

SECTION

8

PROGRAMMING
ROBOTS

Robot Construction & Programming

Programming Robots

Introduction

In this section, you will explore the fundamentals of programming robots, focusing on creating efficient and logical computer programs. Through pre-designed flowcharts, you will learn to handle single and nested decision conditions, develop programs that incorporate controlled feedback loops with interrupts, and apply finite state machines (FSMs) to control various robotic use cases. These activities aim to strengthen problem-solving skills and enhance learners' ability to design and implement sophisticated robotic behaviours, preparing them for real-world applications in robotics and automation.

At the end of this section, you will be able to:

- Create computer programmes from pre-designed flowcharts that have single-decision conditions.
- Create computer programmes from pre-designed flowcharts that have nested decision conditions.
- Create computer programmes from pre-designed flowcharts that have a controlled feedback loop with loop interrupts.
- Formulate and programme FSMs for controlling different use cases.

Key Ideas:

- **Programming robots:** Giving instructions to robots to tell them what to do and how to do it.
- **Block-based coding:** A way of programming that uses visual blocks you can drag and drop, like building with LEGO® bricks
- **Text-based coding:** Traditional programming where you write out instructions using words and symbols. It is more complex than block-based programming.
- **Flowcharts:** Diagrams that show the steps of a process or program using different shapes connected by arrows.
- **Control blocks:** Special blocks in programming that help make decisions or repeat actions. Examples include:
 - If-Then: Does something only if a condition is true.
 - If-Then-Else: Does one thing if a condition is true, and another if it is false.
 - Wait Until: Pauses the program until something specific happens.

- Repeat Until: Keeps doing something until a certain condition is met.
- **Nested decision conditions:** Putting one decision inside another, like a set of Russian nesting dolls, to make more complex choices.
- **Translating flowcharts to code:** Turning the steps in a flowchart into actual programming instructions.
- **Single-decision conditions:** Simple yes/no choices in a program.
- **Finite State Machines:** A Finite State Machine is a system used in computing and electronics that moves between different states based on specific inputs. Each state represents a different condition, and the machine can only be in one state at a time. FSMs are used in devices like traffic lights, vending machines, and robots to control sequences of actions.
- **Controlled Feedback Loop:** A Controlled Feedback Loop is a system where the output of a process is monitored and used as input to make adjustments. This helps maintain the system's performance or stability. For example, in a thermostat, the temperature is constantly measured and adjusted to keep a room at the desired temperature.

INTRODUCTION TO PROGRAMMING WITH BLOCK-BASED APPLICATIONS

In this section you will explore the basics of how to give instructions to robots, just like giving a recipe to follow. You will learn how robotics uses computer programming to control robots and make them perform specific tasks. You will explore block-based programming, This is a method of programming which uses graphical blocks (pictures) to represent code concepts. Instead of writing text-based code, you drag and drop these blocks to create programs, making it easier to learn and understand the principles of programming.

What is Programming?

Programming is like giving a computer or robot a set of instructions to follow, similar to a recipe. It uses special languages called programming languages, which have specific rules (syntax) to tell the computer exactly what to do step-by-step to complete tasks.

There are several programming languages for example C, C++, Python, and Java. These languages are text based and can be written using syntax on any laptop or computer.

No text is required in block based programming, for example Scratch and Snap. These block programming languages use graphical blocks that represent code concepts, which you drag and drop to create programmes.

This is an example of programming using two languages, block and text based.

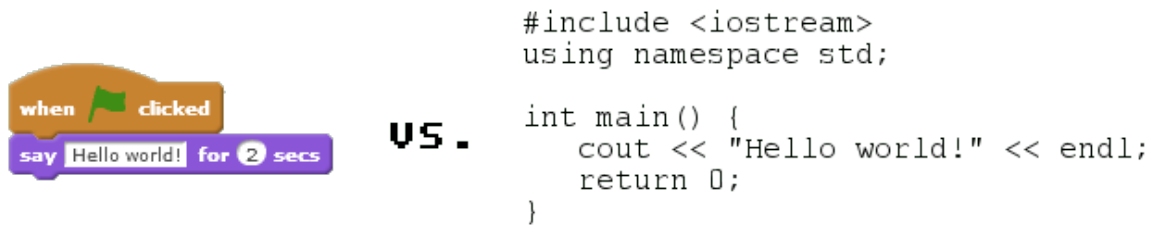


Fig. 8.1: Block-based in Scratch vs. Text-based Programming in C (techresort.org)

Block-Based Programming: A Stepping Stone to text-based Programming.

Block-based programming offers many advantages over text-based programming. Some of these advantages include:

1. **Easy to Learn:** The drag-and-drop interface means you do not need to remember rules, it allows you to test different options.
2. **Colourful and attractive:** The colourful blocks and animations make it easy to learn.
3. **Reduces Errors:** The blocks fit together logically, and mistakes can be easily corrected. This allows you to focus on the core programming concepts.
4. **Experimentation Encouraged:** The visual nature of block-based programming makes it easy to experiment and try different things without worrying about getting it wrong.

A Brief History of Block-Based Programming

In 2003, MIT developed Scratch, one of the most popular block-based programming platforms.

Block-based programming allows you to create a variety of interactive projects, including:

- **Games:** Design chase games, clicker games, or even ping-pong games using blocks to control characters and objects.
- **Interactive Animations:** Bring your imagination to life by creating animated stories and characters.
- **Program Robots:** Create on-screen sprites that can be programmed to move and interact with their environment. This can simulate basic robot behaviour like navigating a maze and avoiding obstacles.

Block-based Programming with LEGO® Education SPIKE™! Application/IDE

In this section, you will be using the LEGO® Education SPIKE™ app, which employs block-based programming to animate LEGO® creations. This app uses visual blocks that snap together like LEGO® bricks to build programs. Detailed tutorials for each block are available in the “Help” section, shown in the diagram below:

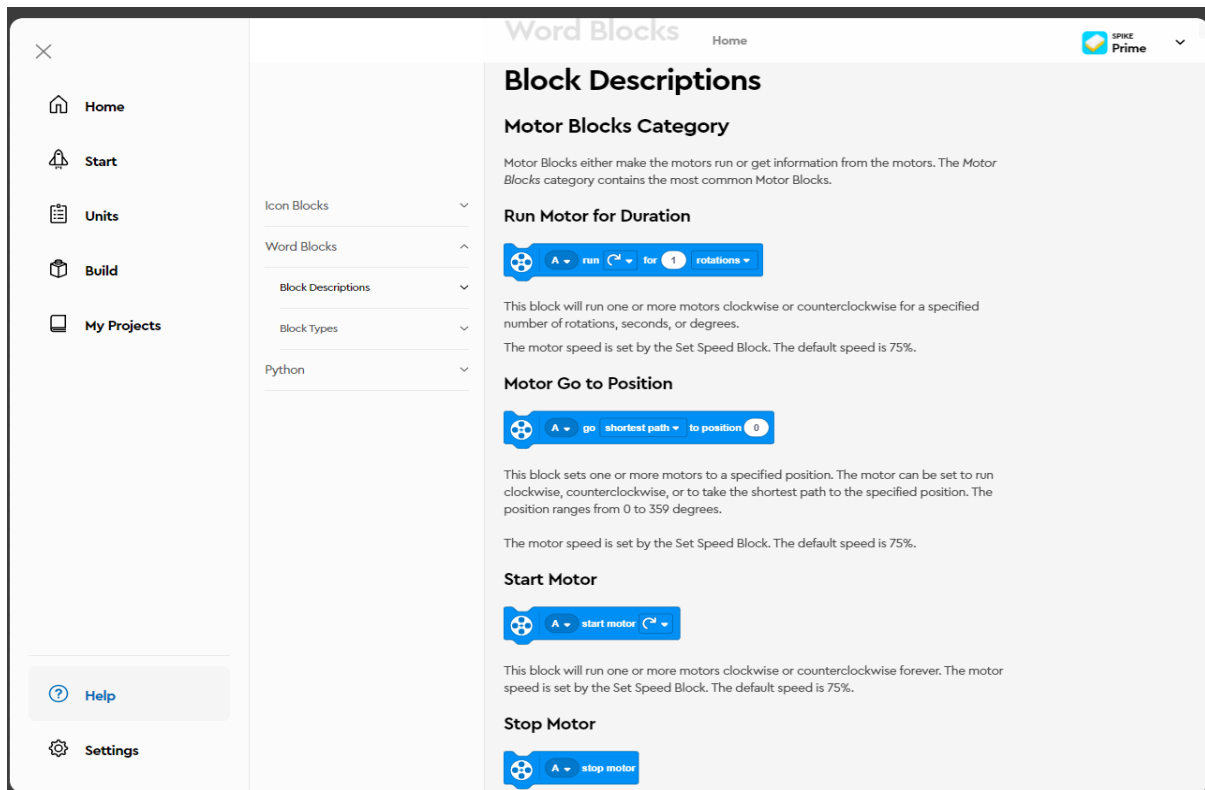


Fig. 8.2: Block Descriptions of each block under the Help Section of LEGO® Education SPIKE™! app

Important Instructions

The following links will help you learn how to use the LEGO® Education SPIKE™ app.

The first is a playlist of eight videos which provides a quick overview of using the LEGO® Education SPIKE™ Prime Kit. It provides tutorials for using both the word-blocks and Python Programming Language. Focus on using the word-blocks not the Python Programming Language.

The second playlist of seventeen videos reviews some of the points made in the first, and using some practical examples, it provides a detailed explanation of how to use the various blocks. Learners can focus on the first fifteen videos in this playlist. The resources are as follows:

1. SPIKE Prime Tutorials https://www.youtube.com/playlist?list=PL_zXBalpjbU33gw5CML3DtL7fN8640qku
2. LEGO® Spike Prime <https://www.youtube.com/playlist?list=PLS9qLR8VoFA62KcAzsUfAOQgLRrEXCp78B>

Using these videos might take some time, so it is best to spread them out carefully over the next few weeks.

Activity 8.1

With a friend discuss what programming means.


After your discussion decide the **main purpose** programming serves in robotics.

Using one example of how programming is used in robots is to actuate movements, discuss the different ways programming can be used to control the tasks robots need to do.

Write a short definition of programming and give three examples how programming has been used in a robot to solve real-world problems.

Activity 8.2

1. Watch the video below and take notes. Either on your own or with two friends, list the differences between block-based and text-based programming. Discuss the advantages and disadvantages of each type of programming.

Link	QR CODE
https://www.youtube.com/watch?v=PI5yZGqGRWI	

2. On your own, draw a table with two columns which shows the differences between text based and block-based programming. State which you would prefer to use and why.

Activity 8.3

Your teacher will put you into small groups for this activity.

In this activity you will explore the LEGO®SPIKE app interface. The Projects section is where you'll find the block-based programming interface. The Programming Blocks menu contains the visual blocks that represent different programming commands. The Block Palette contains all the different types of blocks you can use, categorised by function (motion, events, control, sensors). They snap together like LEGO® bricks to form a sequence of instructions. Create

a simple program to make a robot move or respond to a sensor by following these points.

1. Explore the programming and palette blocks of the app, find out what each does.
2. Create a short program that uses at least three different types of blocks.
3. Each group will present their programs to the class.

Activity 8.4

You will need to work with a friend for this activity.

Using the LEGO® SPIKE kits, build a basic robot structure. Your structure should have sensors and something that moves like wheels or arms. Then, use the SPIKE app to program the robot to perform a task such as moving in a square or responding to a sensor input.

1. In pairs, design a robot and use the SPIKE app to program the actions you want it to take.
2. Your program should have loops, sensors, and at least one other block-based feature (e.g., sound, lights).

CREATING COMPUTER PROGRAMMES FROM FLOWCHARTS

In this section, you will learn how to make a block-based program from a flowchart.

If you have worked your way through the earlier video tutorials and become familiar with the LEGO® Education SPIKE™ app it will help you understand the section.

An important consideration in this section is the meaning and use of words related to programming. Both terms, “block-based coding” and “block-based programming,” are commonly used and generally understood to mean the same thing. However, “block-based programming” is often preferred in learning and professional contexts because it emphasises the broader concept of programming, which includes not just writing code but also understanding logic, structure, and problem-solving.

So, for the overall process of creating programs using blocks, “block-based programming” is the term mainly used in this section.

Translating Flowcharts to Blocks:

The flowchart simply represents a map to solve your defined problem to action the desired task. Each flowchart symbol usually translates into a specific block(s) in your program. Here is a table which will remind you of the basic symbol and their function:






Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

Fig. 8.3: Basic symbol and their function

Note that the decision diamond in the table is represented as a hexagon in the app. In the LEGO® Education SPIKE™ app, the equivalent of input/output in a flowchart is represented by the sensor blocks and output blocks.

- a. **Start/End:** In the LEGO® Education SPIKE app, “Start” and “Stop” blocks are found under the Events category. Start blocks are rounded at the top and are the first blocks used, sometimes needing extra information like sensor input. Stop blocks have a flat base and are usually the last blocks, with no other blocks snapping to their bottom.

Eleven start blocks and one stop block are shown below in **Fig. 8.4**.

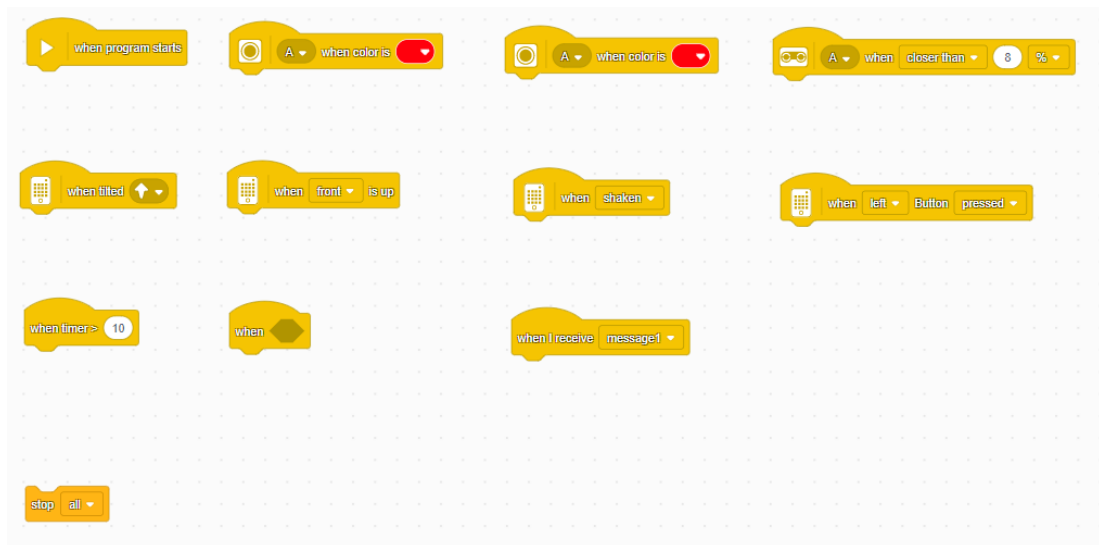


Fig. 8.4: Start and Stop Blocks in LEGO®Education SPIKE app.

- b. Process:** There are a number of these blocks which define the action to be carried out. These processes may be found under the motors, movement, light and sound block categories in the LEGO®Education SPIKE app. Some examples are presented in Fig. 8.5.

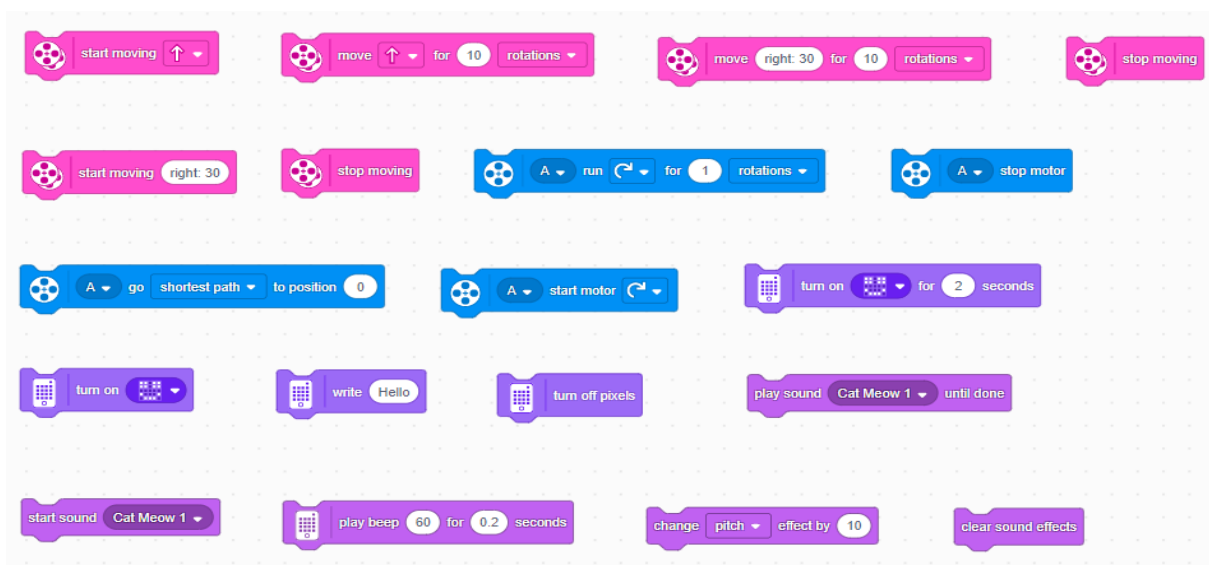


Fig. 8.5: Examples of Blocks that carry out processes in the LEGO®Education SPIKE app.

- c. Decision:** To make decisions based on conditions, you use control blocks like **if-then**, **if-then-else**, **wait until**, and **repeat until**. These blocks have a hexagon shape where you can define the condition to check.

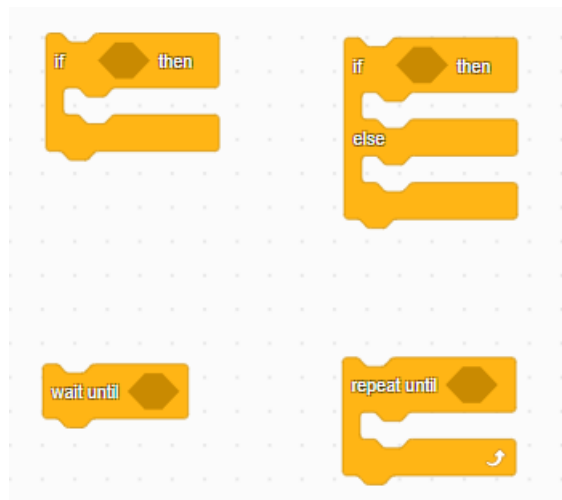


Fig. 8.6: Examples of Control Blocks in the LEGO®Education SPIKE app that are used to implement decisions.

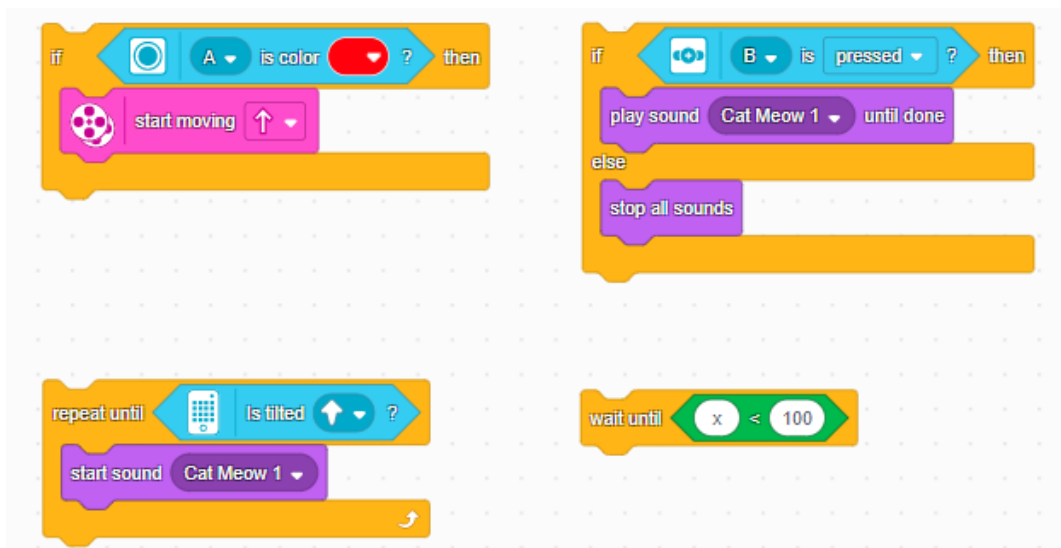


Fig. 8.7: Examples of Control Blocks with specified conditions used to implement decisions.

- d. Connector:** The arrows in a flowchart simply guide you in connecting the correct blocks in your program. They tell of the logical flow of your program and the order in which they are to appear.

Mastering Conditions: When to Use Which Block

Read the following guidelines which will help you decide which block to use.

If-Then Block: Use this block when the robot needs to perform an action only if a specific condition is true. In this case, you are not concerned about what to do when the condition is not true. Think of it as a “yes or no” question where we are only concerned with providing an action when the answer is yes. If the answer is “yes,” the robot performs the action within the block.

This example shows the same action first in a flowchart then in a block program:

- a. **Flowchart:** A robot which moves in the forward direction needs to stop only if it detects an object in front of it using an ultrasonic sensor. If the object is less than 10cm to it must stop. This is shown as a decision in this flowchart diagram:

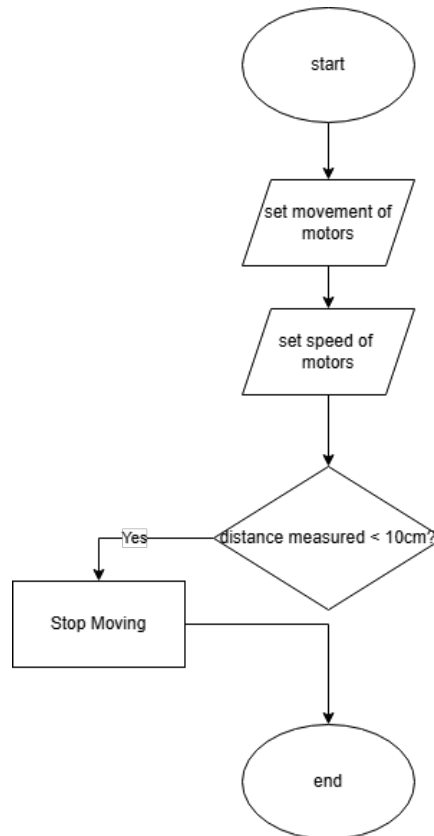


Fig. 8.8: Flowchart depicting a situation with a condition.

- b. **Block Program:** An **if-then** block checks the ultrasonic sensor. If the sensor detects an object in front of it closer than 10cm (true), the robot stops moving.

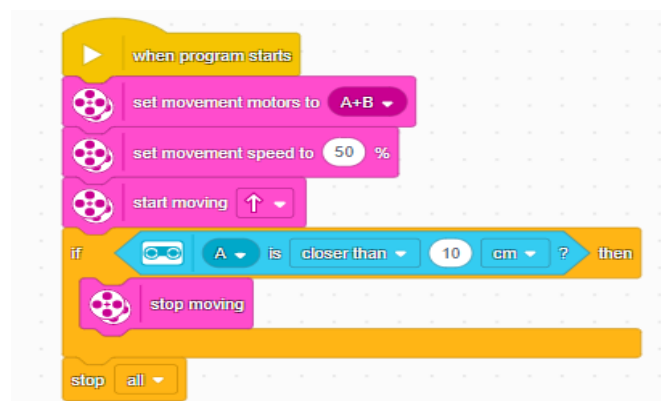


Fig. 8.9: Block program which implements the flowchart using an **if-then** block.

If-Then-Else Block: This block allows for two possible actions based on the condition. If the condition is true, the robot performs one set of actions within the “If” section.

If the condition is false, the robot performs a different set of actions within the “Else” section.

Example:

- a. Flowchart:** A robot installed with a colour sensor is to move forward only when it sees a black colour else it should stop moving.

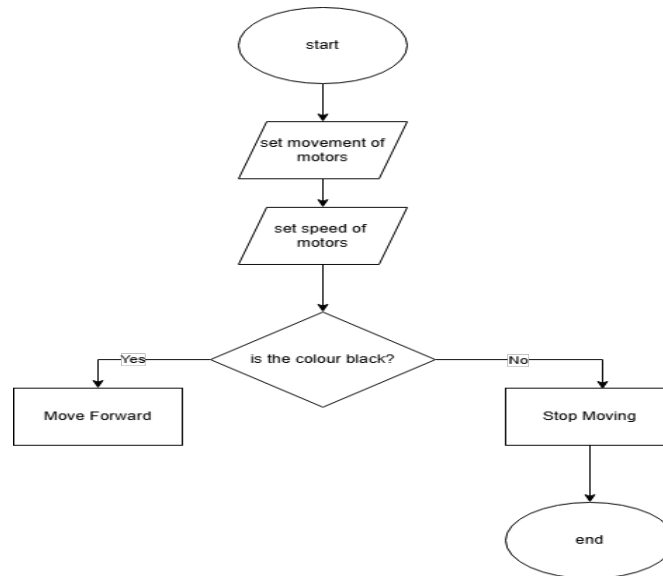


Fig. 8.10: Flowchart depicting a situation with a condition

- b. Block Program:** An **if-then-else** block” checks the line sensor. If the sensor detects black (true), the robot uses a “Turn Left” block. If it does not detect black (false), the robot uses a “Move Forward” block.

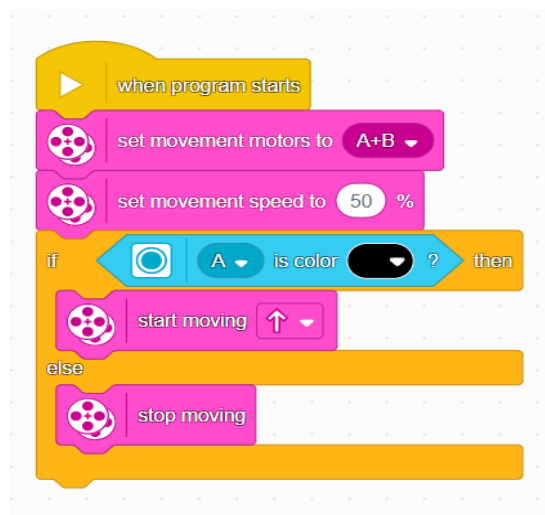


Fig. 8.11: Block program which implements the flowchart using an **if-then-else** block.

Wait Until Block: This block pauses the program until a specific condition becomes true. This is useful for situations where the robot needs to wait for something to happen before proceeding.

Example:

- a. **Flowchart:** A robot with a colour sensor must wait until it sees the colour green before moving forward.

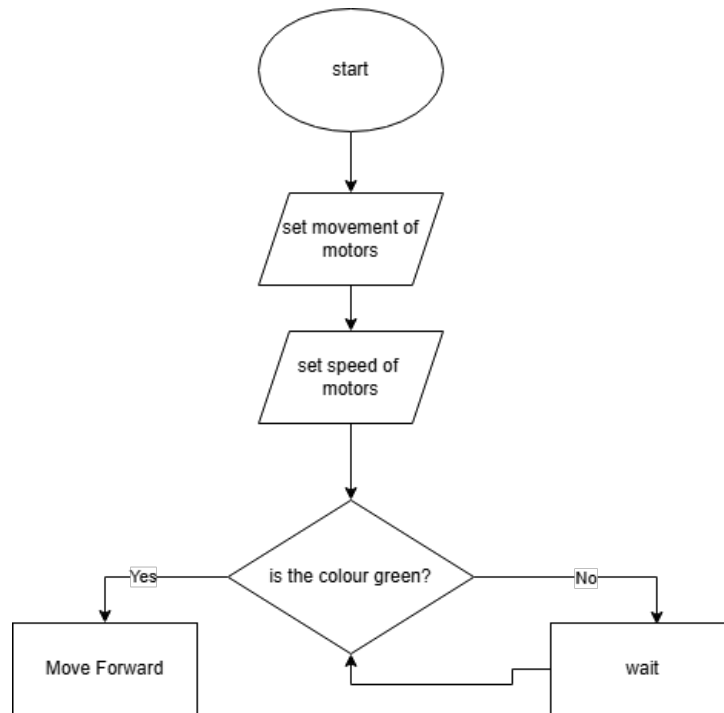


Fig. 8.12: Flowchart depicting a situation with a condition.

- b. **Block Program:** A **wait until** block checks the colour sensor. The program pauses here until the colour sensor detects the colour green (true). Once the colour green is detected, the robot uses a “Move Forward” block.

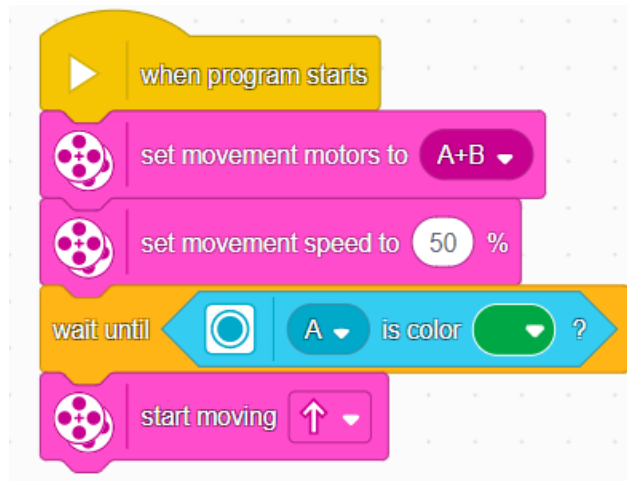


Fig. 8.13: Block program which implements the flowchart using a wait-until block.

Note that the above situation could equally have been implemented using an **if then** block together with a **forever** block as shown below.

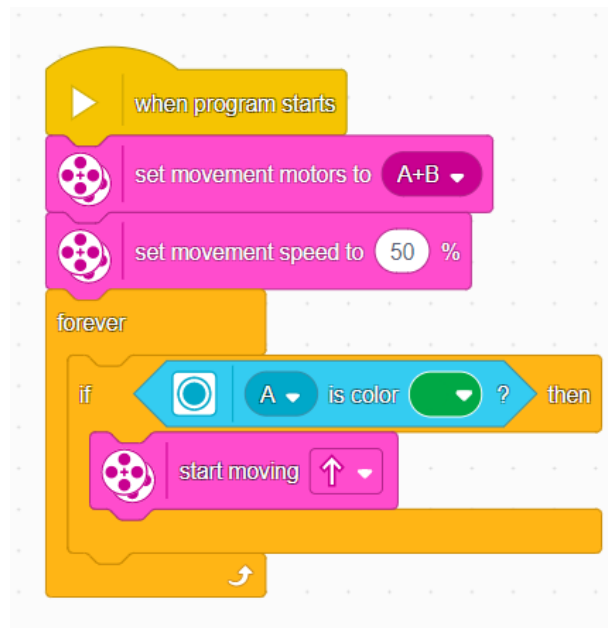


Fig. 8.14: An equivalent of the **wait-until** block program

Repeat Until Block: This block repeats a series of actions until a specific condition becomes true. It is like a loop that keeps running until a certain requirement is met.

Example:

- a. **Flowchart:** A robot needs to keep playing a tune until it sees a green colour.

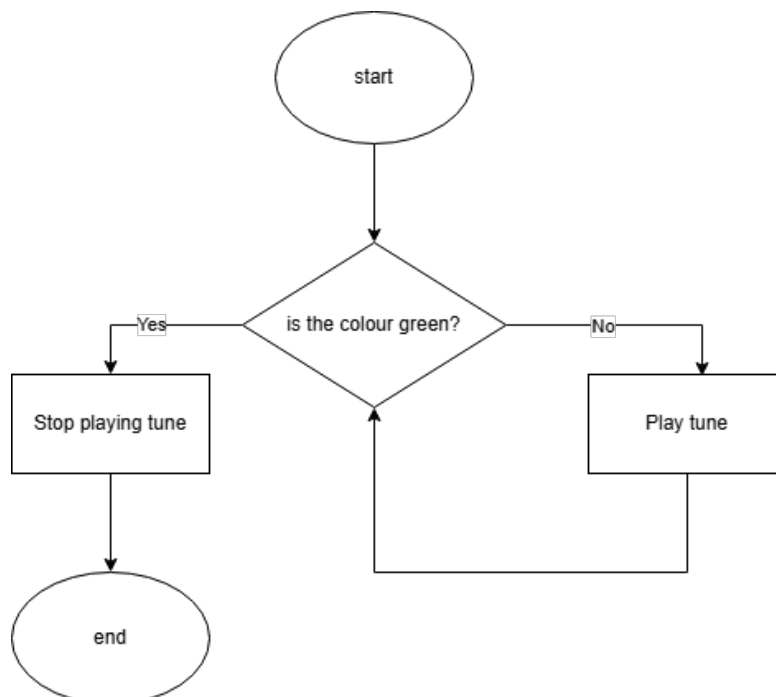


Fig. 8.15: Flowchart depicting a situation with a condition.

- b. **Block Program:** A **repeat until** block contains the actions for playing the tune (meow tune). The loop keeps repeating until the colour sensor detects the colour green (true), at which point the loop stops.

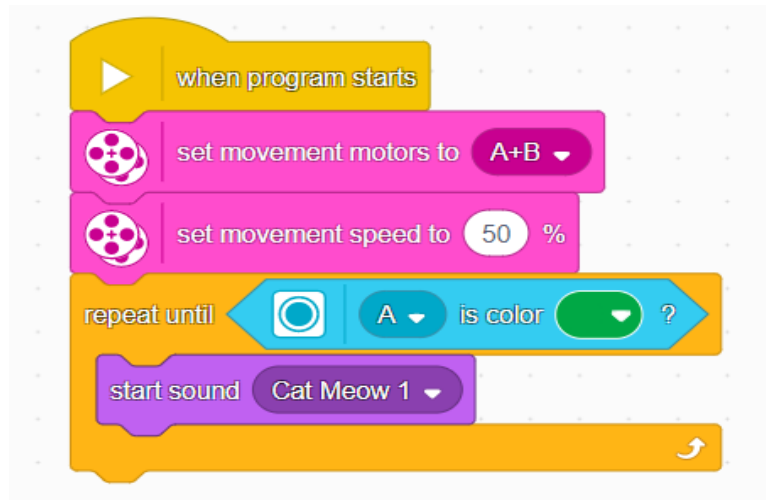


Fig. 8.16: Block program which implements the flowchart using a **repeat-until** block.

Mastering Conditions: Nested Statements

You have mastered using “If then”, “If then Else,” “Wait Until,” and “Repeat Until” blocks to create awesome robot programs. But what if your robot needs to make even more complex decisions? That is where nested blocks come in.

Think of a set of Russian nesting dolls – the bigger doll holds a smaller doll inside. Similarly, nested blocks allow you to place one code block inside another. The outer block controls when the inner block’s code executes.

Why Nest Blocks?

Sometimes, a simple “If-then” or “If-then-Else” statement is not enough. Nesting allows for more complicated decision-making within your program to deal with situations where there are either multi-step decisions or conditional actions with loops. For example, imagine a robot programmed to follow a line. It should move forward when it detects the line. But, if it is only set to be done only for a period of time, say 10 seconds, then the first block (action) should be nested within the loop, which checks the time. So, a check of the time is made before it moves. As long the check shows that it is within the time then the robot can keep following the line.

This example is depicted using the following flowchart and block programs.

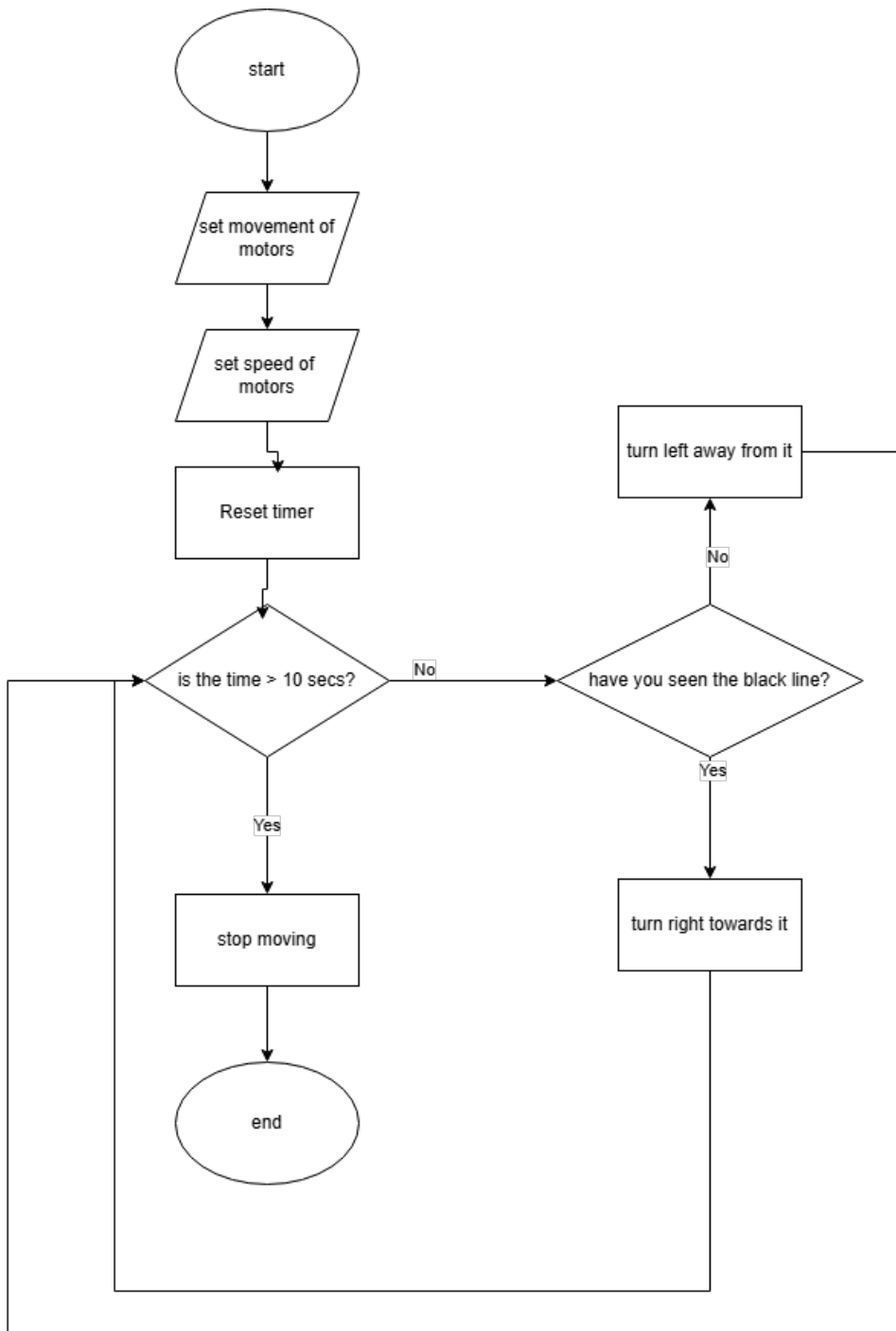


Fig. 8.17: Flowchart showing a line following robot which uses nested blocks (multiple decisions)

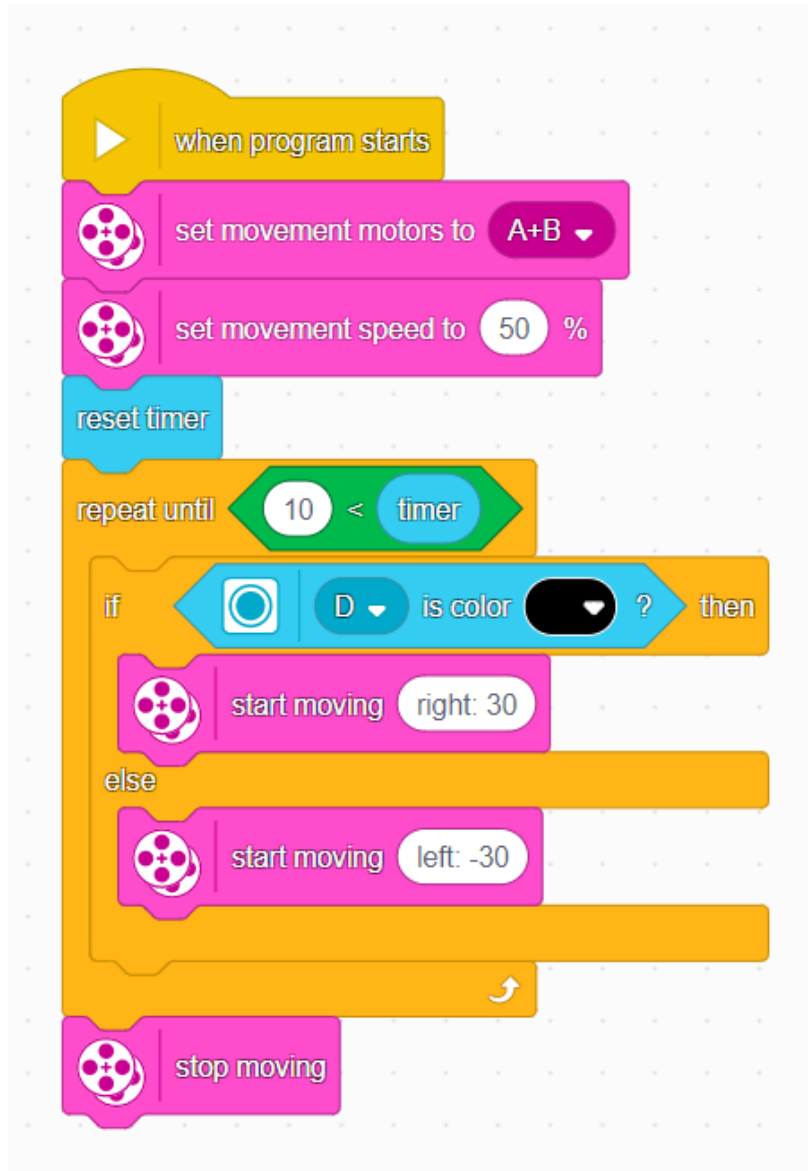
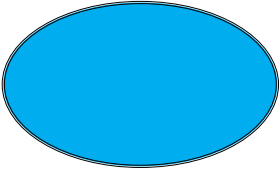
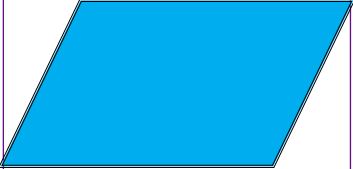

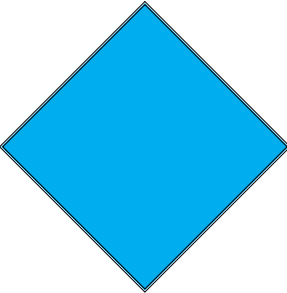


Fig. 8.18: Block program showing a line following robot which uses nested blocks.

Always remember that the inner block's code only executes when the outer block's condition is met. Think of it as a special permission to run the code inside.

Activity 8.5: Identifying Flowchart Symbols and Block Types

Working on your own, copy and fill in the table below. Write the name of the flowchart symbol, its function and match it with the block type described in the section above for the LEGO® Education SPIKE™ app. Explain the function of each block in the LEGO® Education SPIKE™.

Symbol	Name	Function	Block Type
			
			
			
			

Activity 8.6: Control Block Scenarios

For this activity you should be working in pairs.

Scenarios for a block-based programming app (e.g., Scratch or LEGO® Education SPIKE™)

- a. A robot should start running and stop when it detects an obstacle 5cm in front of it.
- b. A light should blink until a button is pressed.

For each scenario identify the appropriate control blocks that would be used to complete it. Discuss when and why you would use each type of control block.

Afterward, program these scenarios using a block-based programming app (e.g., Scratch or LEGO® Education SPIKE™) to see how each control block functions in real-time.

Write an evaluation of each scenario including what went well, what you could have done better, any modifications you made after entering your blocks into the app.

Activity 8.7: Applying Nested Control Blocks

For this activity, your teacher will put you into small groups.

- a. Draw a flowchart diagram using the correct symbols that represents the following statement: “If temperature > 30, then check if humidity > 70, else display ‘cool weather’”.
- b. Translate the flowchart into a program by drawing the appropriate control blocks. You will need to nest blocks to include the check.
- c. Program your drawing using a block-based programming app (e.g., Scratch or LEGO® Education SPIKE™) to see how each control block functions in real-time.

Activity 8.8: Presenting and Peer Review

Exchange the flowchart and program from activity 8.7 with another group.

- a. Review each other’s work and provide constructive feedback on the flowchart and program.
- b. Suggest possible improvements or alternative solutions.
- c. Act in the feedback if you need to make your solution to activity 8.7 work.
- d. Present your work to your teacher with an evaluation which describes what you did right and what you could have done better.

FUNDAMENTALS OF CONTROL PRINCIPLES IN AUTOMATION AND ROBOTICS - FEEDBACK AND NON-FEEDBACK LOOP SYSTEMS

This section explores two essential concepts in automation and robotics: **Finite State Machines (FSMs)** and **Controlled Feedback Loops**. These are critical tools in controlling robot behaviour and other automated systems. FSMs work by transitioning between specific states, such as turning or waiting, based on events. Controlled feedback loops, on the other hand, continuously monitor variables like temperature or position and adjust actions in real-time. Understanding these control mechanisms will enable you to design simple robotics systems, applying them to real-world scenarios like line-following robots or automated.

Finite State Machines:

A finite state machine (FSM) defines a set of distinct states, such as “waiting,” “moving forward,” or “turning,” along with the specific events that trigger transitions between these states. A simple example is a traffic light controller; it exists in three distinct states (red, yellow, and green) and transitions between them are based on a timer (the event).

Characteristics of FSMs:

1. **Finite Set of States:** An FSM operates within a defined set of states. These states represent different conditions or stages the system can be in at any given time. Examples include “waiting,” “moving forward,” “turning left,” or “off.”
2. **Transitions:** Transitions are well-defined pathways between states. Specific events or conditions trigger them. For instance, a robot in a “waiting” state might transition to a “moving forward” state when it detects a button press (the event).
3. **State-Based Decisions:** The actions taken by the FSM depend on the current state and the triggering event. This means the system’s behaviour is determined by its current state and the specific event that occurs.
4. **Deterministic or Non-deterministic:** FSMs can be either deterministic or non-deterministic. In a deterministic FSM, for every state and event, there is only one possible next state, ensuring predictable behaviour. In a non-deterministic FSM, there can be multiple possible next states for a given state-event combination, introducing an element of randomness or flexibility.
5. **Output Generation:** While the primary focus of FSMs is on state transitions, they can also generate outputs in each state. These outputs might involve controlling motors, activating sensors, or displaying information.
6. **Simplicity and Readability:** FSMs offer a clear and concise way to represent complex systems. By breaking down the system into states and transitions, it is easier to understand the logic and control flow.
7. **Modular Design:** FSMs can be modular, allowing for the creation of larger systems by combining smaller FSMs. This promotes code reusability and simplifies complex system design.

Controlled Feedback Loops:

A controlled feedback loop focuses on continuous monitoring and adjustments. It constantly checks a specific value (like temperature or distance) and adjusts the robot’s actions based on that data. Imagine a line-following robot that uses a colour sensor to stay on the line. It continuously checks the sensor reading and makes adjustments if it goes off track.

Characteristics of Controlled Feedback Loops:

- 1. Continuous Monitoring:** The system constantly tracks its output value. This might involve using sensors to measure temperature, distance, colour, or any relevant parameter.
- 2. Error Detection:** The monitored output is compared to a predetermined reference value or desired outcome. This comparison helps identify any deviations or errors between the actual and desired states.
- 3. Adjustment and Error Correction:** Based on the detected error, the system initiates corrective actions or adjustments. This ensures the output is brought closer to the desired value. Systems that can dynamically adjust their behaviour based on feedback are called self-correcting systems. They continuously learn and adapt to maintain optimal performance.

Key Differences:

While both FSMs and controlled feedback loops play a crucial role in robot control, they operate in slightly different ways as tabulated below:

Feature	Finite State Machine (FSM)	Controlled Feedback Loop
Structure	Defined states and transitions	Continuous loop with checks inside
Decision Making	Based on the current state and triggering event	Based on the most recent sensor reading
Examples	Traffic light controller, robot performing a sequence of tasks	Line follower, temperature control system

Activity 8.9

Your teacher will put you into groups for this activity.

In this task, you need to create a system that adjusts itself based on feedback and another system that changes between different states based on inputs.

- 1.** Build a line follower robot equipped with two colour sensors using a kit of your choice.
- 2.** Draw a flowchart to show how the sensor readings adjust the robot's motion and direction.
- 3.** Write a line following program using a block programming app which represents the flowchart. Take a screenshot of your program.
- 4.** Identify with labels which parts of your flow chart and block program represent the FSM and Controlled Feedback Loop.
- 5.** Test your program and evaluate the results.

6. Each group member should give a labelled flowchart, block program picture and evaluation to their teacher.
7. Compare your flowchart and program logic to Figures 8.19 and 8.20: Are they different?

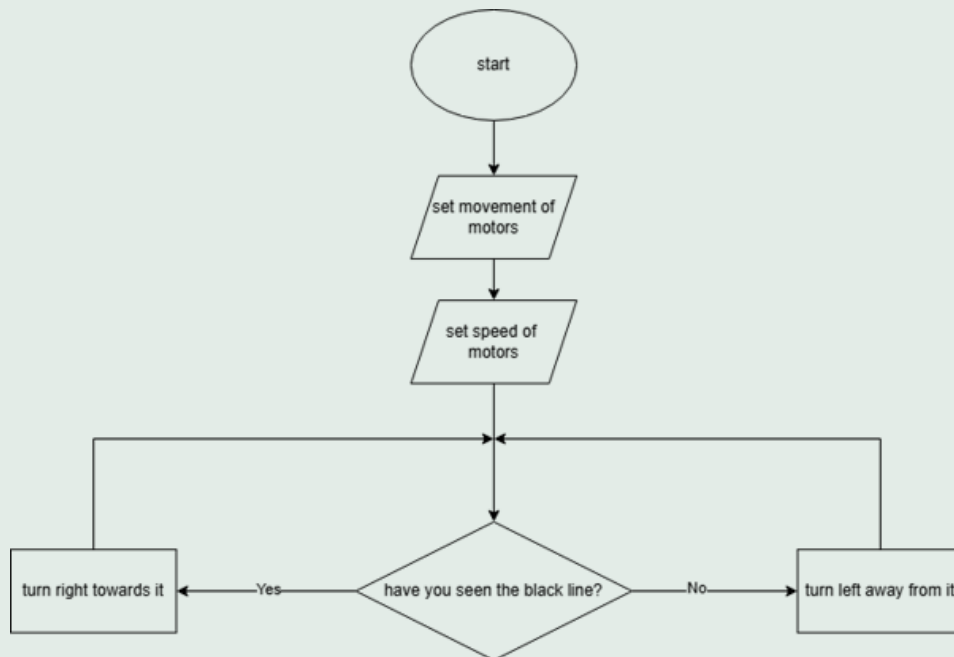


Fig. 8.19: Flowchart showing a line following robot.

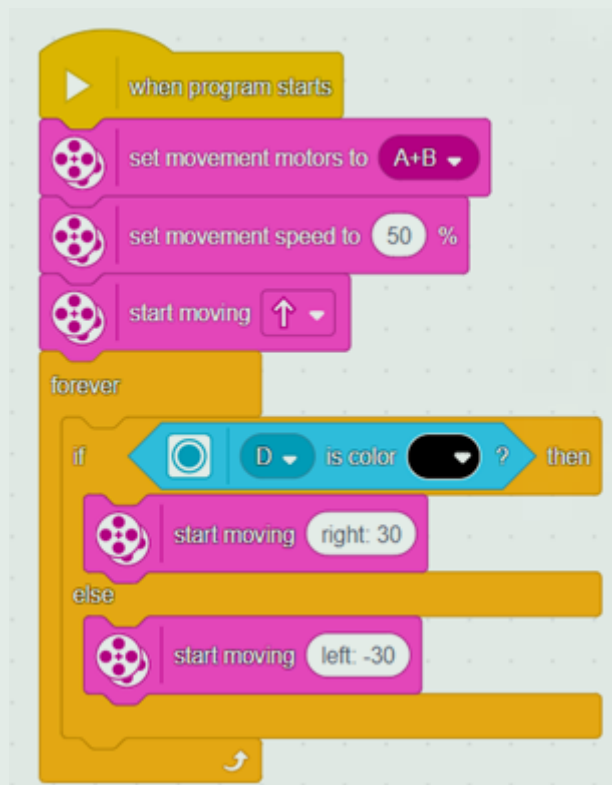


Fig. 8.20: Block programme showing a line following robot which uses nested blocks.

Implementation of a Finite State Machine: Sorting Robot

A robot tasked with sorting coloured balls into different bins can be controlled as a finite state machine (FSM) with several states and transitions. (remember that transitions mean changes from one state to another) These states and transitions are described below.

States:

1. **Waiting:** The robot is in its initial position, ready to receive a ball.
2. **Sense Colour:** The robot uses a colour sensor to identify the ball's colour (e.g., red, blue, yellow).
3. **Move to Bin:** Based on the sensed colour, the robot transitions to the appropriate bin state (e.g., Move to Red Bin, Move to Blue Bin, Move to Yellow Bin).
4. **Deposit Ball:** The robot positions itself over the designated bin and releases the ball.

Transitions:

- From “Waiting” to “Sense Colour”: This transition occurs when the robot detects a ball.
- From “Sense Colour” to one of the “Move to Bin” states: This transition depends on the colour identified by the sensor.
- From “Move to Bin” to “Deposit Ball”: This occurs when the robot reaches the designated bin.
- From “Deposit Ball” to “Waiting”: This transition ensures the robot is ready for the next ball.

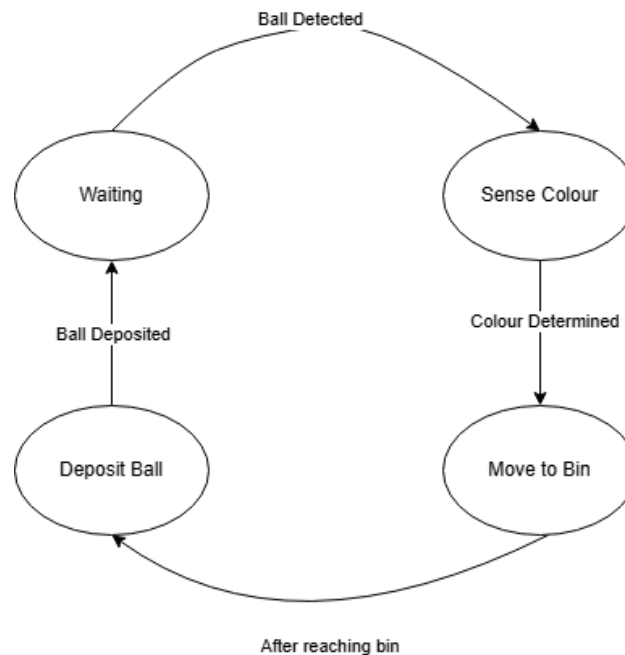


Fig. 8.21: FSM for ball sorting robot

This example demonstrates how a finite state machine can effectively control a robot that performs a sequence of tasks based on sensor data and state transitions.

Introduction to Functions in Block Based Programming

This section explains the use of functions in block-based programming.

In FSMs, states are often represented as functions or routines. A function is a group of blocks that perform a specific task. Here is the procedure for a block-based programming app.

- In your app look for the option to create a new function or custom block.
- Define the function by dragging and dropping the required blocks for the task.
- Add the inputs and outputs the function will need.
- Save it with your custom name.
- When you need to perform that task, you call the function.

This procedure is explained for LEGO® Education SPIKE™ in the tutorials you have already seen on **page [24]**.

Using a function in block-based programming is shown below in Fig. 8.22.

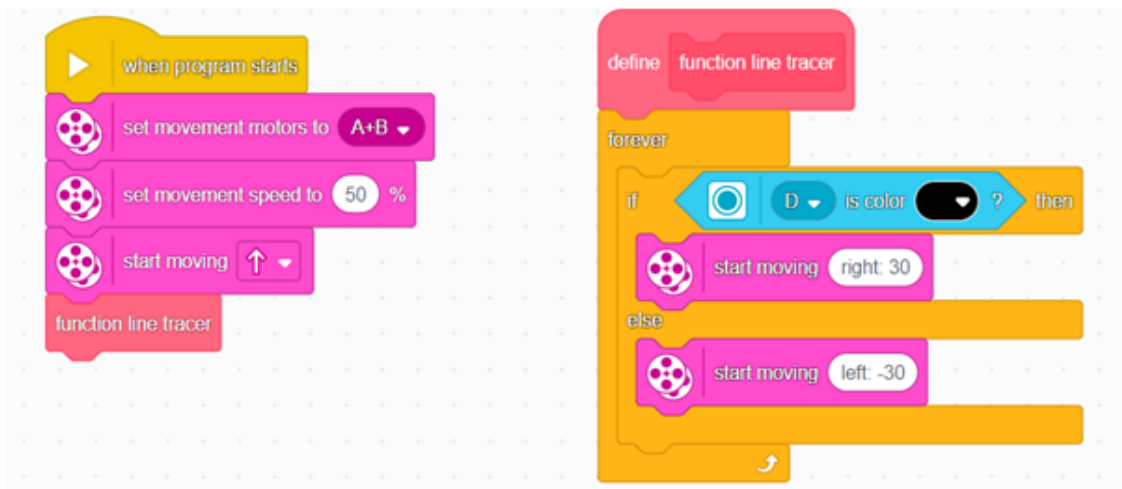


Fig. 8.22: Using functions in block-based programming.

Activity 8.10

1. Create an FSM diagram for a robot which has to navigate its way out of simple a rectangular maze.

This should be a structured roadmap for a robot's actions.

Here are the first two actions for your FSM diagram:

1. Start State:

- Action: Initialise sensors and motors.
- Transition: Move to “Move Forward” state.

2. Move Forward:

- Action: Move the robot forward.
- Transition:
 - If front sensor detects a wall, move to “Turn Right” state.
 - If left sensor detects an open path, move to “Turn Left” state.

2. Develop function blocks to represent states for the FSM diagram they created

Consider what tasks you want the robot to perform when you call the function.

Here is the function for the first state for your maze navigating robot:

a. Start State:

- Function Name:** ‘initialise’
- Description:** Sets up sensors and motors
- Blocks:**
 - [Initialise Sensors]
 - [Initialise Motors]

Review Questions.

1. Scratch is an example of a block-based programming app. (True/False)
2. In the LEGO® Education SPIKE app, which block would you use to make a robot move?
 - a) Sound block
 - b) Motion block
 - c) Sensor block
 - d) Loop block
3. Explain programming in your own words.
4. Why is programming important in Robotics?
5. Explain the difference between text based and block-based programming
6. True or False: In a flowchart, a diamond shape is used to represent a decision point where conditions are evaluated.
7. Which of the following flowchart symbols represents a process or action step?
 - a. Oval
 - b. Rectangle
 - c. Diamond
 - d. Arrow
8. Which control block would you use in a block-based programming platform to repeat an action until a specific condition is met?
 - a) If-Then
 - b. Wait Until
 - c. Repeat Until
 - d. If-Then-Else
9. What is the main difference between an “If-Then” and an “If-Then-Else” control block in block-based programming?
10. Why is it important to use nested decision conditions in programming? Provide an example where nested conditions would be necessary.
11. Define a Finite State Machine (FSM) and give one example of where it can be used.
12. What is a controlled feedback loop, and how does it differ from an FSM?
13. Explain how a traffic light system can be modelled as an FSM.
14. Describe one real-world application of a controlled feedback loop.

Answers to Review Questions

1. True
2. b
3. Programming is the process of writing instructions to control the actions and behaviours of a robot.
4. Programming is important in robotics because it allows you to define how a robot senses its environment, processes information, and performs tasks, enabling it to operate autonomously or semi-autonomously. Essentially, programming brings robots to life by telling them what to do and how to do it.
5. Block-based programming uses visual blocks to represent code, making it easier to understand programming concepts without memorising syntax. Text-based programming requires writing lines of code using a programming language from memory.
6. True
7. b) Rectangle
8. c) Repeat Until
9. The “If-Then” block executes an action only when a condition is true, while the “If-Then-Else” block provides two outcomes: one when the condition is true and another when the condition is false.
10. Nested decision conditions are important for handling complex logic where multiple criteria need to be evaluated. For example, in a temperature control system, nested conditions can check if the temperature is too high, and then further check if the humidity is also high before triggering an action like turning on a fan or air conditioner.
11. An FSM is a system that transitions between defined states based on specific events. Example: A traffic light controller transitions between “red,” “yellow,” and “green” based on a timer.
12. A controlled feedback loop continuously monitors outputs and adjusts inputs, while an FSM relies on specific state transitions. Feedback loops are used in systems that require constant adjustment, like a line-following robot.
13. A traffic light can be modelled as an FSM, with three states: red, yellow, and green. Transitions between these states occur based on a timer event.
14. A real-world application of a controlled feedback loop is a thermostat that continuously checks room temperature and adjusts the heating or cooling system to maintain a set temperature.

Extended Reading

- a.** Block Coding – An A To Z Guide
<https://www.codingal.com/coding-for-kids/coding-guides/block-coding-guide/>
- b.** Block-Based vs Text-Based Coding
<https://moonpreneur.com/blog/block-based-vs-text-based-coding/>
- c.** Getting started with LEGO® Education SPIKE™ Prime.
<https://education.lego.com/en-us/start/spike-prime/#Introduction>
- d.** Spike Prime - Tips and Tricks
https://www.youtube.com/playlist?list=PLxFeBwC-gJMSV_EILSS1K6k1dXSYudln9
- e.** Finite State Machines
<https://www.youtube.com/watch?v=4XEK7OU2gIw&pp=ygUdZHJhd2luZyBmaW5pdGUgc3RhdGUgZGlhZ3JhbXM%3D>

ACKNOWLEDGEMENTS



Ghana Education
Service (GES)



List of Contributors

Name	Institution
Isaac Nzoley	Wesley Girls' High School
Kwame Oteng Gyasi	Kwame Nkrumah University of Science and Technology
Kwame Owusu Opoku	Prempeh College