Introduction

This task is mainly meant to strengthen our skills in algorithm design and build on our programming abilities. The primary goal is to create a well-structured flowchart to help us implement the Simon game on our SwiftBot. The flowchart must be detailed and logical for the implementation to be straightforward.
The game consists of the user being able to push buttons, which will generate random colours. After five rounds, the user can choose between continuing or quitting the game. If the user has five points by the end of the fifth round, the robot will do a celebration dance. In essence, the robot's key functionalities are generating random colours, tracking the user's score, giving the user the option to quit, and mobility.

Mandatory Functionalities

1. **Assigning the buttons and colours**
   Each button (A, B, X, Y) on the SwiftBot corresponds to a colour of the LEDs
   Assign (Red, Green, White, Blue) to the underlight LEDs (FRONT_LEFT, FRONT_RIGHT, MIDDLE_LEFT, MIDDLE_RIGHT, BACK_LEFT, BACK_RIGHT).

2. **Initialising the game**
   The game should start with a score of 0.
   The score of the current round and the current round number should be displayed at all times (Except when the game is over).

3. **Random sequence generation**
   A random sequence of LEDs should be created and the corresponding lights should blink.
   For every round after that, a new random colour should be added to the original sequence and the previous sequence must stay the same and must not change unless the player restarts the game.

4. **User Input**
   Button inputs - A, B, X, Y - The button input must be recognised by the program and it should add a number (based on the button pressed) to the array. After the random colour sequence is shown to the user, the user should be able to press the buttons that are assigned to each colour in the right order and the program should validate if the combination of the buttons pressed match and are in the order they need to be in. If it does not match, the game should end and the user should be given the choice to either restart or to quit. If it matches, the user goes on to the next round. Every 5 rounds, the user is asked whether they want to continue playing the game or not. When this happens the user can input either yes or no in the open command prompt to either go on to the next 5 rounds or quit.
   Starting the game should also require an input and if they get a game over message they will have to choose from either quitting or restarting with another command prompt input.

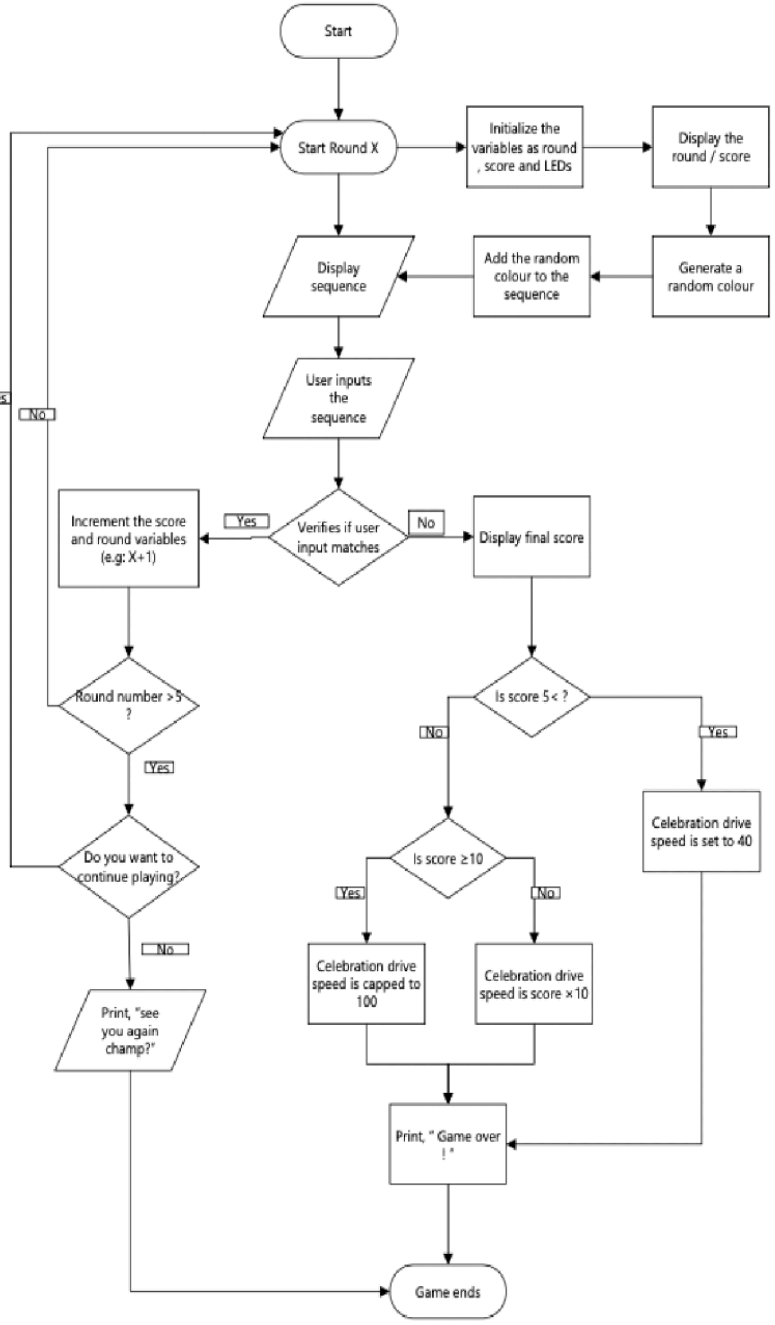
5. **Scores**
   The round score and round number should be displayed after each round.
   The score should increment by 1 after every successful round.

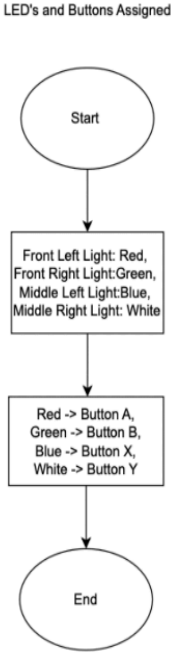
6. **Game Rules**
   The game should terminate if the user inputs the wrong sequence (if the randomly generated and the input arrays do not match.).
   The user should be given the option to quit or continue every 5 rounds.

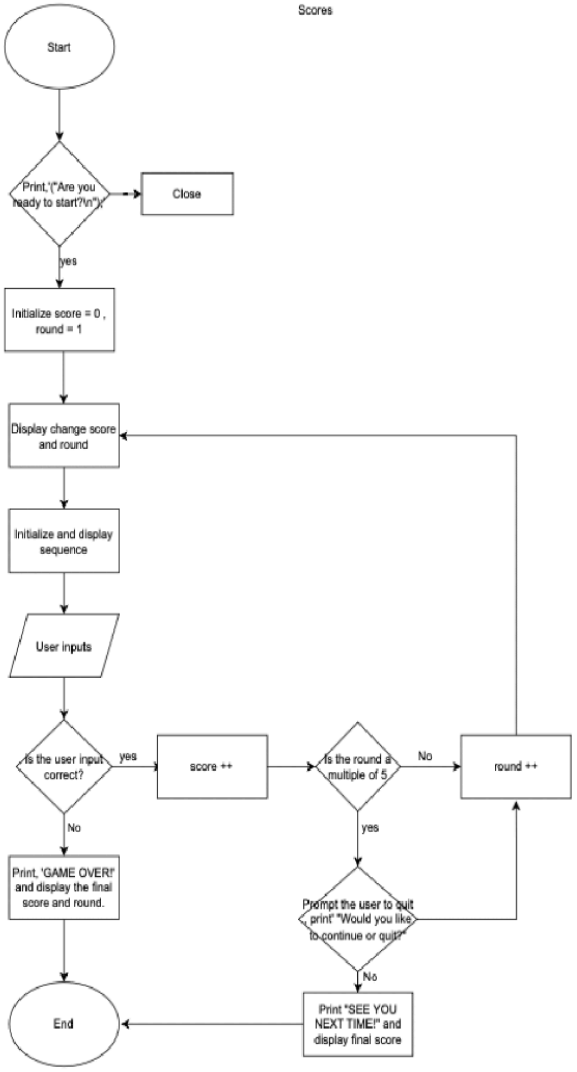
7. **Celebration**
   The celebration should start after the user either fails the game or they decide to quit at any time when they are asked if they want to continue or not.
   The celebration should be a V-shape movement, where the SwiftBot moves 30cms in one direction, then it returns to the starting position, then it moves 30cms again after turning 45 degrees. After this the SwiftBot must move back again, and it should randomly blink before doing the V-shape movement and after it too.
   The speed at which the V-shape movement is done should be based on the score the user gets. The higher the score, the faster the robot should do celebration, but the distance it moves must stay 30cm always. It must always go the same distance so the time it moves must also change as the speed changes (which obviously changes according to the score)
   The score * 10 is the speed at which the SwiftBot will perform the celebration
   The minimum speed of the celebration should be 40, even if the user doesn't get a score of 4. The maximum speed at which the robot will go is 100, so after round 10, if the user continues to keep going, the Swiftbot's celebration will stay at 100 speed.

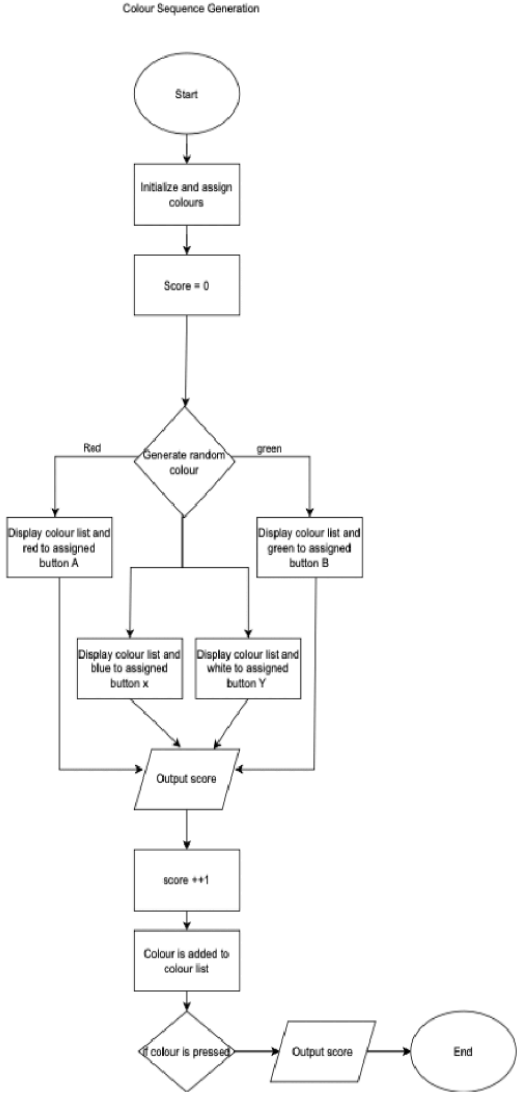
8. **Display messages**
   Display the game score before terminating.
   Display "Do you want to continue?" after every 5 rounds.
   Display the two inputs the user can put for this, yes or no.
   Display "GAME OVER!" if the user fails the round.
   Display the different inputs the user can use to either quit or continue.
   Display "SEE YOU NEXT TIME!" if the user quits.
   Display the speed at which the SwiftBot will do the celebration.

Algorithm Design









This flowchart visually represents the steps and logic of the program that meets the specification. We used standard flowchart symbols to ensure clarity and understanding.

Ovals - were used to indicate the start and end points of the programme.

Rectangles - were used to represent tasks or processes, indicating specific actions to be performed.

Diamonds - were employed to illustrate decision points, where a choice is made that impacts the following flow of the process.

Arrows connect these steps, showing the direction of the programme. These symbols ensure the flowchart is easy to follow and interpret.

Differences between the designed solution and the solution implemented

The requirements almost completely match the implemented solution, as we managed to finish everything on time and we didn't need to leave out anything. We decided to make the user interface visually appealing for the player so they are encouraged to play the game. Which wasn't a requirement, but we decided to make it look slightly better than just having text saying 'start game, type 1'. Some other slight differences were not having yes or no as the available inputs but instead 1 or 2, and we clarified what 1 does and what 2 does in different scenarios.

The first part of the requirements is assigning the buttons to the different colours and also assigning the underlight LEDs to have each one only blink a set colour. We didn't specify which button will correspond to what colour or the underlights in the requirements, however in the implementation we did. We assigned the front left underlight LED to be red, the middle left to be blue, the front right to be green and the middle right to be white. We assigned the button A as red, the button B as green, the button X as blue, and the button Y as white.

The random sequence generation went fine, worked as laid out in design and every new colour was added to the original array. These were then compared with the input array to check whether the two matched or not.

The celebration works as we planned it in the design, we had no problem there, I added a hard cap of score at 30 because, meaning the user can not go above 30 score. I simply did this because the likeliness of someone being able to get a score higher than 30 without writing each colour down on a piece of paper is unlikely. Obviously the whole point of this game is to train memory and writing it down kind of defeats the purpose.

The game rules have not changed in the implemented solution and the display messages remained generally the same as in the designed solution.

The user input worked, the button presses were registered and a number based on the input was added to the array. The command prompt inputs also worked as expected in the designed solution

Errors we have seen in the implementation

We encountered one small problem during testing. The problem was that when two buttons were pressed after one another too quickly, the SwiftBot did not receive the data from one of the button presses because we wrote a code to avoid busy-waiting to make sure the same button press is not registered multiple times when it's only pressed once.

Potential improvements in the future

We could have added a code which prints a message saying 'button press registered' every time a new number is added to the array. This would be helpful because the user would know what button was not registered so they could press the button again to feed the data in the array. However this was not mandatory and the game runs fine. This is more of a quality of life change.

Conclusion

This report shows the design and implementation plan for the Simon Swift game. The Simon Swift game depends on interactive memory and utilises SwiftBots LEDs and buttons. The main goal is to create a Java program that allows users to interact by engaging them in a sequence that is progressively harder which strengthens the memory.

The main functionalities are the random colour sequence generator, a score tracker for the whole game and every round, interactive buttons. Additionally, we made clear game rules which include the right terminating conditions and the option for plates to quit or continue after every 5 rounds.

The design of the algorithm is shown through flowcharts which ensures a clear understanding of the program's logic and data structures. This approach makes implementation very effective as well as makes teamwork skills and problem-solving skills better.

Overall, This project helps improve important skills in software development and group work. By using the feedback from the tutor meetings This group plans to create a successful and functional game.

References

Brunel University London. (n.d.). *Brightspace: Lesson 51578, Topic 1706829*. Available at: https://brightspace.brunel.ac.uk/d2l/le/lessons/51578/topics/1706829 (Accessed: 5 December 2024).

Brunel University London. (n.d.). *Swiftbot Maven Repository*. Available at: https://swiftbot-maven.brunel.ac.uk/ (Accessed: 5 December 2024).

W3Schools. (2019). *Java Tutorial*. Available at: https://www.w3schools.com/java/ (Accessed: 5 December 2024).