetrace, commandsExecutedFromFileReader
Create a new ArrayList called fileReaderCommands
Switch, based on commandLetter
```

/ The following cases; f/b/r/l, use the commandLetter to select the appropriate case. Each case then uses the finchDuration and finchSpeed values to make the finch move. */*

```
Case commandLetter equals F/f
    Move myFinch forward for finchDuration at finchSpeed
    Break
End Case
Case commandLetter equals B/b
    Move myFinch backward for finchDuration at finchSpeed
    Break
End Case
Case commandLetter equals R/r
    Move myFinch to take a right turn
```

```
          **Move** myFinch forward for finchDuration at finchSpeed
          **Break**
End Case
Case commandLetter equals L/l
          **Move** myFinch to take a left turn
          **Move** myFinch forward for finchDuration at finchSpeed
          **Break**
End Case
```

/ This case retrieves the previously executed commands and using a for and if loops, it reads the array backwards and if T is found as commandLetter than the command is skipped. Each command is then executed one by one by calling the functionSelector method. */*

```
Case commandLetter equals T/t
          While True
                    Set numberOfMovesToRetrace = Input("Please Enter The Number Of
                    Movements To Retrace:"")
                    If numberOfMovesToRetrace > size of trace ArrayList then
                              Print "Error - the number of movements to retrace entered exceeds the
                              number of executed movements"
                    Else
                              Set numberOfMovesTraced = size of the trace arraylist
                              OUTER_LOOP
                              If numberOfMovesTraced > (size of trace arraylist –
nunberOfMovesToTrace) then
                                        Set commandLetter = Get trace[numberOfMovesTrraced]
                                        Set finchDuration = Get duration[numberOfMovesTraced]
                                        Set finchSpeed = Get speed[numberOfMovesTraced]
                                        If commandLetter is equal to T or t then
                                                  Decrement numberOfMovesTraced by 1
                                                  Break OUTER_LOOP
                                        Else
                    Call functionSelector()
                    Decrement numberOfMovesTraced by 1
                    Break OUTER_LOOP
                End If
            Else
                Break
            End If
        End If
    End While
End Case

/* This case writes down the local time on a text along with all the previously executed
commands (including the 'T' commands) to a text file. If no text file is found the a new text
file is created. */

Case commandLetter equals W/w
    While True
        If a text file exists then
            Write the current local time in HH: MM: SS format to the text file
            Write the trace ArrayList to the text file
            Break
        Else
            Create a new text file
            Write the current local time in HH: MM: SS format to the text file
            Write the trace ArrayList to the text file
            Break
        End If
    End While
End Case
```

/ This case makes few checks such as if a text file exists, if the text file is readable and if the text file has at least 3 commands. Then the commands are read and broken down when a comma is found. Each command is stored in a fileReaderCommands ArrayList and then executed one by one using the functionSelection(). If any of the conditions are not meet, then an error message is displayed. */*

```
Case commandLetter equals X/x
    If a text file exists then
        If the text file is readable then
            If he texts file has at least 3 commands then
                Skip the first line of the text file
                Read the next line and split when a comma is found
                Store the commands in the fileReaderCommands ArrayList
                Set commandsExecutedFromFileReader = 0
                Second_Outer_Loop
                If commandsExecutedFromFileReader <= size of the
                fileReaderCommands Arraylist then
                    Set commandLetter = Get
fileReaderCommands[commandsExecutedFromFileReader]
                    Set finchDuration = Get
duration[commandsExecutedFromFileReader]
                    Set finchSpeed = Get speed[num
commandsExecutedFromFileReader berOfMovesTraced]
                    Call functionSelector()
                    Increment commandsExecutedFromFileReader by 1
                    Break Second_Outer_Loop
                Else
                    Break
                End If
            Else
                Print “Error – Text File Must Contain At Least 3 Commands”
                Break
            End If
        Else
            Print "Error – Unable To Read The File"
            Break
        End If
    Else
        Print "Display "Error – File Not Found"
        Break
    End If
End Case
```

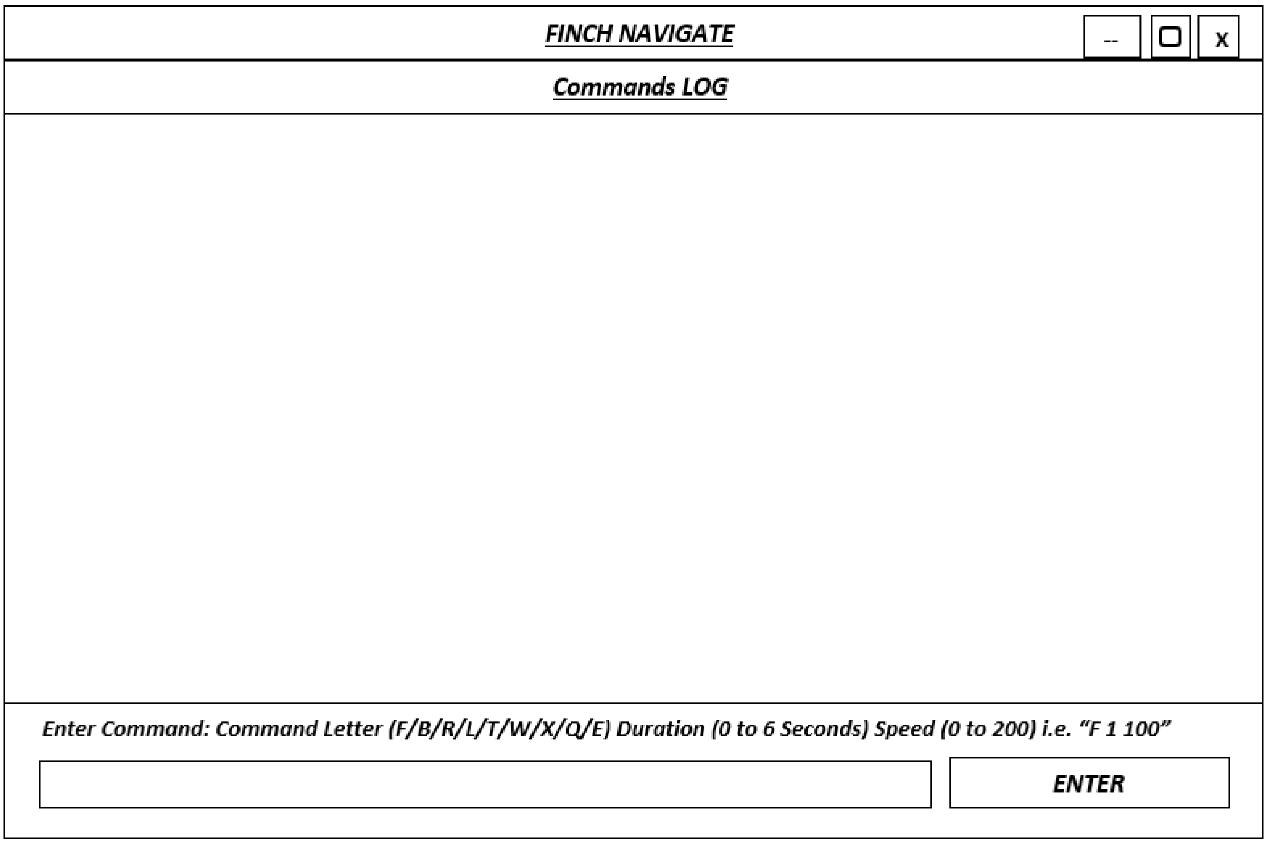
/ The E/e case take in an integer input and set the beak of the finch to its led colour. Case Q causes the program to terminate and lastly, the default case prints out a print message and breaks out of the switch statement. */*

```
Case commandLetter equals E/e
    Print "Please Enter A Number between 0 to 200"
    Set the beak of the finch to the led colour using the input from the user
    Break
End Case
Case commandLetter equals Q/q
    Print "Thank you for using navigate. See you soon."
    Terminate the program
    Exit
End Case
Case DEFAULT
    Print "Thank you for using navigate. See you soon."
    Break
End Case
```

Outputs: *Text file with commands written to it, trace, duration, speed, commandLetter, finchDuration, finchSpeed, Led Colour shown on the beak of the finch and termination of the program.*

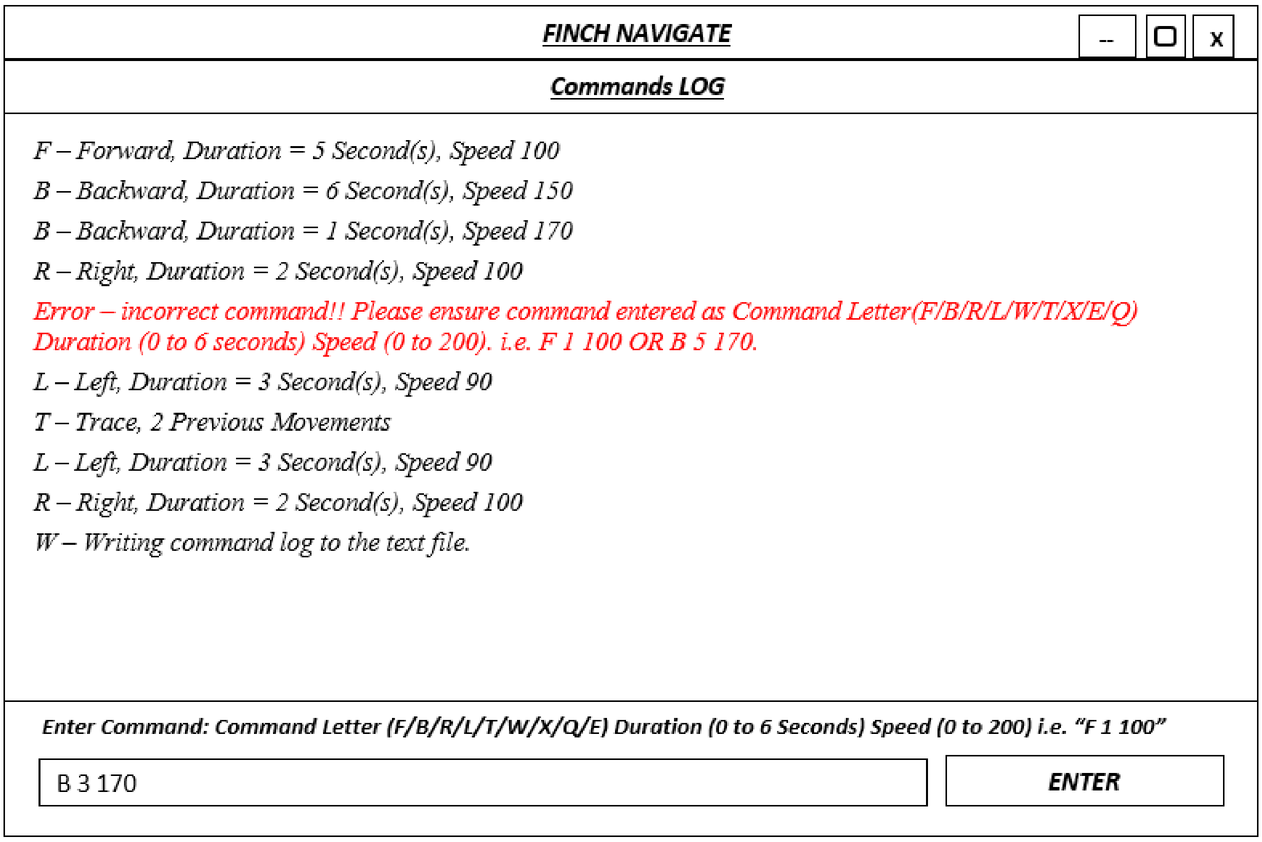
Graphical User Interface (GUI) – Prototype Design

Starting Screen:



Enter Command: Command Letter (F/B/R/L/T/W/X/Q/E) Duration (0 to 6 Seconds) Speed (0 to 200) i.e. "F 1 100"

Program in use:



F – Forward, Duration = 5 Second(s), Speed 100
B – Backward, Duration = 6 Second(s), Speed 150
B – Backward, Duration = 1 Second(s), Speed 170
R – Right, Duration = 2 Second(s), Speed 100
Error – incorrect command!! Please ensure command entered as Command Letter(F/B/R/L/W/T/X/E/Q) Duration (0 to 6 seconds) Speed (0 to 200). i.e. F 1 100 OR B 5 170.
L – Left, Duration = 3 Second(s), Speed 90
T – Trace, 2 Previous Movements
L – Left, Duration = 3 Second(s), Speed 90
R – Right, Duration = 2 Second(s), Speed 100
W – Writing command log to the text file.

Enter Command: Command Letter (F/B/R/L/T/W/X/Q/E) Duration (0 to 6 Seconds) Speed (0 to 200) i.e. "F 1 100"

B 3 170