Your First Robot

01

Teaching Suggestion

Lesson 1 Your First Robot

Overview
15 mins

- Introduce the lesson by asking questions or telling interesting stories. - 10 mins
- Introduce the content and learning objectives of this lesson. - 5 mins

Inquiry
35 mins

- Open software tool mBlock 5 and give instructions. - 5 mins
- Interface introduction - 10 mins
- Graphical programming - 10 mins
- Add hardware device - 5 mins
- Connect device and upgrade firmware - 5 mins

Programming
40 mins

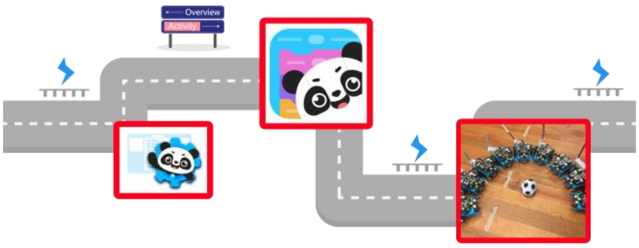
- Demonstrate the first program to the students. - 10 mins
- Students try on their own. - 5 mins
- Announce the challenge and let the students discuss in groups. - 10 mins
- Students complete the challenge and discover more possibilities. - 15 mins

Activity
25 mins

- Explain the rules of the game. - 3 mins
- Arranging the site and determining the team members. - 5 mins
- Start the game. - 12 mins
- Summary and clean up. - 5 mins

Reflection
5 mins

Part 1 Overview



In this lesson, students will learn about the features and basic operations of the mBlock programming software. Students in teams need to complete the assembly of their first mBot and learn how to program it. When ready, students will participate in a fierce competition with their mBots. During the competition, students not only need to master the operation of mBot, but also soft skills such as job assignment, teamwork, game strategy, and communication.

Objectives

I can successfully complete the assembly of my first robot and ensure that the components are securely mounted.

I can identify the name of the parts inside the kit and understand how to use it.

I can use the basic tools and understand the dimensions of all the fastening parts.

I can successfully install mBlock 5 software and build a programming environment for mBot.

I can locate and identify various functionalities in the mBlock5 software interface.

I can use a various ways to establish connection between the mBot and computer.

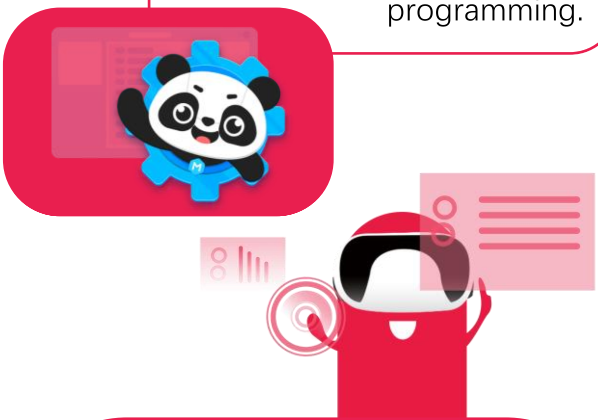
I can complete my first program to control the robot.

I can creatively complete the construction of basic functional structure as needed.

Part 2 Inquiry

What is mBlock 5?

mBlock 5 is a programming tool for STEAM education. It is inspired by Scratch 3.0 and supports graphical and text programming.



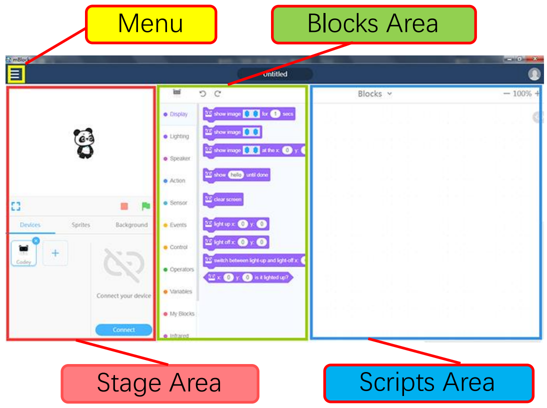
With mBlock 5, children are able to create engaging stories, games and animations, and program hardware like Makeblock robots, Arduino and micro:bit. It supports Python programming as well. You can just switch to Python mode with one-click. Moreover, its AI and IoT features give children a chance to have fun with some cutting-edge technologies. Besides, mBlock 5 allows you to sync programs across platforms between Web, mobile devices and PC.

Introduction

1.Download and install mBlock 5

Please visit：
http://www.mblock.cc/software/mblock/mblock5/

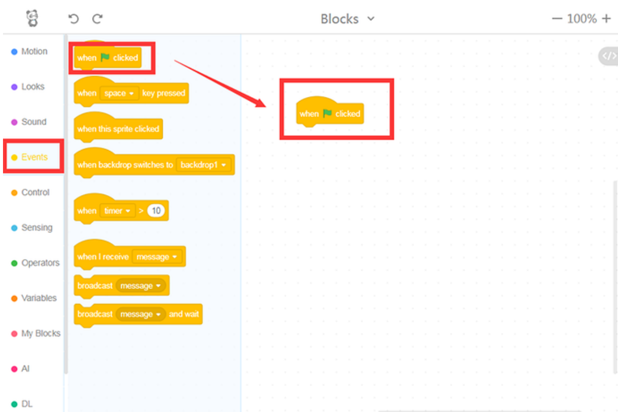
2. Interface introduction



Stage Area: You can present your designs, connect devices, set your sprites and backgrounds here.
Blocks Area: You can find the blocks you need by category and color in Blocks area.
Scripts Area: You can program in the Scripts area by dragging blocks to this area.
Menu: In this area, you can change the language, open and save files, go to Example Programs and Help.

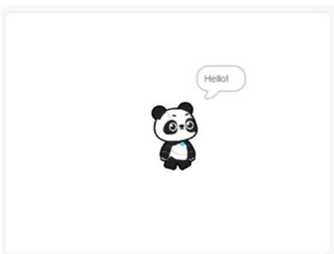
3. Graphical programming

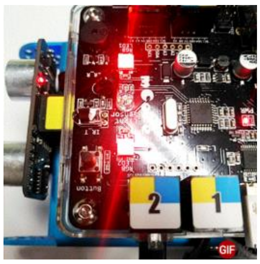
Select the blocks you need from the Blocks area. Left click the block and hold it. Drag the block to the Script area and drop it.



The blocks of different colors and shapes can be connected with each other.

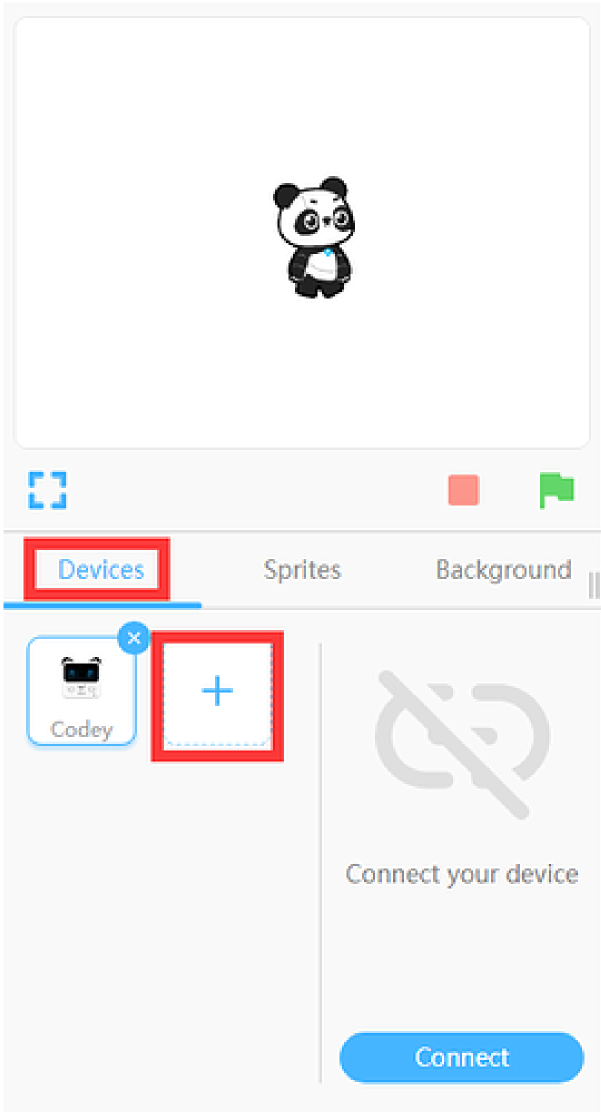
Click the block and you can observe the effects directly in the Stage area or on the hardware.



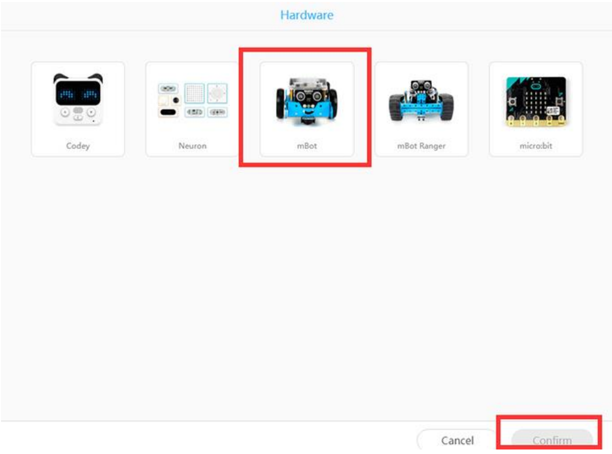


4. Add hardware device

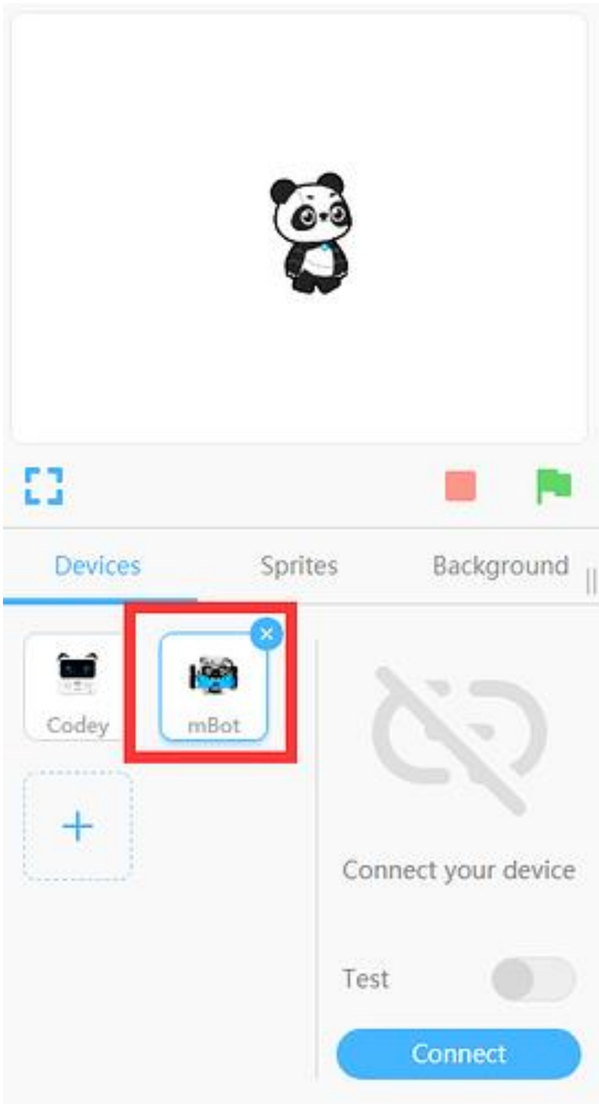
Open mBlock 5 and click the plus button under the Device category



In the Hardware Library, select the Device you need and click Confirm.



Go back to the homepage. Look, the new hardware mBot is here. You can continue to add other devices.

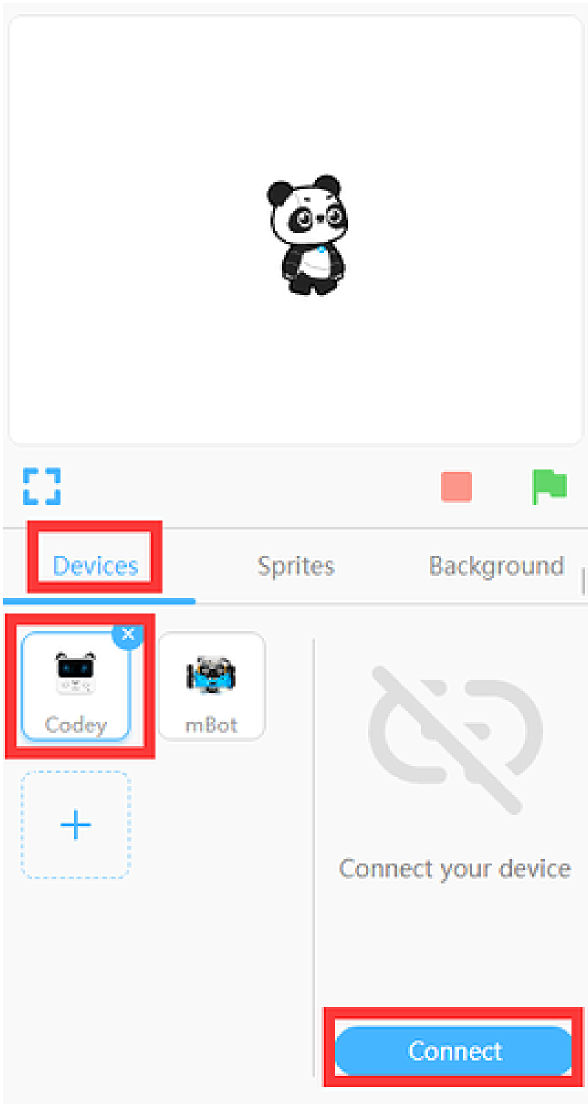


5. Connect Devices and upgrade

Power on your hardware device and connect it to the computer via a USB cable or a Bluetooth dongle.

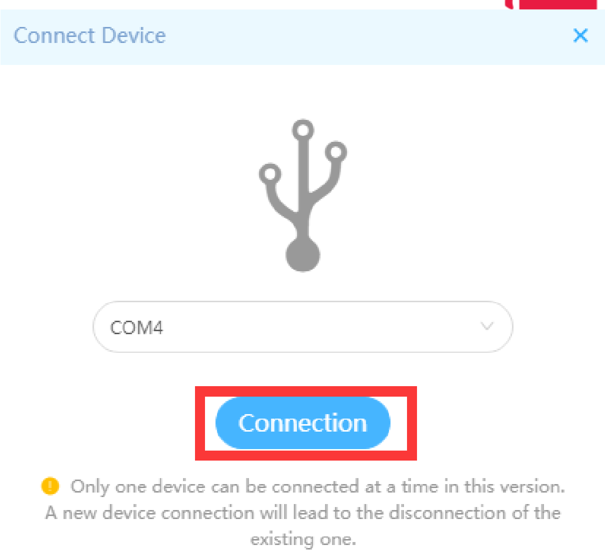
A. Use a USB cable to connect your device

- Connect your hardware to the computer via a USB cable.
- Under the Devices category, select the hardware device you want to connect and click Connect.



Click Connection.

Note: COM 4 is the serial port number and it might be different on another system or PC. On Mac it would be like 1410 or 14230. You can just click Connection.



Return to the homepage. If it shows Device Connected, then it means that the device has been connected to the computer.

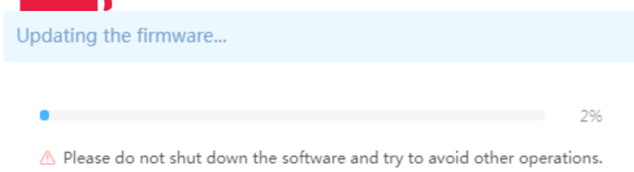
B. Connect via a Bluetooth dongle

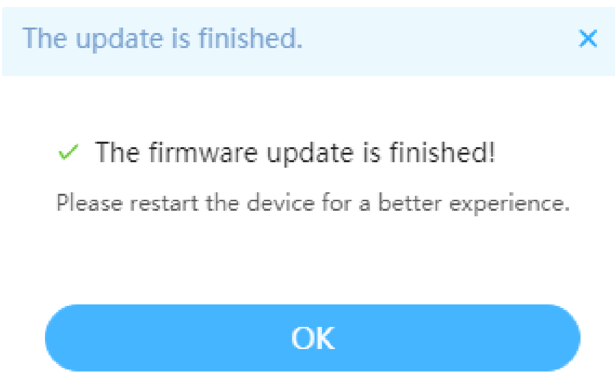
If your computer supports Bluetooth and your hardware device has a Bluetooth module, you can control or program your robot wirelessly.

First, plug the Bluetooth dongle into the USB interface and you will find the dongle flashes blue light. Make sure the device is powered on and placed near the dongle. At this time, the dongle will stop flashing and turn solid blue.

Next, follow steps 2, 3, and 4 in Use a USB cable to connect your device

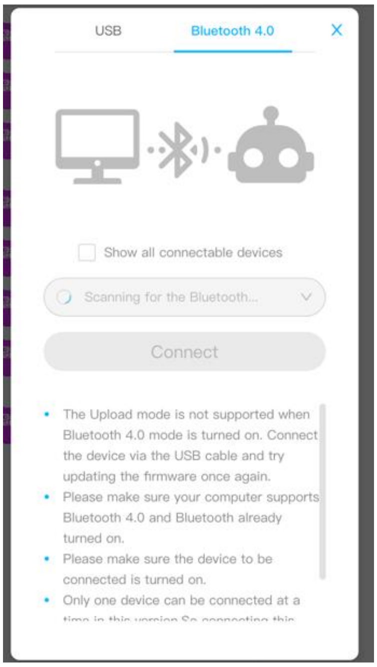
Note: Before you start to control the device with mBlock 5, a window might pop up to tell you to Update Firmware. You can just click Update Now. It will take you 2-3 minutes. Then click OK, and the firmware is upgraded now.





C. Connect via computer Bluetooth

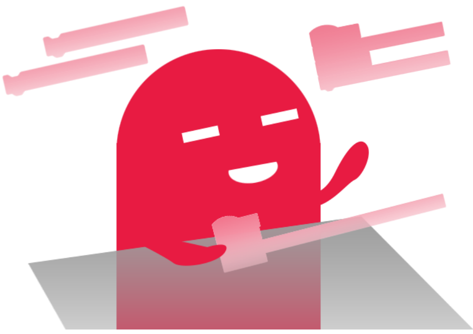
If your computer supports Bluetooth and your hardware device has a Bluetooth module. You can control your robot wirelessly.



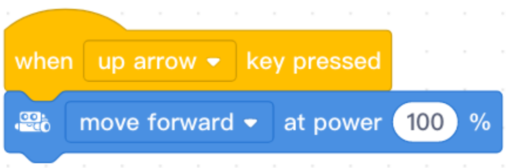
Attention: Program upload is not supported via Bluetooth connection.

Part 3 Programming

The first program

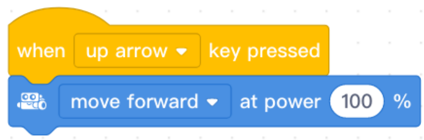


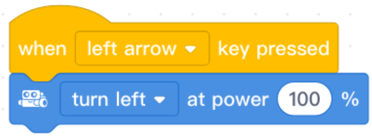
Bluetooth connection between the mBot and computer is recommended for larger classes. With wired connection, any program related to the movement of the mBot may cause the wire to tangle as the program runs immediately after upload.

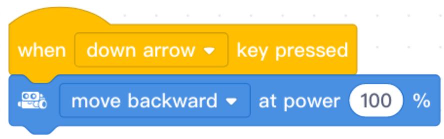


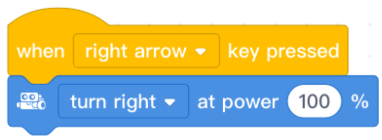
When the “up arrow” on the keyboard is pressed, the mBot will move forward at 100% speed, and it will stop when the up arrow released.

Can the student control the movement of the mBot using the four arrow keys?



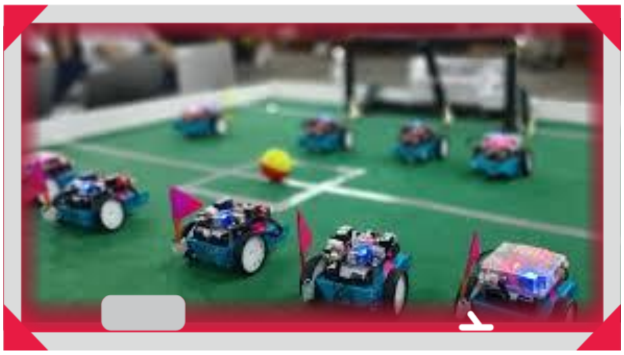






Part 4 Activity

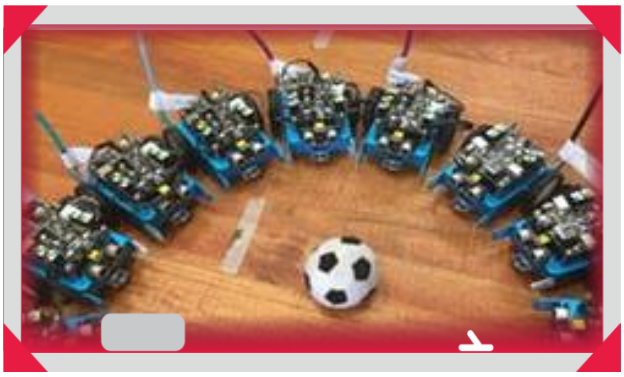
Activity A - Soccer Game



Time: 3 mins per round

Prepare a “soccer field” using any material you could find in the classroom.

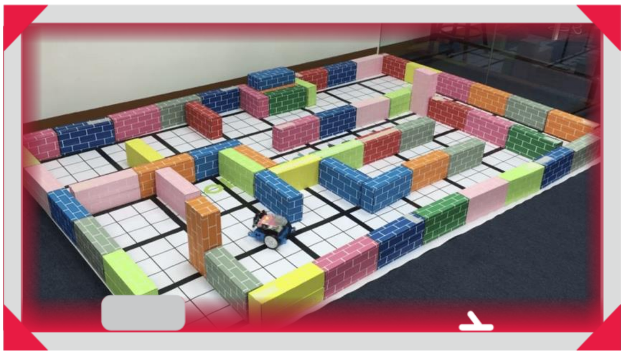
Activity B - Sumo



Time: 3 mins per round

Establish a border ring by drawing or using tape. Player aims to push the other mBots out of the circle using his or her own. The last mBot inside the circle wins the game.

Activity C - Maze



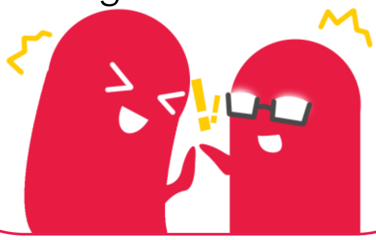
Time: 3 mins per round

Players control the mBot to go from the entrance to the exist as quickly as possible. The player who finishes the maze in least amount of time wins the game.

Part 5 Reflection

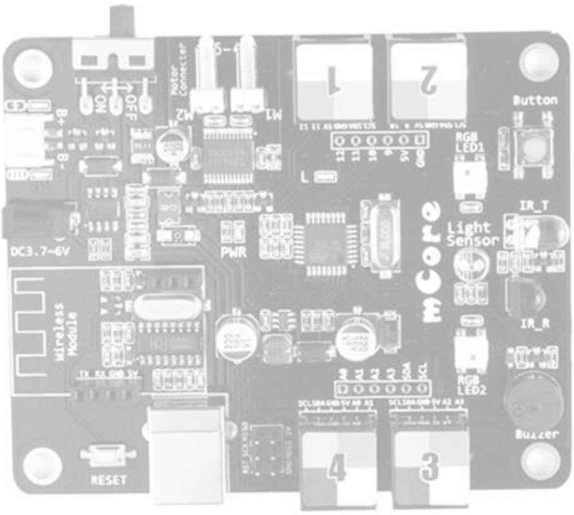
How did you feel when you finished assembling your first mBot and writing its first program?

What kinds of structural improvements could you apply to the mBot for better performance during the activities?



What do you need to do in order to upload your program to the mBot? Why is upgrading the firmware necessary?

Makeblock Sensors



02

Teaching Suggestion

Lesson 2 Makeblock Sensors

Overview
20 mins

- Introduce the lesson by asking the question: What are sensors? - 5 mins
- Introduce some common sensors and electronics - 10 mins
- Introduce the function of sensors on mCore. - 5 mins

Sensor Usage
25 mins

- Introduce I/O System to students by example. - 15 mins
- How to use a Makeblock sensor with the mBot? - 10 mins

Programming
50 mins

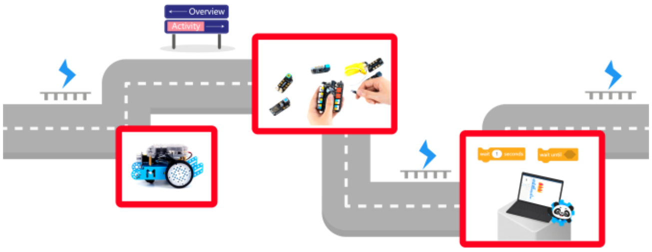
- Introduce the Programming Examples of Buzzer\ LEDs\ Light Sensor\ Ultra-sonic Sensor. - 40 mins
- Students review knowledge and ask questions. - 10 mins

Key Programming Topics
25 mins

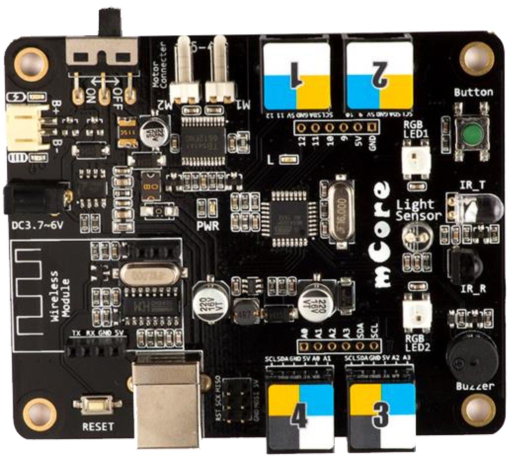
- Open mBlock5 and Introduce some commonly used blocks. - 10 mins
- Wait Blocks and Wait Until Blocks - 5 mins
- Forever Blocks and Repeat Until Blocks - 5 mins
- If-then Blocks and If-else Blocks - 5 mins

Reflection
5 mins

Part 1 Overview



In this lesson, we will cover several electronics and sensors that are either integrated with the mCore or come with the competition package. This includes – onboard buzzer, onboard LEDs, onboard light sensor, and ultra-sonic sensor.



Objectives

What are sensors?

Sensors are specialized electronics that detect and/or respond to various environmental properties such as light, sound, temperature, humidity,

Common sensors and electronics

The most commonly seen sensors and electronics used in level one competitions are line followers, color sensors, buzzers, mp3 modules, Bluetooth modules, LED boards, LED lights, ultra-sonic sensors, and servos.

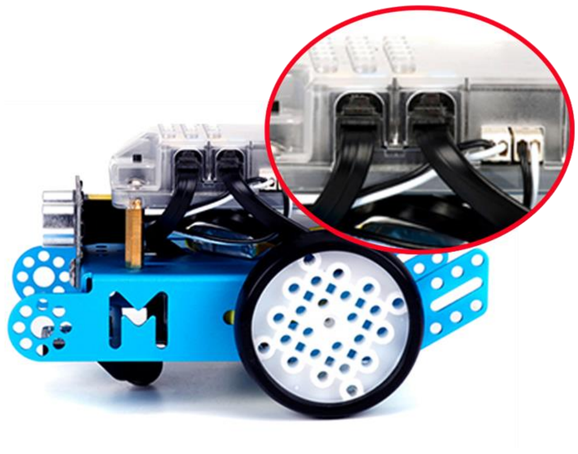
Why sensors?

Just like humans use their five major senses to gather information about the surroundings and the world in order to explore, navigate and make logical and productive decisions, sensors provide machineries with useful data for processing and in turn become smarter and more capable devices.

Part 2 Sensor Usage

I/O System
Inputs and Outputs

Before using a sensor, it is essential to first understand its I/O system, namely its inputs and outputs. The input refers to what the sensor is designed to sense or detect. And the output refers to how the gathered information or data is represented in programming environments or presented to the real world. Finally, it is up to the programmers to decide how to utilize this information in programs to better accomplish various tasks. Some sensors, however, require a few extra steps for setup. This will be covered in a sensor specific context.



How to ues a Makeblock sensor with the mBot?

Using a Makeblock sensor with the mBot is a straight forwards process which follows a simple color-coded rule – any sensor could be used with any RJ25 port on the mCore as long as the port and the sensor has a pair of matching color stickers.

Part 3 Programming

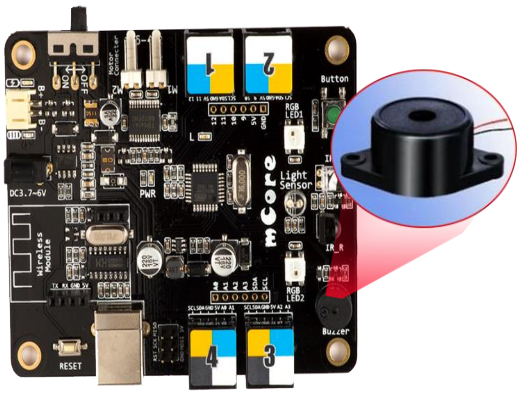
Buzzer

Buzzers are not sensors, therefore do not detect or respond to any environmental properties. However, they do respond to electronic signals sent by programming instructions.

Input

Output

Buzz at a specified frequency for a specified amount of time.



Programming Example

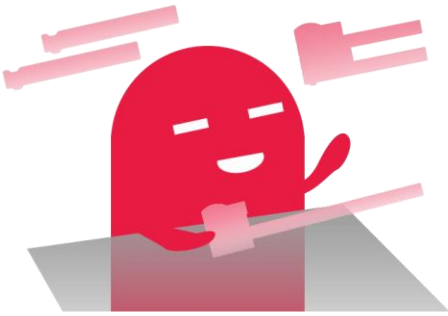
1. Buzz at 700Hz for 1 second

```
play sound at frequency of (700) Hz for (1) secs
```

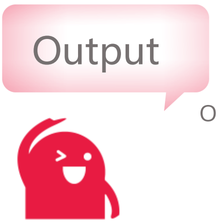
2. Play a song of lullaby

```
play note E4 for 0.25 beats
play note E4 for 0.25 beats
play note G4 for 0.5 beats
wait 0.5 seconds
play note E4 for 0.25 beats
play note E4 for 0.25 beats
play note G4 for 0.5 beats
wait 0.5 seconds
play note E4 for 0.25 beats
play note G4 for 0.25 beats
play note C5 for 0.5 beats
play note B4 for 0.5 beats
play note A4 for 0.5 beats
play note A4 for 0.5 beats
play note G4 for 0.5 beats
wait 0.1 seconds
play note D4 for 0.25 beats
play note E4 for 0.25 beats
play note F4 for 0.5 beats
wait 0.5 seconds
play note D4 for 0.25 beats
play note E4 for 0.25 beats
play note F4 for 0.5 beats
wait 0.5 seconds
play note D4 for 0.25 beats
play note E4 for 0.25 beats
play note F4 for 0.5 beats
play note E4 for 0.5 beats
play note D4 for 0.5 beats
play note C4 for 0.5 beats
```

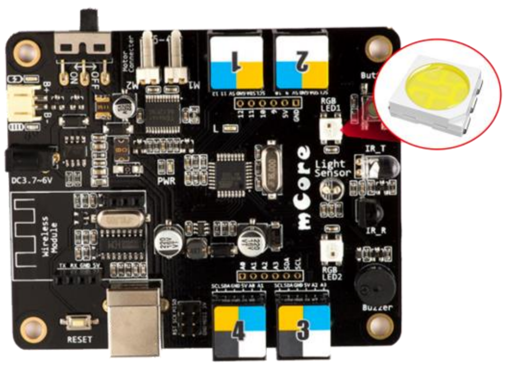
Buzzer



There are two LED bulbs on the mCore. Since they are not sensors, they do not detect or respond to any environmental properties. However, they do respond to electronic signals sent by programming instructions.

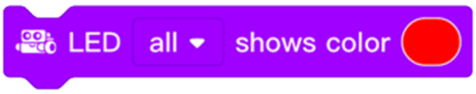


Each LED lights up in the specified color represented in either RGB values or built-in color profiles for either a specified amount of time or forever.

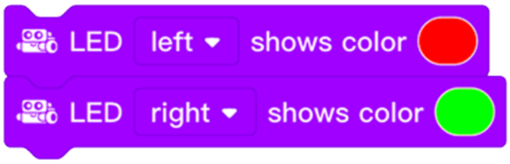


Programming Example

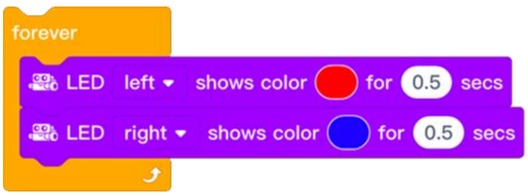
1. Set both LEDs to red forever



2. Set left LED to red and right LED to green



3. Ambulance effect



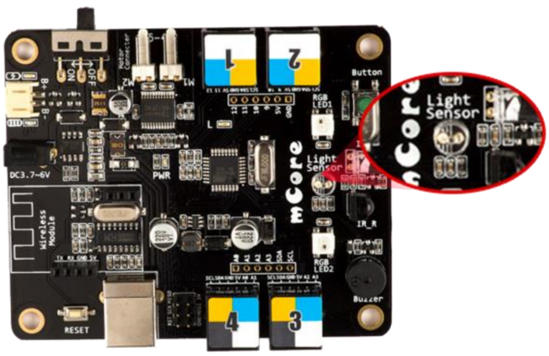
Light Sensor

Input

Light sensors detect the intensity of the ambient light. It is important to note that onboard LEDs may affect the values detected by the onboard light sensor.

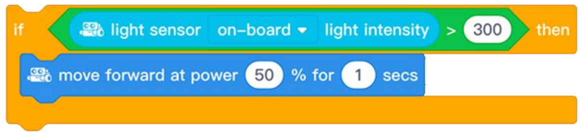
Output

The onboard light sensor returns a non-negative integer representing the intensity of the ambient light. A higher value corresponds to a brighter environment while a lower value corresponds to a darker environment.



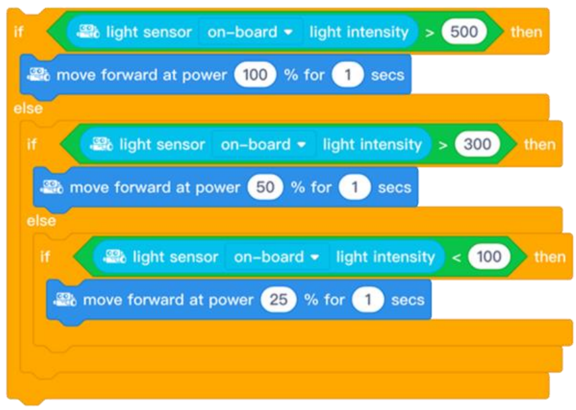
Programming Example

1. 50% power when light



2. 50% power when light intensity larger than 300

3. 100% power when light intensity larger than 500, 50% power when light intensity between 300 and 500, and 25% power when light intensity less than 100



Ultra-sonic Sensor

Input

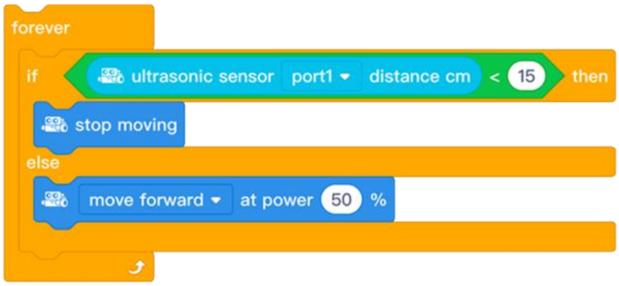
Ultra-sonic sensors broadcast directional high frequency sound waves and detect their reflection timing.

Output

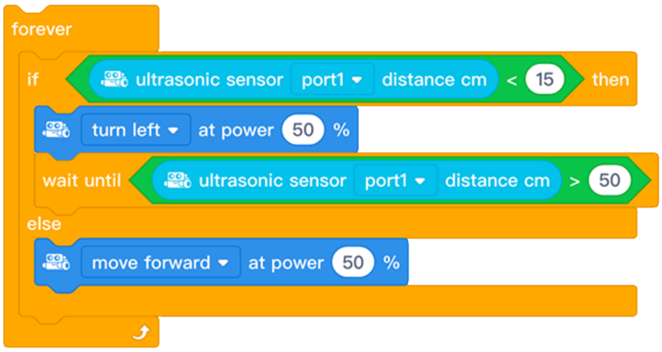
The sensors return a non-negative number representing the distance to the closet object in centimeters.

Programming Example

1. Move forward at 50% power or stop whenever an obstacle is within 15cm perimeter

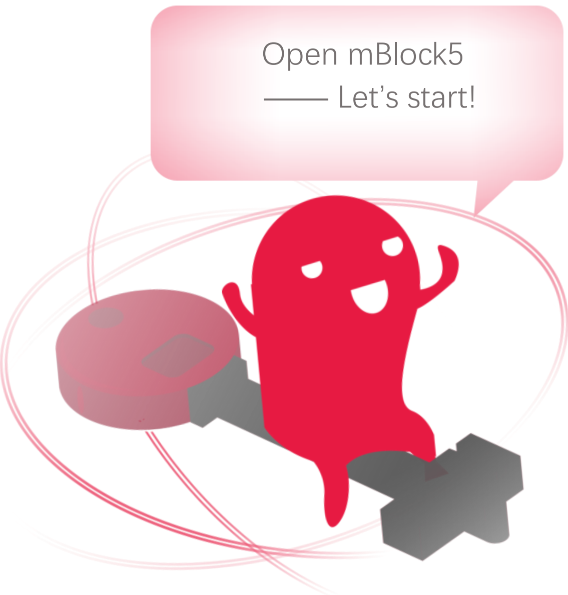


2. Automatic obstacle avoidance



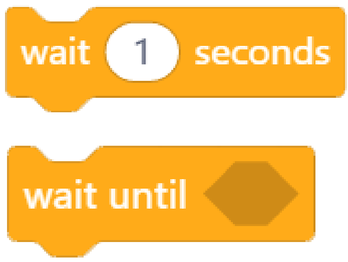
Procedural programming and execution flow

mBlock5 uses block stacking to help students develop essential skills for procedural programming. One of the most important concepts students should undertstand is the execution flow.



mBlock programs are executed in a top-down manner where each block is precisely executed for the specified amount of time or until a specified condition is met before the next block is executed.

Waits

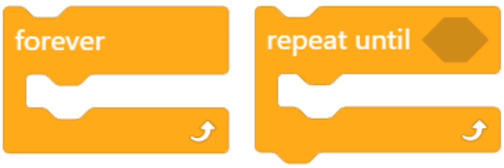


Wait blocks are the simplest and easiest to understand form of control blocks. Wait blocks, as their name suggest, halts the execution of the program for the specified number of seconds. It is important to note that wait blocks do not halt the program, but the execution flow of the program.

Any block that comes before the wait block, if there is one, still maintains its effect.

Instead of specifying the number of seconds, wait until blocks let users specify the condition that must be met before the execution halt is lifted.

Loops

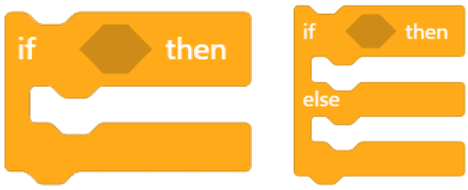


Loops are major building blocks for more complex programs. Loops help programmers avoid repeating code and achieve task automation. Forever loop is one of the most common loops used by programmers. As its name suggests, forever loops have no end condition and therefore all the blocks residing inside forever loop are executed one round after another. Due to the nature of forever loops, only one forever loop can appear in a program.

Repeat until blocks could be understood as a forever loop with an escape condition. Repeat until blocks behave just like forever blocks before the specified condition is met.

It is important for students to realize loops introduce a new shape to the execution flow. Before loops, the execution flows straight from top to bottom while loops can force the execution flow to go in a circular motion.

If statements

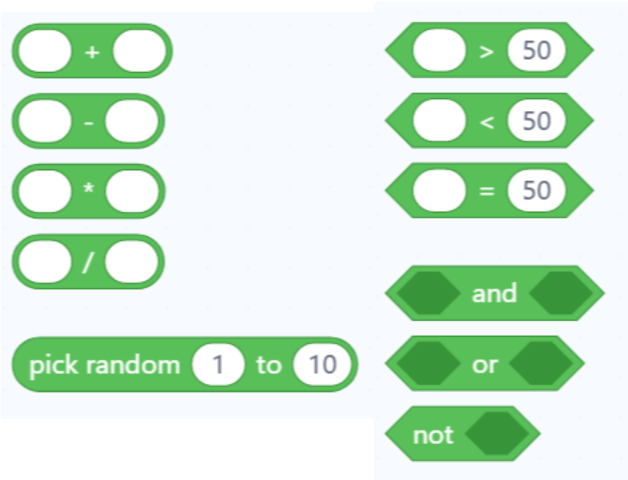


Just like loops, if statements are essential to complex programs. If statements allow programs to behave differently under various circumstances. The simplest and easiest to understand form of if statements is the if-then block. If-then blocks usually present no challenge for the students to understand since it appears in natural language all the time.

Although if-else statements do not present any more logical challenge for students, the syntax and bracketing sometimes confuse students.

The real challenge lies with nested if statements. Combined with operators, they require meticulous planning and ordering to avoid bugs and errors.

Operators



The operators' category consists mostly common mathematical symbols and logics such as comparing numbers, calculations, random number generation, and logic gates.

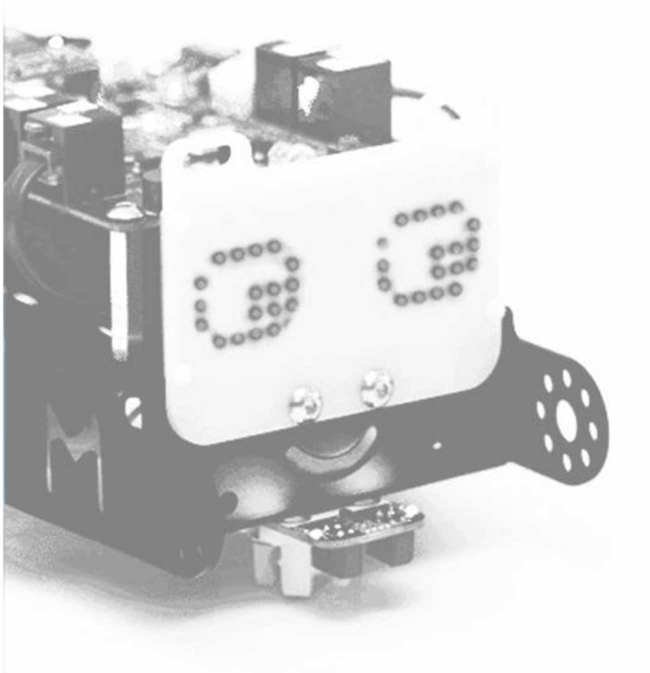
Part 5 Reflection

What are sensors?

What is input? What is output?

How to use block stacking for procedural programming?

Me LED Matrix



03

Teaching Suggestion

Lesson 3 Me LED Matrix

Overview 15 mins

- Introduce the lesson by asking questions or telling interesting stories. - 10 mins
- Introduce the content and learning objectives of this lesson. - 5 mins

Inquiry 30 mins

- Module instruction - 15 mins
- Students draw their own plans for using the Me LED Matrix and discuss with other students. - 10 mins
- Connection Method: how to use a Me LED Matrix with the mBot? - 5 mins

Programming 55 mins

- Show the mBlock Programming function list to students. - 10 mins
- Explain the event executive examples to students. - 45 mins

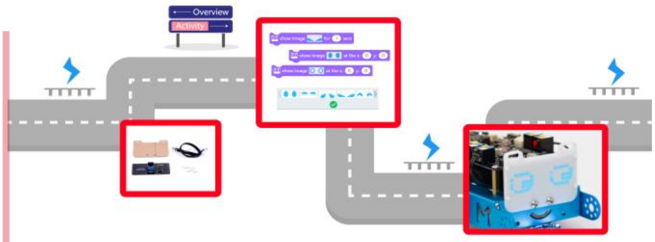
Key Programming Topics 15 mins

- Introduce variable and explain how to make it in mBlock 5. - 5 mins
- Introduce "Set ( ) to ( )" and "Change ( ) by ( )" Blocks. - 5 mins
- Introduce axis. - 5 mins

Reflection 5 mins

Part 1 Overview

In this lesson, students can not only understand the basic parameters and usage characteristics of the Me LED Matrix, but also learn how to program the LED matrix to display the output values of other sensors. In the end, under the guidance of the instructor, combined with the knowledge of 2D coordinate system, students will attempt the ultimate task of this lesson.



Objectives

I can understand the basic parameters of the Me LED Matrix and its functional characteristics.

I can understand the different functionalities of the blocks related to the Me LED Matrix.

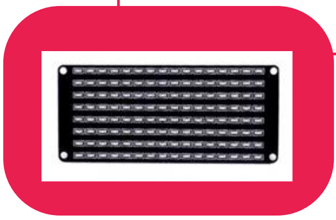
I can program the LED Matrix to display numbers, texts, or symbols.

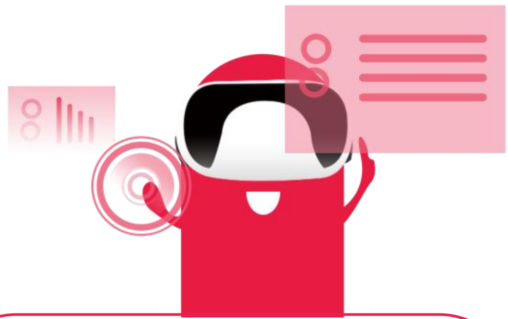
I can gradually master the programming control skills of Me LED Matrix through step-by-step learning and continuous experimentation.

Part 2 Inquiry

Module Instruction

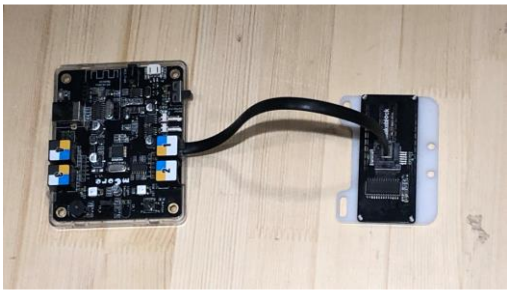
The Me LED Matrix (8 x 16) contains a total of 128 blue LEDs.





The LED Matrix could be programmed to display numbers, texts, and symbols. The blue sticker on the connection port indicates the LED Matrix could only be connected to a port with blue sticker on the main board.

Connection Method

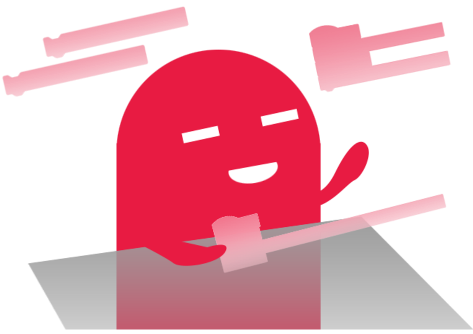


Connecting with RJ25
Since the port of LED Matrix (8 x 16) has a blue sticker, you need to connect it to a port with blue sticker on the mCore using a RJ25 cable.

Application

The LED Matrix is a great tool for outputting information. It could either be used to inspire students with creative projects by displaying text, patterns, and symbols or showing sensor outputs that are otherwise difficult to obtain or visualize.

Part 3 Programming



It's usually very useful to have a display to show various information of the robot. However, actual displays are often more difficult to program and consumes a lot more energy than the LED Matrix. Despite its very limited resolution and feature, young students can easily get used to its controls and put their creativity at work.

mBlock Programming function list

1.Display customized images for 1 sec and off

LED panel port1 ▾ shows image for 1 secs

2.Display customized images

LED panel port1 ▾ shows image

3.Display customized images with x&y coordinate

LED panel port1 ▾ shows image at x: 0 y: 0

4. Display string/characters

LED panel port1 ▾ shows text hello

5. Display string/characters with x&y

LED panel port1 ▾ shows text hello at x: 0 y: 0

6. Display numbers

LED panel port1 ▾ shows number 2048

7. Display the time

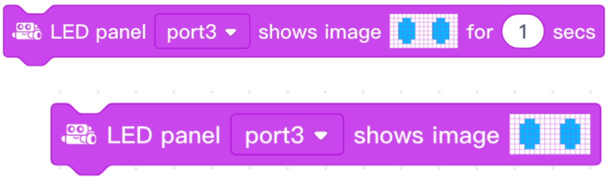
LED panel port1 ▾ shows time 12 : 0

8. Screen clearing function

LED panel port1 ▾ clears screen

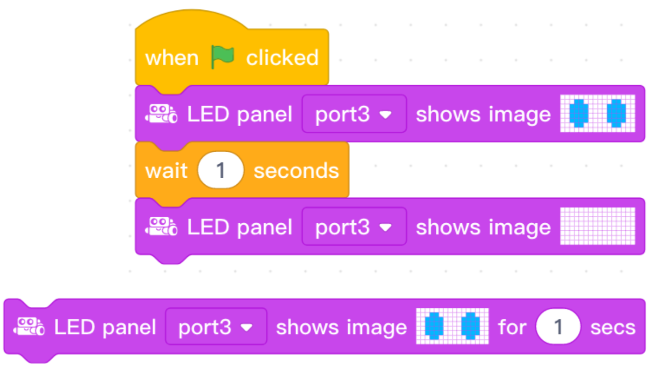
What is the difference between the effects of executing the two blocks below?

Students can drag out the following two blocks, double click on them and observe their different effects.



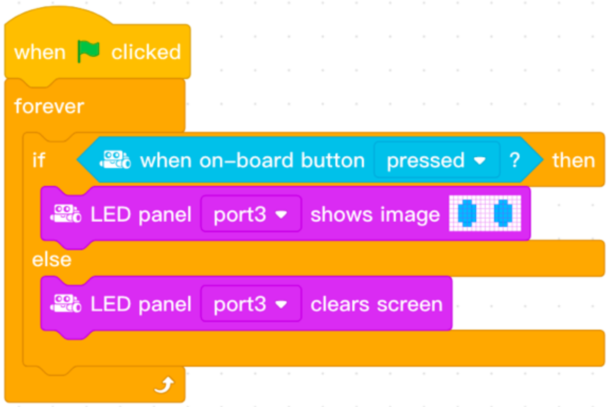
Do the following two pieces of code have the same effect on the LED Matrix?

Decomposing a single complex block into multiple simpler blocks is a good practice for more complex exercises in the future.



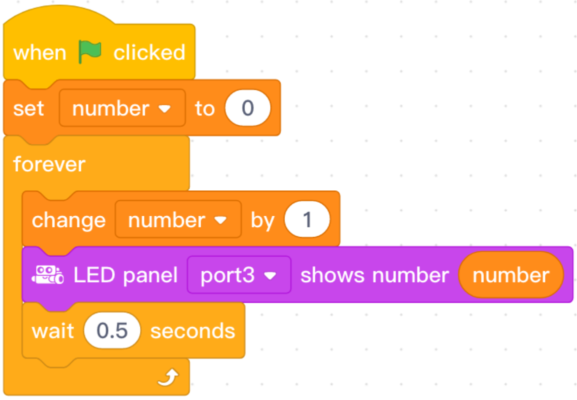
How could the on-board button be used to switch patterns being displayed on the LED Matrix?

Use the "if...then...else" conditional statement to switch between two custom patterns.



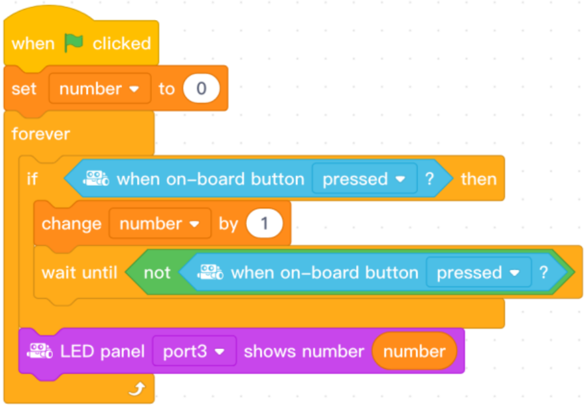
How do you write a program to display incrementing numbers on the LED Matrix?

The Change () by () block is a Variables block and a Stack block. The block will change the specified variable by a given amount. Along with a 'wait' block, the numbers can be displayed on the LED Matrix in a controlled manner.



Is there a way to achieve the same effect of displaying incremental numbers manually using the on-board button?

Students can try to use other methods to achieve the same functionality as the sample program. In the process, they can constantly expand the understanding of the basic programming logic.

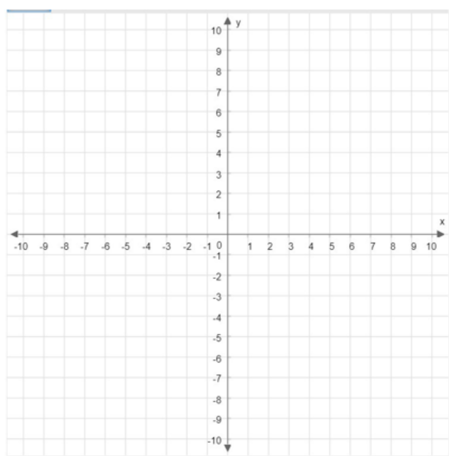


How to use the Me LED Matrix to monitor the sensor outputs?

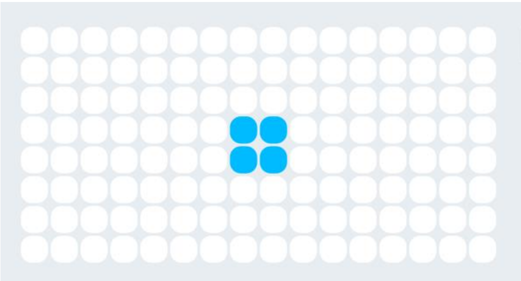
Being able to display sensor readings on the LED Matrix is essential to understanding certain program behaviors during execution so that fixing the program becomes an easier task.

How can I make the pattern displayed on the LED Matrix move?

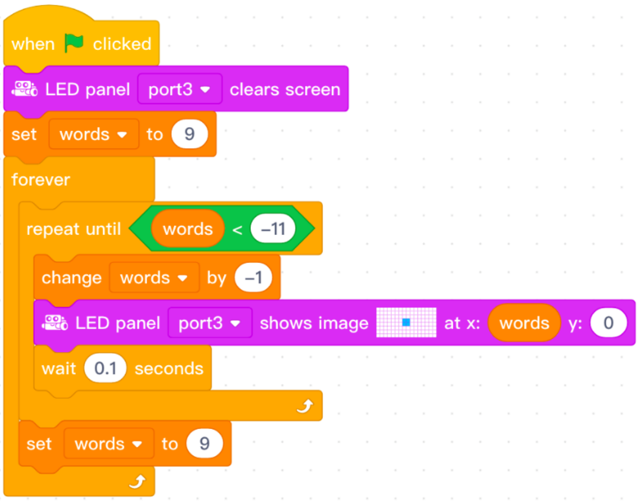
What is the xy coordinate system?



What is the orientation of the xy coordinate system on the LED Matrix?



How to calculate the number of steps the image has moved on the LED Matrix?

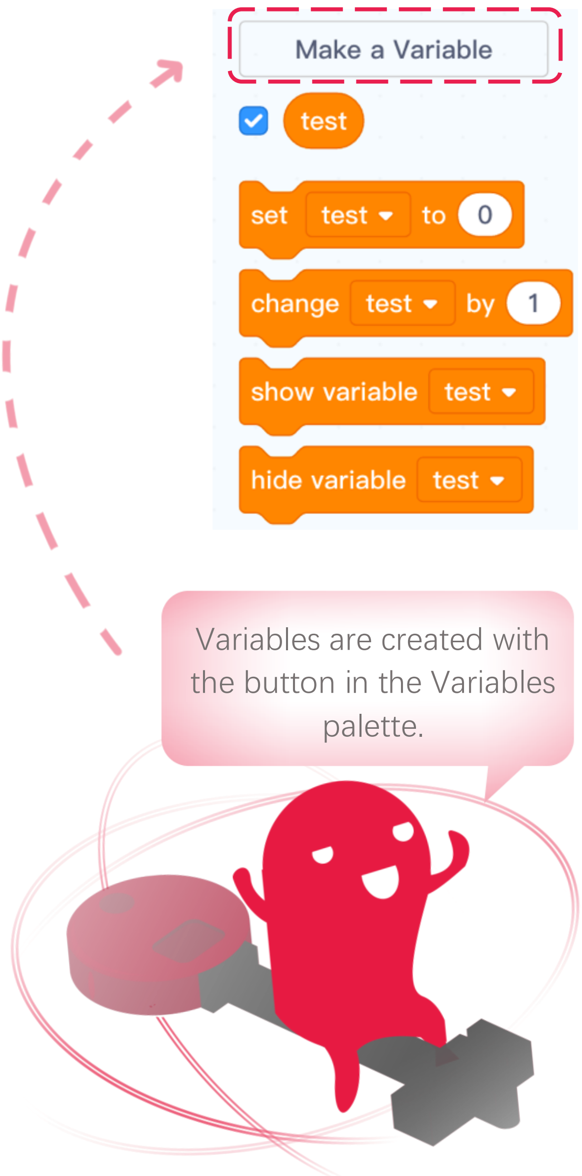


How to display moving text on the Me LED Matrix?

Determine the movement effect of the text on the Me LED Matrix, and use the image movement method you learned before to try to complete the moving text.

Variable

A variable is a changeable value recorded in mBlock 5 memory.



Variables can only hold one value at a time, unlike lists. These values can be either numbers or strings — any text. A small bubble showing the current value of the variable will appear when the variable is clicked on. Unlike many other programming languages, variables must be created prior to when the project actually runs. This only results in a small amount of RAM being used to store the value for use when the project actually runs.

Set ( ) to ( )

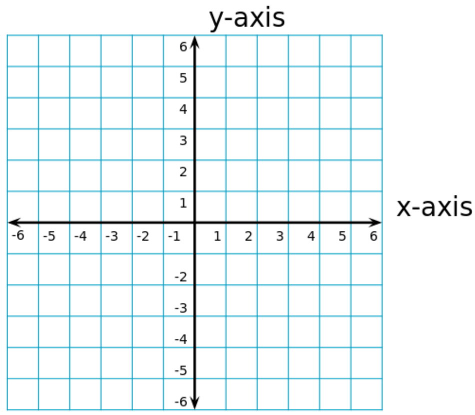
The Set ( ) to ( ) block is a Variables block and a Stack block. The block will set the specified variable to the given value: a number.

Change ( ) by ( )

The Change ( ) by ( ) block is a Variables block and a Stack block. The block will change the specified variable by a given amount.

Axis

Coordinate graphing sounds very dramatic but it is actually just a visual method for showing relationships between numbers. The relationships are shown on a coordinate grid. A coordinate grid has two perpendicular lines, or axes, labeled like number lines. The horizontal axis is called the x-axis. The vertical axis is called the y-axis. The point where the x-axis and y-axis intersect is called the origin.



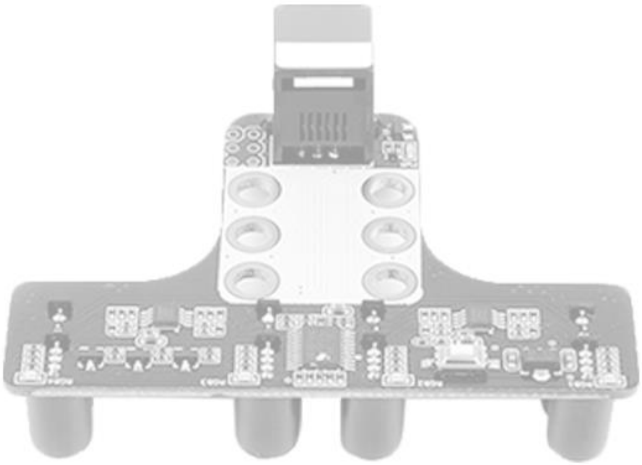
Part 5 Reflection

What information can be displayed on the Me LED Matrix?

How to display sensor output on the LED Matrix?

What is the xy coordinate system? How are the axes on the Me LED Matrix oriented?

RGB Line Follower



04

Teaching Suggestion

Lesson 4 RGB Line Follower

Overview
15 mins

Introduce the lesson by asking questions or telling interesting stories.
- 10 mins

Show the content and learning objectives of this lesson.
- 5 mins

Inquiry
30 mins

Hardware introduction
- 10 mins

Explain the concept of the fill lights separately by asking questions: Can we switch the fill lights on the RGB line follower to green, red, or blue?
- 10 mins

Connection method: How to use an RGB line follower with the mBot?
- 5 mins

Learning method: how dose the learning method work?
- 5 mins

Programming
55 mins

Before programming，let the students download the RGB line follower extension from the extension center.
- 5 mins

Explain the programming examples to students.
- 45 mins

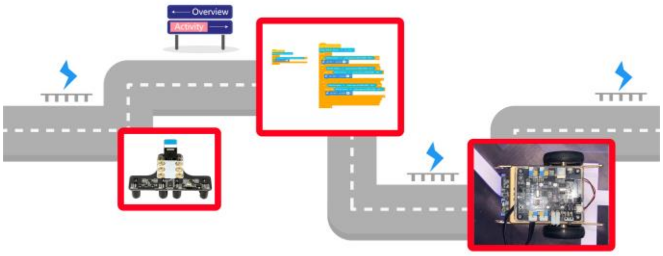
Key Programming Topics
15 mins

Give an example of what is binary number and find out the difference between binary and decimal.
- 5 mins

Discover nested if statements from practical applications and summarize its complicated use.
- 10 mins

Reflection
5 mins

Part 1 Overview



Line following is the heart of level one automatic stage. Being able to reliably follow tracks on the map is a prerequisite for getting to desired locations and completing the required missions. RGB line follower is a powerful sensor that can provide contestants with reliable and customizable information regarding the map, which can in turned be used for completing various tasks.

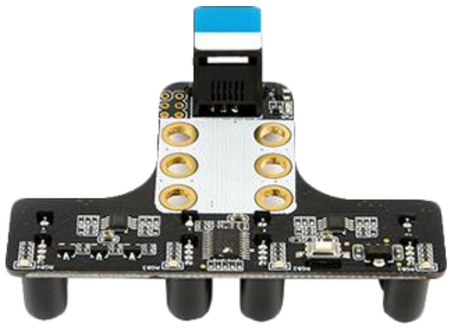
Objectives

Students can properly use the RGB line follower to learn the track and background color.

Students can properly read the RGB line follower status

Students can use motor differential speed to follow a track

Students can use sensor status to program a basic line following logic.

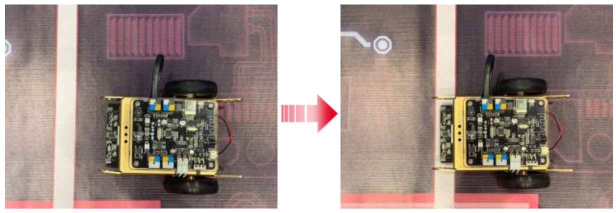


Part 2 Inquiry

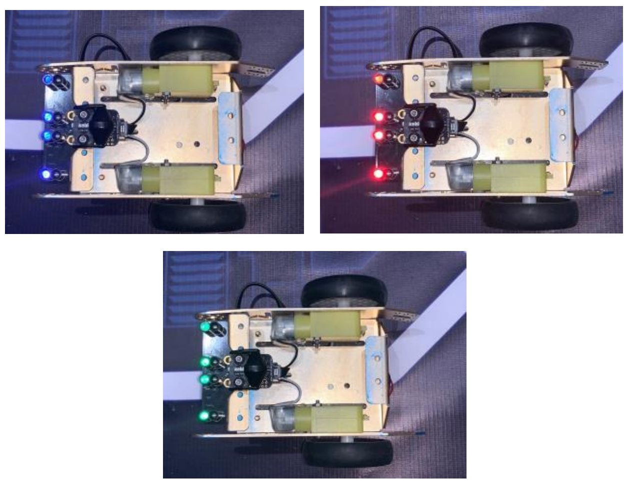
Hardware

RGB line followers carry four independent sensors that are capable of learning and memorizing the color of the track and the color of the background.

To learn the background color, place your mBot so that all four sensors are seeing the background color and press the small white onboard button once. The indicator lights will start flashing indicating the learning has begun. Once the learning is complete, the lights will stop flashing. To learn the track color, simply have all four sensors on the track and double click the button.

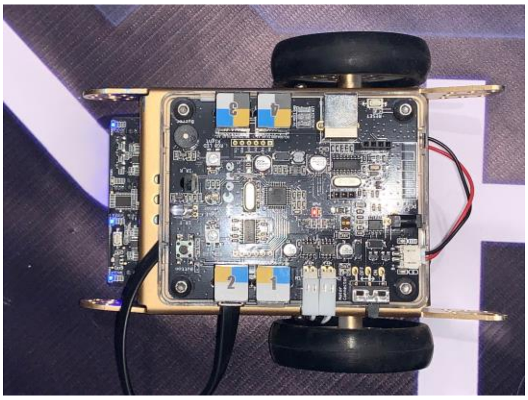


- Along with the four sensors are four fill lights that could be set to red, green, or blue. It is up to the contestants to decide which color of fill light is most fitting for the map and helps gather the most reliable stream of data. To change the color of the fill lights, simply press and hold the button for approximately two seconds.



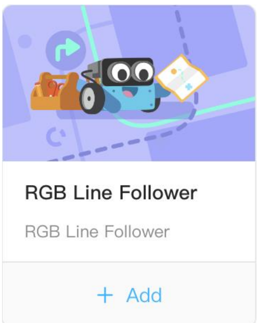
- Once the background and track colors have been learnt and have been set, each of the four indicator lights indicates the status of each of the four sensors. An indicator that is on suggests the corresponding sensor sees the background.

And an indicator that is off suggests the sensor sees the track. When reading the status of the line follower, an on indicator reads as a '1' and an off indicator reads as a '0'. Since the first sensor resides on the rightmost side of the line follower, the proper way of reading the line follower status goes from right to left. For instance, the following status reads as '1011'



Software

Before using RGB line followers in mBlock 5, students must first download the RGB line follower extension from the extension center.



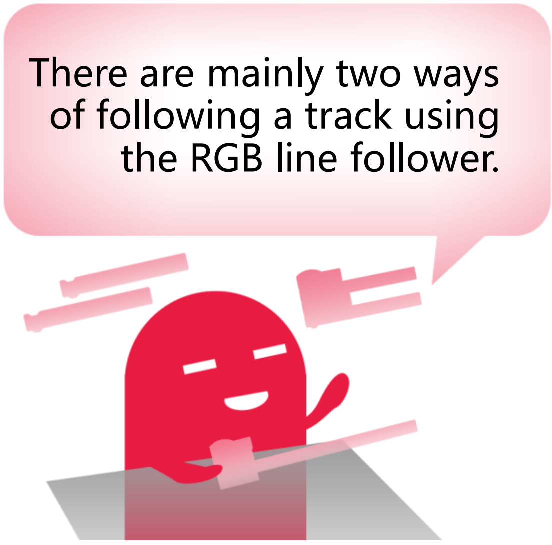
Unlike mBlock3, mBlock5 requires students to first initialize the RGB line followers connected to the mBot. Since the mCore supports a maximum number of four-line followers being connected at the same time, each line follower is, after initialization, referred to as line follower 1, 2, 3, and 4 without having to specify the ports.

A new feature is that students can now set the fill light color through software. The biggest advantage this brings is the ability to change the fill light color during program execution so that the mBot could better adapt to various map features.

The other related blocks include retrieving or asking questions about the status of each sensor, setting sensitivity, and retrieving the motor differential speed. These blocks will be covered in programming examples.

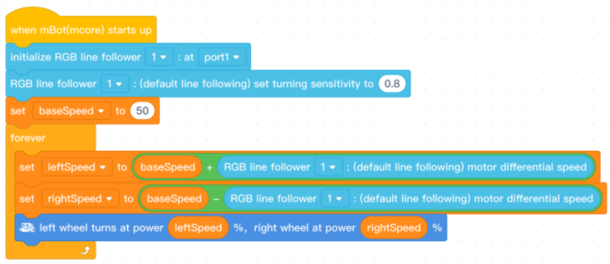
It is important to note that RGB line follower blocks are only available in upload mode.

Part 3 Programming Control



First is by using the built-in motor differential speed and turning sensitivity.

This method is relatively easy to set up and use without the need to understand its working principles. However, since its working principle is not apparent, the degree of control and customization is limited. Students mostly rely on trial and error to figure out the optimal parameter values.

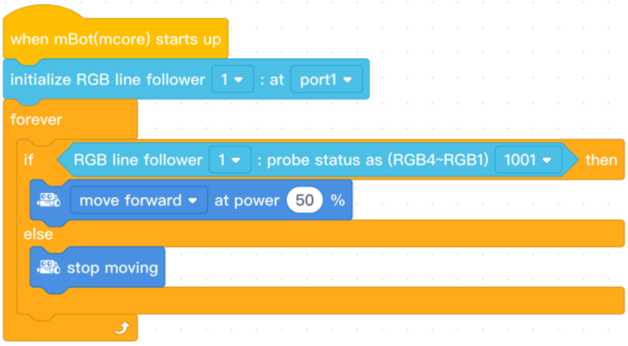


The **second** method is to custom build your own line following logic using sensor outputs.

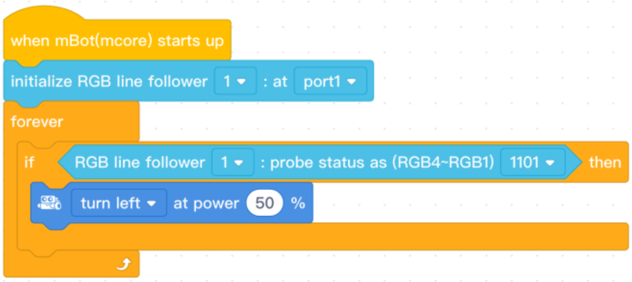
Compared to the first method, this method requires students to be comfortable working with sensor status, if statements, and sometimes loops. Since this method does not rely on any hidden logic, students have a much higher degree of control and easier time of debugging.

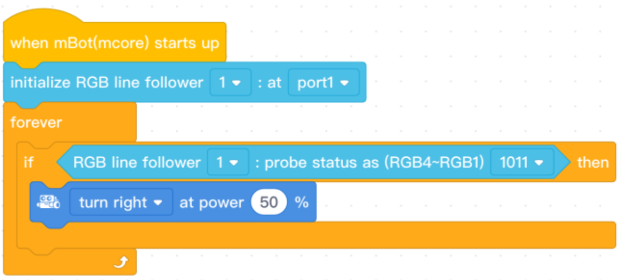
The first step to building your own line following logic is establishing a clear goal and figuring out what information from the line follower can help you achieve that goal. Assuming the goal is to make the mBot go forward whenever it is straight on the track or stop otherwise, the real challenge is deciding what counts as being straight on the track.

For the purpose of this example, the line follower status, when the mBot is straight on the line reads as '1001' hence the program below.

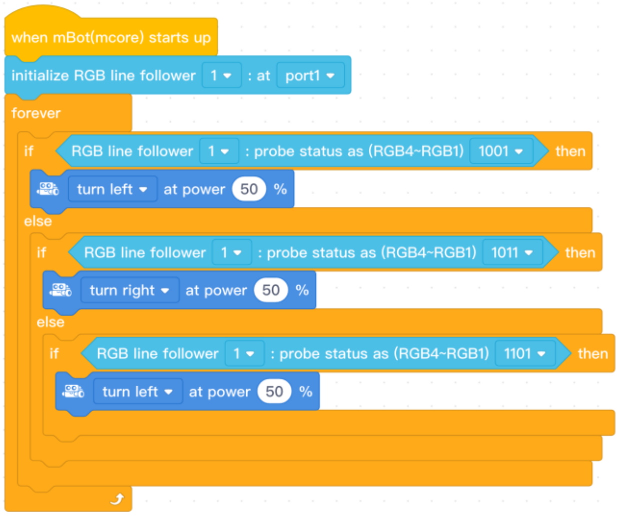


After some testing, it comes naturally that the next step is to make the mBot turn with the track. For this objective, the key is to decide the conditions under which the mBot goes left or right. Assuming the mBot starts straight on the track, the very first status indicating it's going off track is either '1101' or '1011'. '1101' Indicates the mBot is off to the right. And to go back on track it should turn left and vice versa.





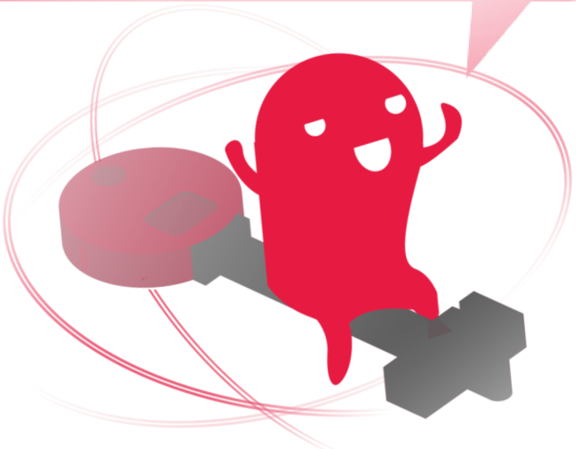
Now comes the question – how should one logically combine the three conditions above and have their mBot automatically follow a simple track? Ask students to test and compare the following two programs. Do they both work? Which one is more elegant? And what is redundant about the other one?



Binary Number

"0" & "1"

In mathematics and digital electronics, a binary number is a number expressed in the base-2 numeral system or binary numeral system, which uses only two symbols: typically "0" and "1".

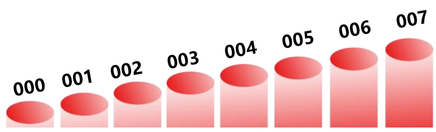


Binary numbers are important because of its straightforward implementation in digital electronic circuitry using logic gates. The binary system is used by almost all modern computers and computer-based devices including RGB line followers to represent their sensor status.

Decimal counting uses the ten symbols 0 through 9. Counting begins with the incremental substitution of the least significant digit (rightmost digit) which is often called the first digit. When the available symbols for this position are exhausted, the least significant digit is reset to 0, and the next digit of higher significance (one position to the left) is incremented (overflow), and incremental substitution of the low-order digit resumes.

Decimal number	Binary number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Decimal increment



Rightmost digit is reset to zero, and the digit to its left is incremented.

Binary increment

Binary counting follows the same procedure, except that only the two symbols 0 and 1 are available. Thus, after a digit reaches 1 in binary, an increment resets it to 0 but also causes an increment of the next digit to the left:

0000 → 0001

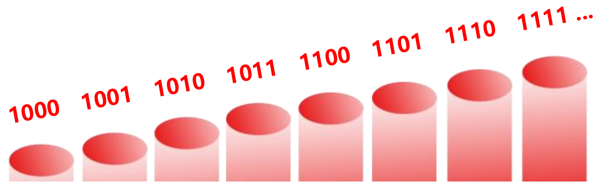
rightmost digit starts over, and next digit is incremented.

0010 → 0011

rightmost two digits start over, and next digit is incremented.

0100, 0101, 0110, 0111

rightmost three digits start over, and the next digit is incremented.



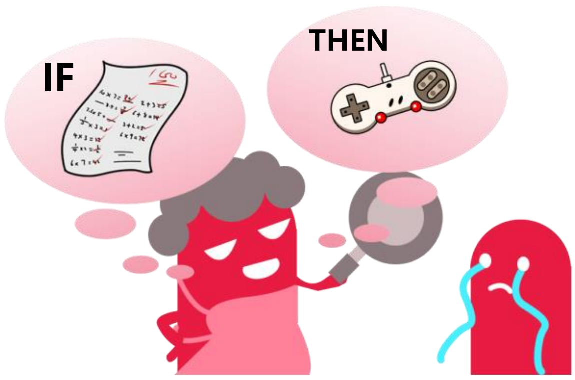
An observant student would soon realize that the maximum number of status a binary number can represent is equal to two to the power of the number of digits.

For instance, RGB line follower uses a four-digit binary number which can represent a total of 24 = 16 states.

Nested If Statements

It is not unusual for students to get intimated or confused by the syntax or the concept of nesting a few if statements together. Imagine the following scenario – Tony's mother tells him the only way he can play games is to get full score on his math exam. Translated to a simple if statement, this becomes——

If (Tony gets full score on math exam)
Then
(Tony plays games)



But being a strict mother, she later adds that if he doesn't get a perfect score on math exam, he must get a perfect score on his English exam otherwise he gets grounded for a week. What would the if statement look like now?

If (Tony gets full score on math exam)
Then
  (Tony plays games)
Else
  If (Tony gets full score on English exam) Then
    (Tony is not grounded)
  Else
    (Tony is grounded for a week)

To summarize, nested if statements, in general, follow the following template:

```
If (CONDITION #1) Then
        DO THIS
    Else / CONDITION #1 is not met
        If (CONDITION #2) Then
            DO THIS
        Else / Neither CONDITION #1 nor #2 is met
            If (CONDITION #3) Then
                DO THIS
            Else / None of the conditions above is met
                Do THIS
```

It is important to note that

more than often nested if statements contain implicit ordering logic. Going back to Tony's example, does the program still do what his mother intends to do if condition #1 and #2 switch places?

Part 5 Reflection

What's most confusing about nested if statements to you?

In what cases do the RGB indicator lights turn on and off?

How many track states can the RGB line follower recognize?

Color Sensor

05

Teaching Suggestion

Lesson 5 Color Sensor

Overview
15 mins

Introduce the lesson by asking questions: Is the world in the eyes of mBots colorful? Do you think they can recognize colors?
- 10 mins

→ Show the content and learning objectives of this lesson.
- 5 mins

Inquiry
25 mins

Hardware introduction
- 10 mins

→ Two different pictures to comparing and guide the students to think about which field of view looks like a mBot with the color sensor ?
- 10 mins

→ Connection Method: How to use a color sensor with the mBot?
- 5 mins

Programming
60 mins

Before to program, let the students to download the color sensor extension from the extension center.
- 5 mins

→ Set LED fill lights and explain the event executive examples to students.
- 45 mins

→ Challenge for students.
- 10 mins

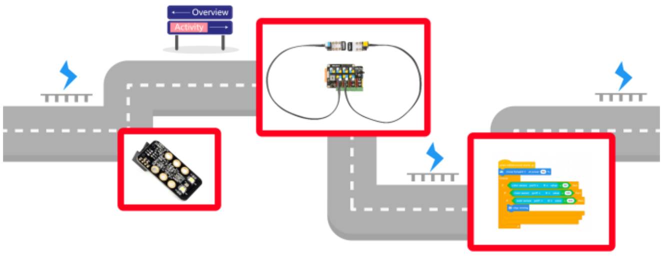
Key Programming Topics
15 mins

Combine the knowledge of RGB to demonstrate the difference between high-level and low-level colors in mBlock 5.
- 5 mins

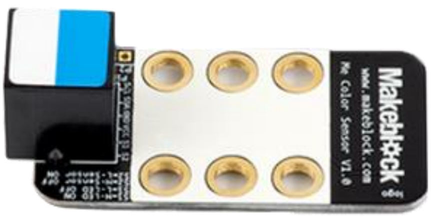
→ Students go back to the programming example to identify and solve problems.
- 10 mins

Reflection
5 mins

Part 1 Overview



The color sensor is involved in many different missions in level one competitions. From simply successfully detecting certain colored cards on the map to using the information of colored cards on the map to perform various maneuvers, knowing how to properly use the color sensor is essential to level one competitions.



Objectives

Students can retrieve the output of a color sensor using LED board.

Students can use a color sensor in conjunction with onboard LED lights.

Students can detect a standard color using a color sensor.

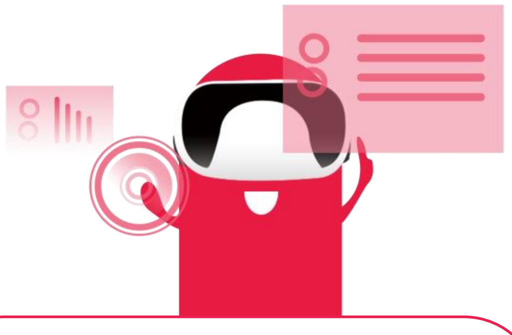
Students can detect a non-standard color using a color sensor.

Students can properly utilize RGB information to achieve multi-color separation.

Part 2 Inquiry

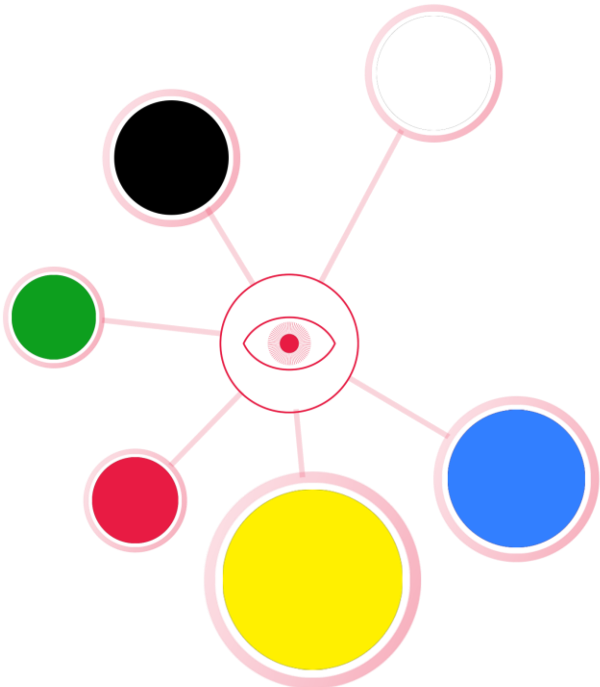
Hardware

The Me color sensor contains two digital color sensors and two LED fill lights and takes reflection light as input. The color sensor analyzes the light input and outs the RGB values of the light.



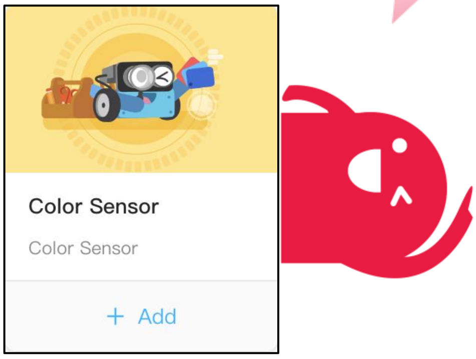
Since objects of different colors reflect light of different colors, students could use the output information to differentiate various colors.

- It is worth mentioning that the color sensor reading is greatly affected by shadows or the general lighting condition. It is generally accepted that the optimal distance the color sensor should be from the object of interest is a little bit less than 1 centimeter. It is also important to note that the sensor has a delay around 0.2 seconds.



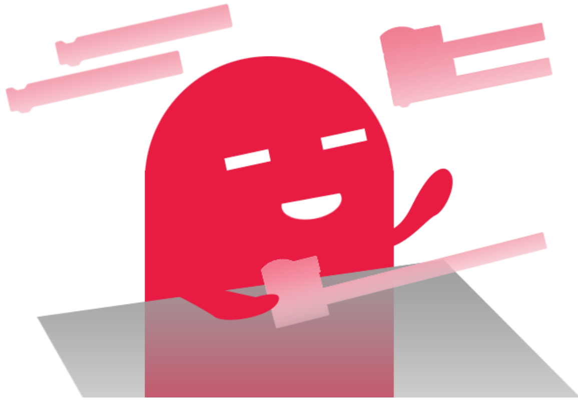
Part 3 Programming Control

Before using the color sensor in mBlock 5, students must first download the color sensor extension from the extension center.



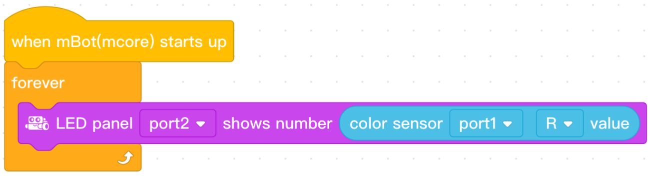
In mBlock 5, a new block to set LED fill lights either on or off is provided. Since the color is extremely sensitive to lighting condition, setting the fill lights on and off causes the color sensor to produce vastly different outputs.

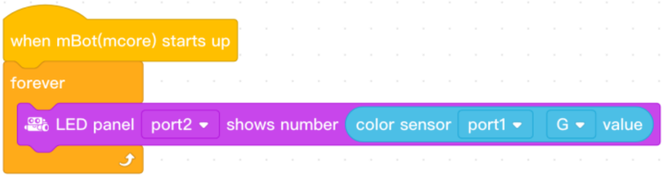
It is up to the students to decide which option helps provide the most useful and reliable stream of output. It should be noted that the availability of this new block opens the possibility of dynamically turning the fill light on and off during program execution. If not explicitly specified, the default is having the fill lights on. All color sensor blocks must be used in upload mode.



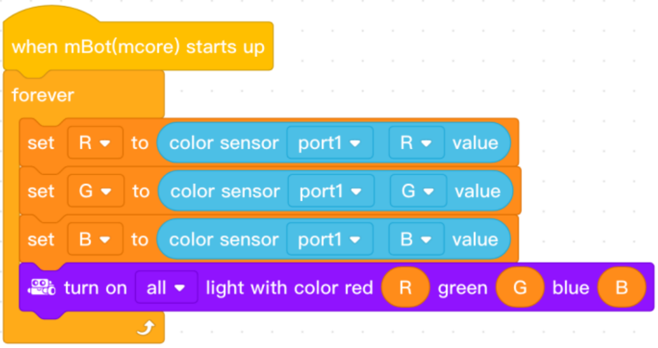
Programming Examples

When developing a program that utilizes the color sensor, it would be helpful to be able to see what the color sensor is reading. To do this, students could have the LED board display the output of the color sensor in real time. Look at the following program and think about why is the forever block necessary?



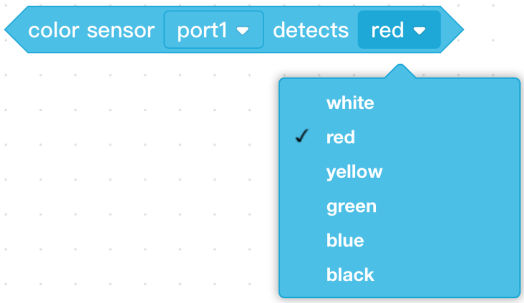


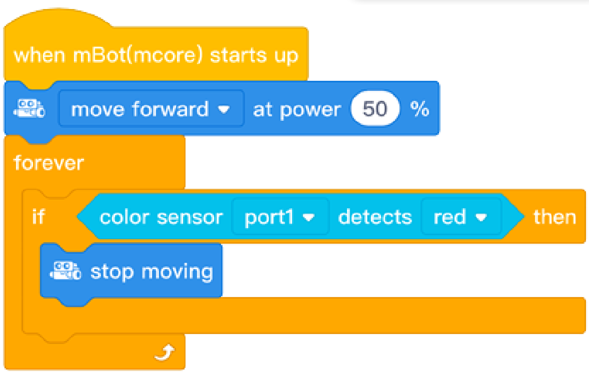
Another useful skill which is sometimes necessary in level one competitions is to set the onboard LED lights to whatever color the color sensor is sensing. This program could also be used to visually evaluate how accurately the color sensor is detecting the colors. Students can achieve this in two ways, with or without using variables.

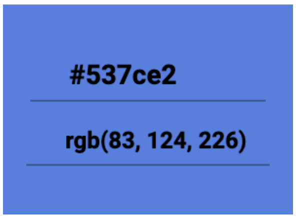


The 'color sensor detects' block allows students to ask whether the color sensor is seeing a certain color that matches one of the six built-in color profiles which include white, red, yellow, green, blue, and black. These color profiles are nothing magical but sets of pre-defined RGB values or ranges of RGB values that are built-in by the developers of the color sensor extension. Using the 'color sensor detects' block, students can easily make their mBots perform certain maneuvers when the color sensor has detected certain colors.

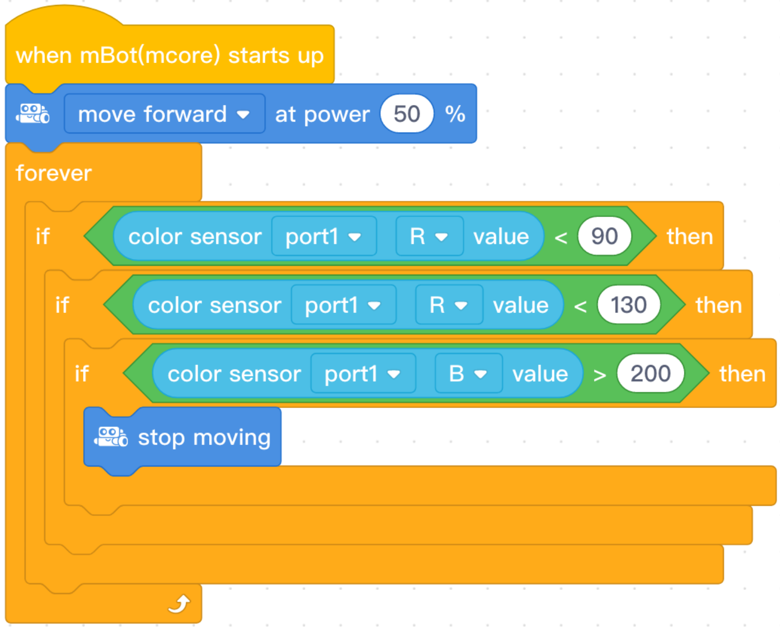
Using the 'color sensor detects' block, students can easily make their mBots perform certain maneuvers when the color sensor has detected certain colors. In the following example, the mBot simply stops when the color red is detected.





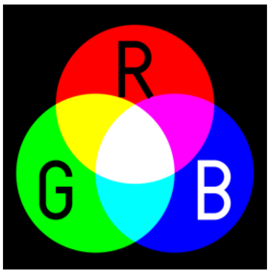


Since the built-in color profiles are limited to only six colors and the pre-defined RGB values might not suit the lighting condition of a given usage scenario, students may want to define their own color profiles. To do so, students could use the R/G/B value blocks in conjunction with if statements. Suppose the color of interest has the RGB values of (83, 124, 226), what would the program look like in order to achieve the same mission as the example above?



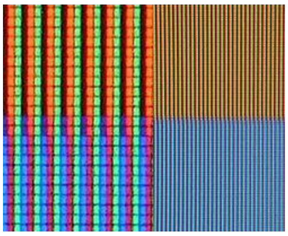
Depending on how similar the color of interest is to the color of other map features and whether it is the only color of interest, the program above may or may not work well. The reason is that the range we have defined contains approximately 90 x 130 x 55 = 643,500 different colors. It is up to the students to decide how narrowly they want to define the range in order to reliably complete their missions. Another question for students to think about is whether there is a way to simply the program above?

RGB



It is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

The main purpose of the RGB color model is for the sensing, representation and display of images in electronic systems, such as televisions and computers.



RGB sub-pixels in an LCD TV (on the right: an orange and a blue color; on the left: a close-up)

RGB phosphor dots in a CRT monitor

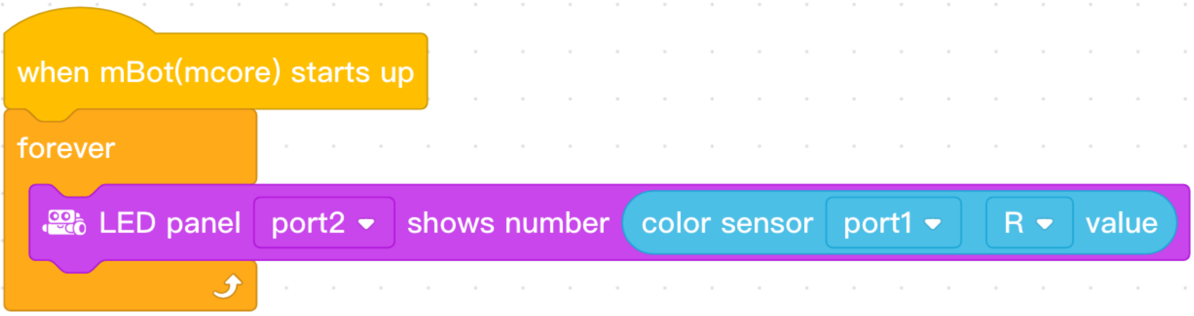
RGB is a device-dependent color model: different devices detect or reproduce a given RGB value differently, since the color elements and their response to the individual R, G, and B levels vary from manufacturer to manufacturer, or even in the same device over time. Thus an RGB value does not define the same color across devices without some kind of color management.

In mBlock 5, the level of each primary colors ranges from 0 to 255, where a lower value represents a lower level of the color and a higher value representing a higher level of the color.

It is interesting to note that the maximum number of colors the color sensor can recognize in mBlock 5 is approximately
255 x 255 x 255 = 16,581,375.

Execution Flow

Going back to the first programming example, why is the forever block necessary for the LED screen to display the sensor output in real time? The reason is that without the forever block, the execution simply flows through the 'show number' block once and terminates since there is no further blocks to be executed. Thus the LED board would only display the initial sensor output and never update the value again. Despite the simplicity of this concept, having a solid understanding of the execution flow is crucial to building or debugging larger, more complex programs.



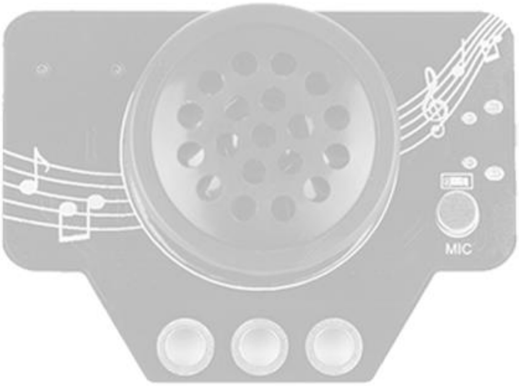
Part 5 Reflection

Does the color sensor produce the same output in different lighting conditions?

What is RGB and what is it used for?

Can you come up with two different methods to stop your mBot at a certain color?

Mp3 Audio Player



06

Teaching Suggestion

Lesson 6 Mp3 Audio Player

Overview
10 mins

Introduce the lesson by asking questions: as the sound of the buzzer is really monotonous, why can't we find a way to change it?
- 5 mins

→ Show the content and learning objectives of this lesson.
- 5 mins

Inquiry
25 mins

Hardware introduction
- 10 mins

→ Students decide what music to play and try to download it.
- 10 mins

→ Connection Method: how to use a Me Audio Player with the mBot?
- 5 mins

Programming
75 mins

Use the mBlock Programming function list to explain programming control of Me Audio Player.
- 10 mins

→ Let students think and discussion: What are the necessary functions for the buttons of common audio devices?
- 5 mins

→ Show and explain the event executive examples.
- 40 mins

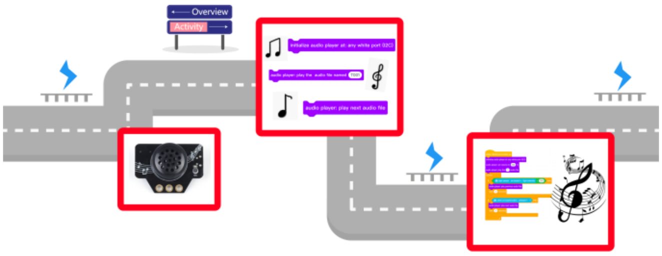
→ Challenge for students.
- 20 mins

Key Programming Topics
5 mins

Emphasize the usage of wait until blocks to students.
- 5 mins

Reflection
5 mins

Part 1 Overview



In this lesson, students will first focus on mastering the basic operations of the Me Audio Player. Then students will have a chance to explore some creative usage of the audio player and thereby deepening their understanding of this piece of electronic.

Objectives

I can understand the installation and setup process of the audio player.

I can properly process audio files into the correct format and store them into the TF memory as required by the audio player.

I can come up with creative ideas utilizing the audio player hence increasing its usability.

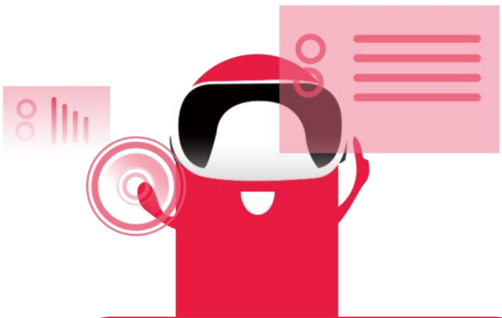
Can I extend my knowledge of common sensors?

I can enable and realize basic audio playback through programming.

Part 2 Inquiry

Introduction of the Me Audio Player

Me Audio Player module, compatible with the entire series of makeBlock control boards, is able to play back and record sounds with a built-in decoder. This module's connection port is marked with a white sticker, meaning that it is controlled by I2C signals, and must be connected to a port with white sticker on the main board.



An external TF memory card is required to store audio files.

Usage, Audio storage and Connection

- A non-flashing, blue on-board LED indicator suggests the audio player is in play back mode, while a flashing indicator means it's in recording mode.

- The module's metal hole area is the reference area in contact with the metal beam;
- With reverse polarity protection, reverse current will not damage the IC;
- Since the Me Audio Player module's connection port is marked with a white sticker, when using the RJ25 port, it needs to be connected to a port with white sticker on the main board;
- The on-board micro USB connector could be used to edit files in the TF memory card. Therefore, a card reader is not necessary;
- The module supports the following audio file formats: MP3, WMA, and WAV.

mBlock Programming

1. Choose a port

initialize audio player at: any white port (I2C)

2. Playback by specifying file index

audio player: play the 1 audio file

3. Playback by specifying file name

audio player: play the audio file named T001

4. Specify playback mode

audio player: set mode to single ▾

5. Play previous audio file

audio player: play previous audio file

6. Play next audio file

audio player: play next audio file

7. Toggle between a pause and continue playback

audio player: pause/continue

8. Stop playback

audio player: stop playing

9. Volume adjustment by explicit percentage specification

audio player: set volume to 50 %

10. Increase volume by default amount

audio player: volume up

11. Decrease volume by default amount

audio player: volume down

12. Record and save the file in name specified

audio player: start recording and save to T001

13. Stop recording

audio player: stop recording

Part 3 Programming Control

What are some essential functionalities the button on the audio player should be able to realize?

How to play back an audio file by specifying its index?

Although the following program is simple in construction, it could be a powerful debugging tool for more complex programs. Especially for checking out issues with the audio files themselves.

```
when mBot(mcore) starts up
initialize audio player at: any white port (I2C)
audio player: set volume to (100) %
audio player: play the (1) audio file
```

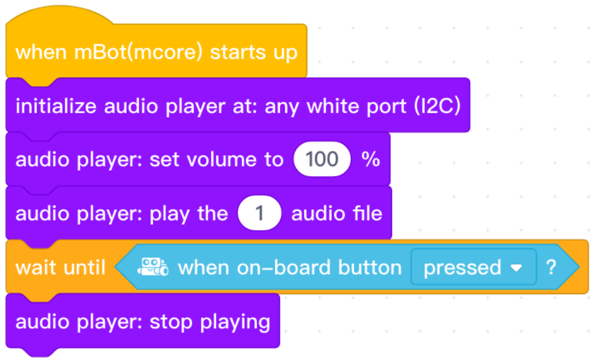
How to control the audio player to stop playback after a specified period of time?

Students can adjust the 'wait' time to play back the audio file for the specified duration.

```
when mBot(mcore) starts up
initialize audio player at: any white port (I2C)
audio player: set volume to (100) %
audio player: play the (1) audio file
wait (5) seconds
audio player: stop playing
```

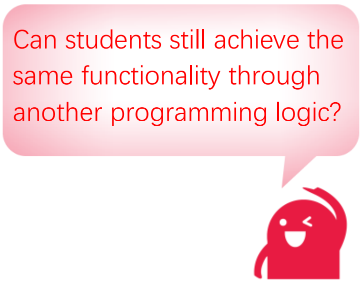
How to play back an indexed audio file indefinitely and stop at any moment?

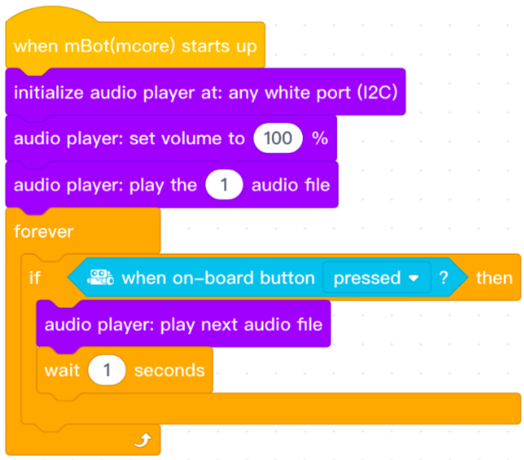
The on-board button could be programmed to interrupt audio playback at any moment. The effect is the same as playing back for a non-specified time frame.

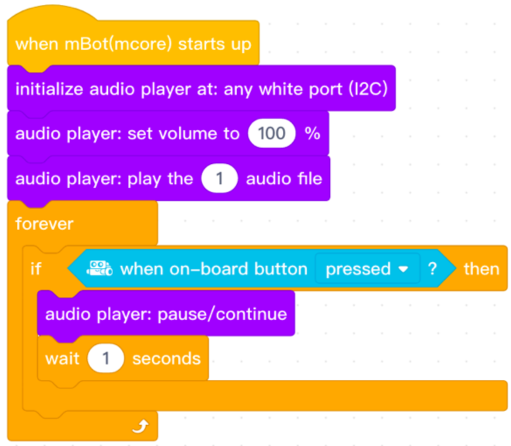


Can you write a program to play the next audio file without using 'wait' or 'wait until' blocks?

This sample program is for inspiration purpose only.





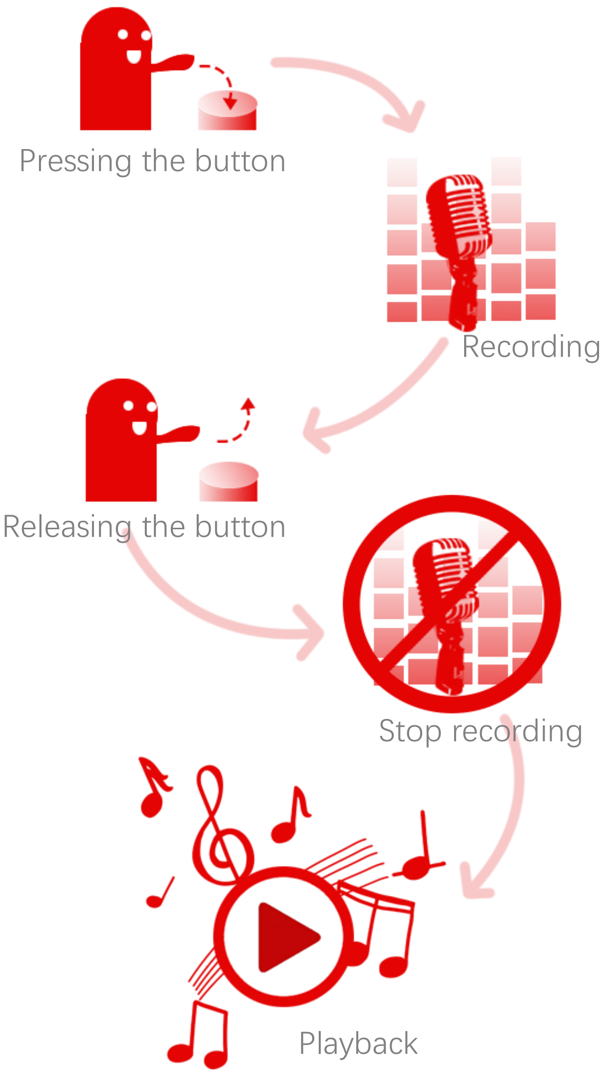


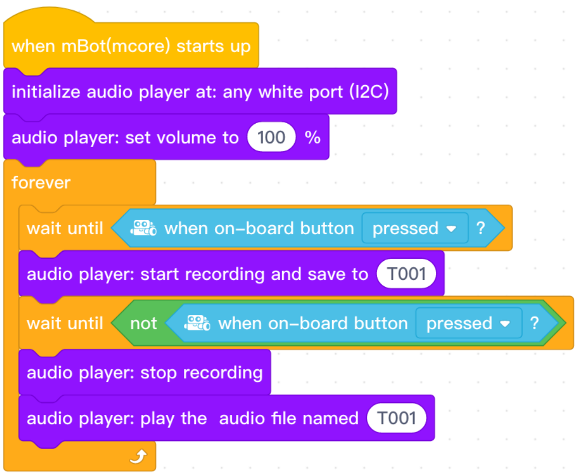
Challenge:

Is it possible to use only one physical button to go to either the next or previous file?

What is the work flow of recording and then playing the recorded file? Why is a clear flow chart crucial to programming?

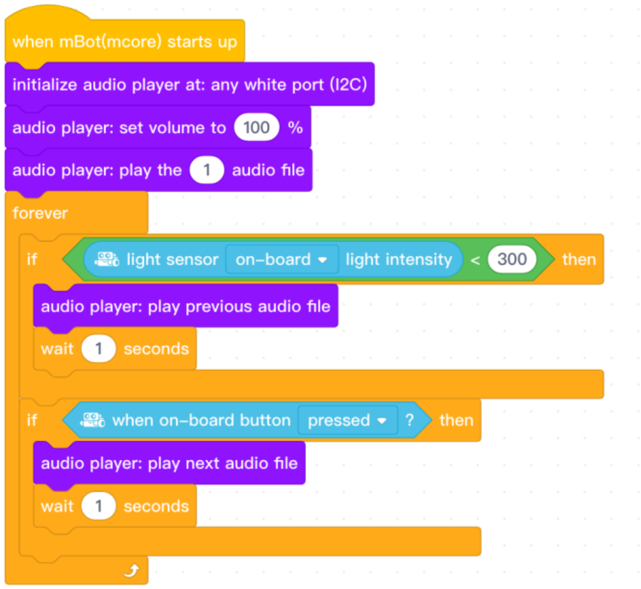
Have the students come up with flow charts and briefly describe them.





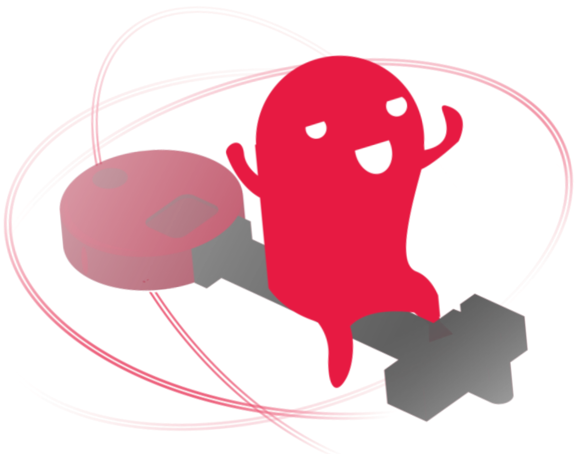
How could sensors such as light sensor, ultra-sonic sensor, or RGB line follower be utilized to help control the audio player?

- Which sensor is most suitable for this task and why?
- Can you make it happen? What other creative ideas do you have?



Wait until

The Wait Until ( ) block is a Control block. The block halts program execution until the specified Boolean condition is true.



Part 5 Reflection

What are the precautions when using the Me Audio Player?

Is it possible to implement all the common features of an audio player using the Me Audio Player?

What was the most challenging parts of today's lesson for you? How did you overcome the difficulty?

Bluetooth Controller

07

Teaching Suggestion

Lesson 7 Bluetooth Controller

Overview
10 mins

Introduce the lesson by asking questions: After writing the program, the mBot starts running by itself……Is that all? Do you want to control it?
- 5 mins

→ Show the content and learning objectives of this lesson.
- 5 mins

Inquiry
20 mins

Module Introduction
- 10 mins

→ Connection Method
- 5 mins

→ Bluetooth Pairing
- 5 mins

Programming
75 mins

Use the mBlock Programming function list to explain programming control of Bluetooth Controller. - 10 mins

→ Explain the event executive examples.
- 40 mins

→ Challenge for students.
- 20 mins

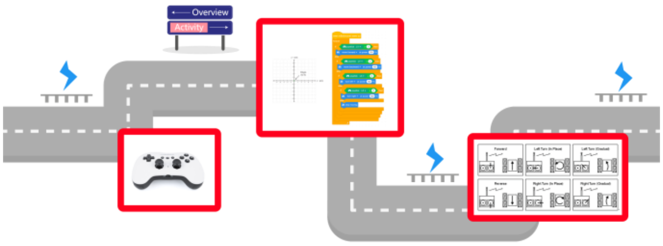
Key Programming Topics
5 mins

In order to make students better use the Bluetooth Controller, the concept and usage of the joystick should be proposed.
- 5 mins

→ Emphasize the usage of () / () blocks to students.
- 5 mins

Reflection
5 mins

Part 1 Overview



In this lesson, students will learn the basics of using the Bluetooth controller. And they will try to customize each button or joystick to realize the control of mBot hardware.

Objectives

I can understand the layout and features of the Bluetooth controller.

I can pair and set up the Bluetooth control to work.

I can program to use the Bluetooth controller to control mBot movement.

I can use other display devices to read the return parameters of the joystick.

I can associate the displacement of the joystick with the wheel motor speed.

I can use mathematical operations to complete accurate assignments.

I can understand the advantages and disadvantages of precise numerical and range definitions in the program.

Part 2 Inquiry

Module Instruction

Bluetooth Controller features 15 buttons and 2 thumb sticks for a more flexible Makeblock robot control.

It can allow you to customize control operation by defining each button with a function in an anti-interferent operation during robotics competition, within the transmission distance of 20m.

Connection Method

Bluetooth controller can only be connected to the Makeblock robots with a Bluetooth module.

Step 1 - Turn on your robot and connect it to the computer via USB cable.

Step 2 - Open mBlock 5 and select the device as mBot.

Step 3 - Select the currently used serial port.

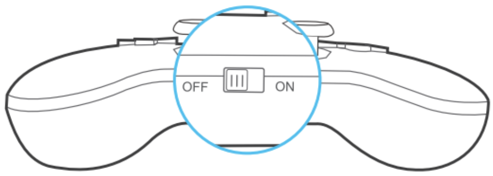
If multiple serial port options appear, please select the port that is newly added after the robot is connected to the computer.

Step 4 - Select the device as mBot and add the extension file.

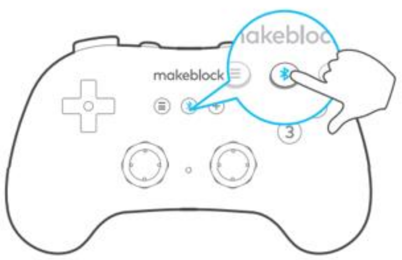
Step 5 - Start to program the controller.

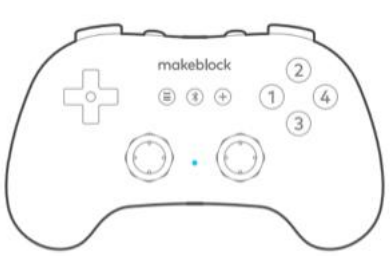
Bluetooth Pairing

Step 1 - Turn on the controller. The indicator flashes blue.



Step 2 - Get the controller close to your robot. Press the “Bluetooth button” until the indicator flashes more frequently, then release the button, and the controller will connect to your robot automatically.





Part 3 Programming

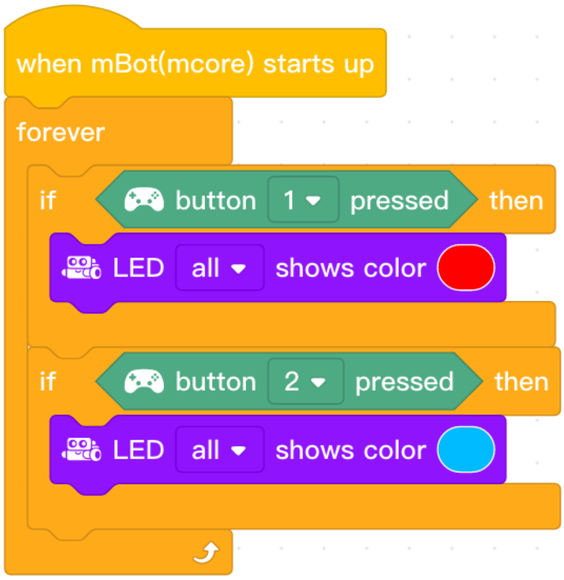
mBlock Programming function list

Joystick RX, RY, LX and LY control

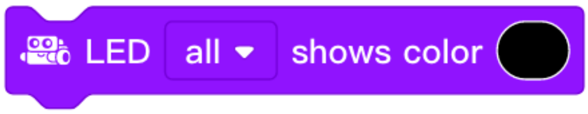
Buttons of controller control

How to switch the color of the onboard LEDs by using the buttons on the Bluetooth controller?

Usually we use this program to test whether the Bluetooth remote control is connected to the motherboard. Therefore, students are advised to add this program to the list of basic programs for troubleshooting hardware and software.

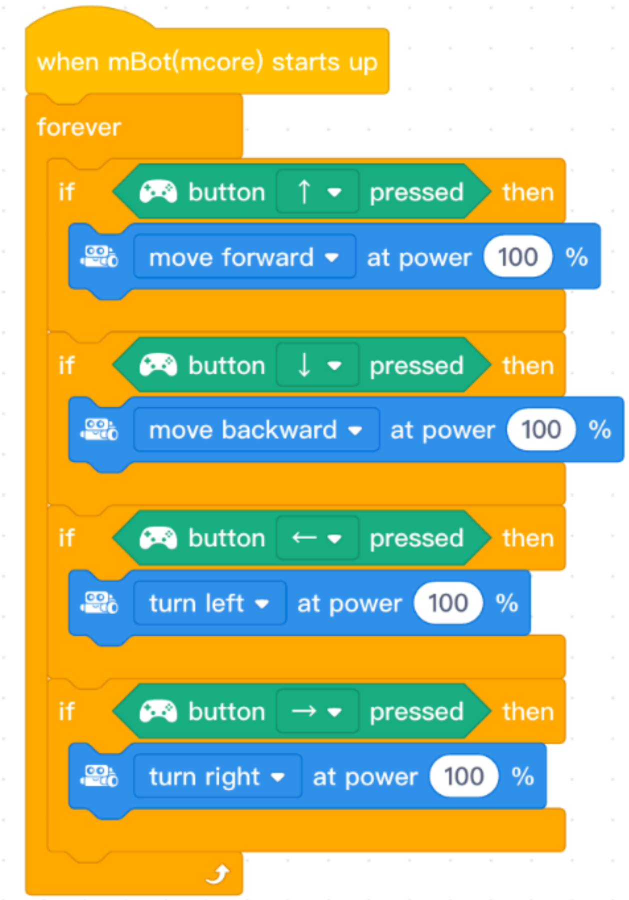


When working on the LED color control program, we usually add the following script block before the loop. Students can consider why they need to join this block?



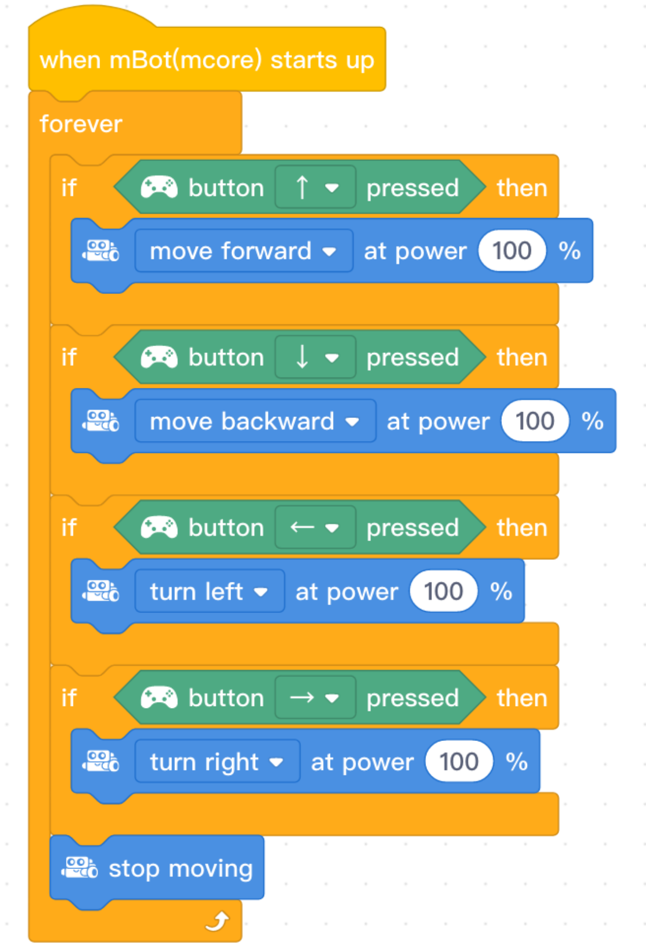
How do we program to control the mBot movement via a Bluetooth controller?

According to the above LED programming control learning, we can use the "if...then..." condition to switch the states through the buttons on the Bluetooth controller.



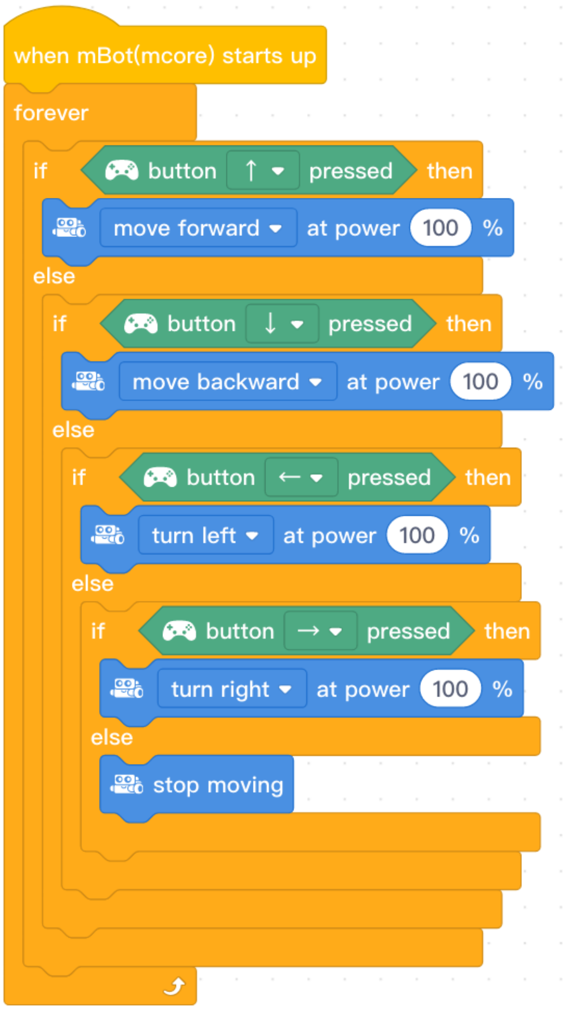
So, after the students complete this sample program, can they control the mBot to move normally?

If not, can students try to describe the problem found and try to find a solution?



This is a plausible ending solution, students can try to add a "stop mobile module" in different locations to solve the problem of how to stop mBot. But does this solution bring new problems?

Bluetooth control sample program



Challenge:

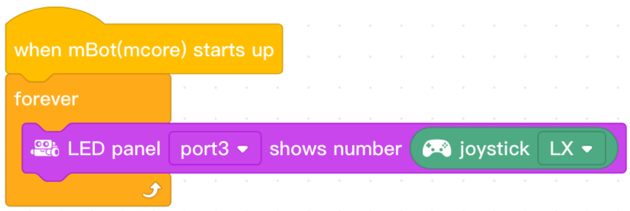
Can students use the onboard LED to indicate the different states of travel?

e.g.
move forward = green
move backward = blue
turn left = left LED
turn right = right LED
stop = red

Is it possible to use the left and right motor differential control modules to change different turning radii in the left and right turn control?

How to read the XY axis value of the joystick on the Bluetooth controller?

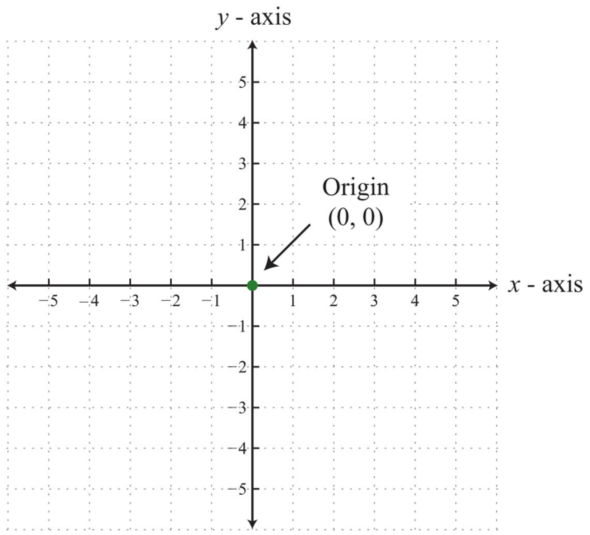
Before using the joystick, the most important thing is to understand the returning value on the mBlock 5 software. We can edit the control program according to the definition of different values or ranges.

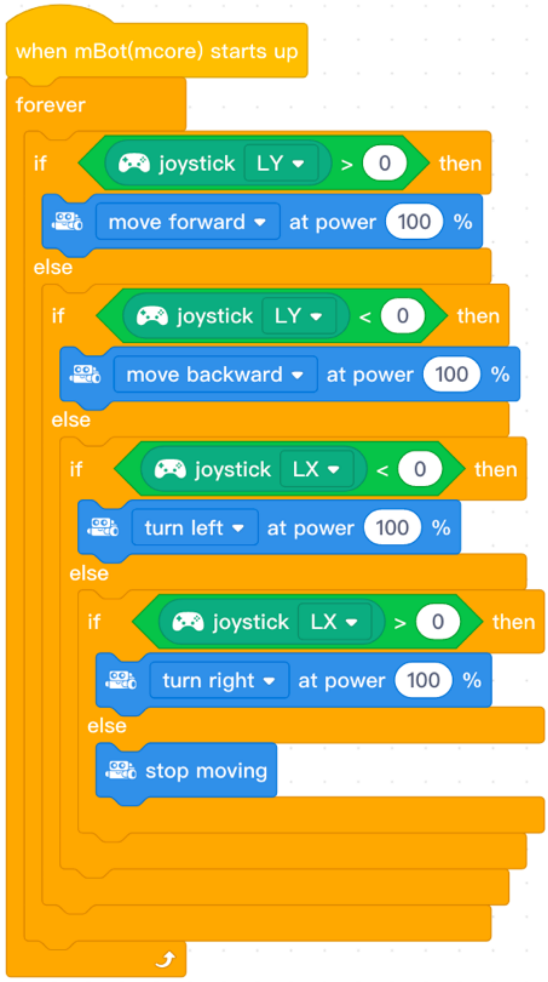


Students can upload this program and draw a schematic diagram of the XY axis, marking the highest value of the two ends of LX, LY, RX, RY and the value or value range after the joystick is homed

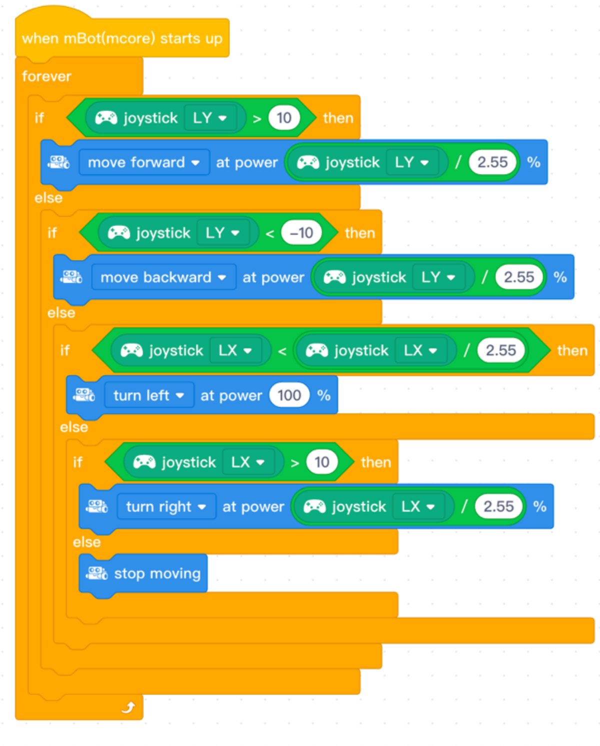
How to program to use the joystick on the Bluetooth controller to control the movement of mBot?

When the joystick leaves the home position, how does the remote control determine the direction in which it is pushed?



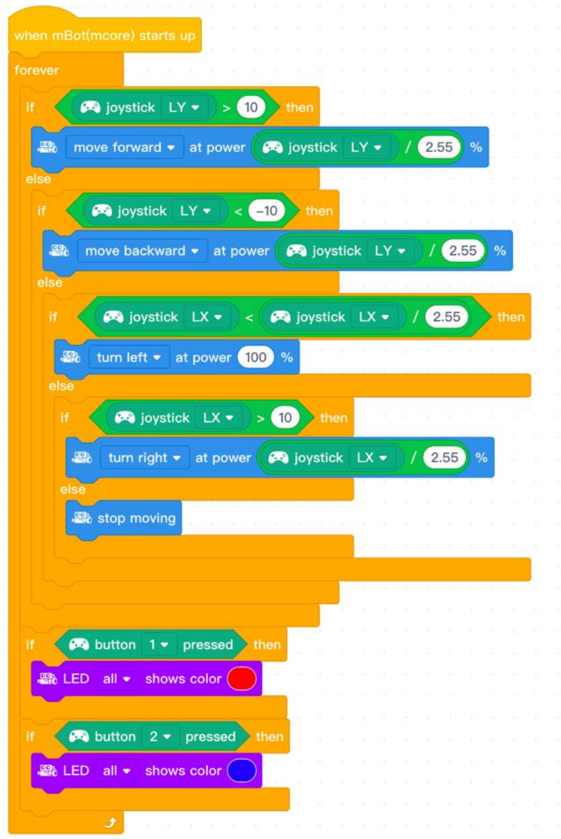


How do we program to change the movement speed of the mBot with the displacement of the joystick?



- When the joystick leaves the home position, how does the remote control determine the direction in which it is pushed?
- Why do we need to add the mathematics of "divide by 2.55" to the program?
- Why do we use greater or less than positive or negative 10 instead of 0 in our program?

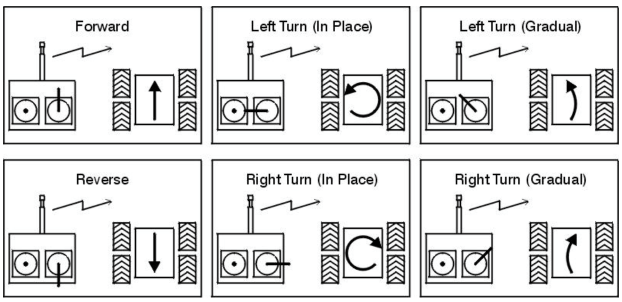
How do we combine the Bluetooth controller controlled motor movement program with other sensors or actuator control programs?



- Students can perform creative extension programming based on this sample program. You can also try to adjust the parameters in the program to feel the changes in the control program.

Joystick

A joystick is an input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling. A popular variation of the joystick used is the analog stick. We can push the joystick to all directions or push it down as a button.



() / () Block

The () / () block is an Operators block and a Reporter block. The block divides the second value from the first and returns the result. If the first value is not evenly divisible by the second, the reported value will have decimals. The numbers can be typed directly into the block, or Reporter blocks can be used instead.

This block can be stacked inside itself — this can be used to fit more numbers in.

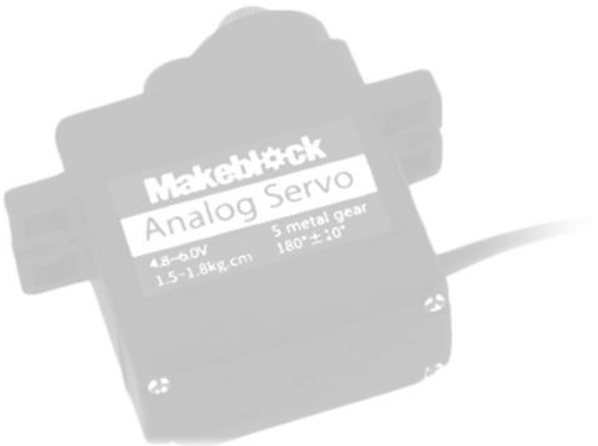
Part 5 Reflection

How can we use the Bluetooth controller to control the mBot's motors and actuators?

How to read the analog value of the joystick? Why do we have to read the joystick returning value?

How does the differential control of the wheel motor and the speed control through the joystick displacement help in the process of completing the task?

9g Servo



08

Teaching Suggestion

Lesson 8 9g Servo

Overview 15 mins

Lesson introduction and objectives announcement
- 15 mins

Inquiry 30 mins

Introduction of the working principle of a servo
- 10 mins

→ Servo wiring and application
- 10 mins

→ Servo hardware preparation-holder and arms
- 10 mins

Programming 60 mins

Programing preparation to add the extension of servo in the software
- 5 mins

→ Programming-Basic program to control the 9g servo in mBlock 5
- 3 mins

→ Program comparation and analysis
- 40 mins

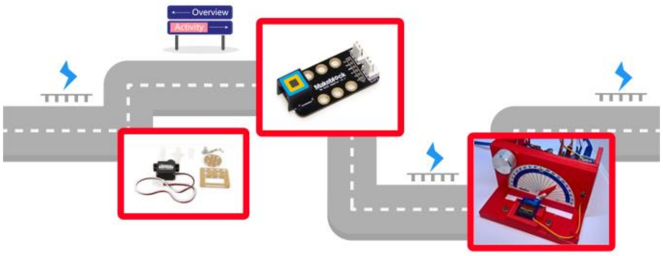
→ Challenge inquiry
- 12 mins

Key Programming Topics 10 mins

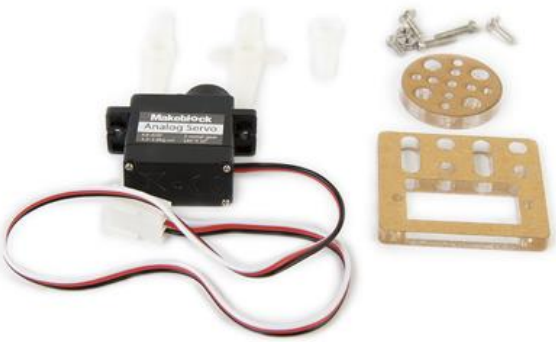
Key Programming topics- button debounce and repeat until
- 10 mins

Reflection 5 mins

Part 1 Overview



In this lesson, students start with the most basic how to properly install the servo and its accessories, and through the step-by-step procedure, finally edit the more complicated program that uses the Bluetooth controller to control it. And in this lesson, there is more hardware knowledge, and students can understand that in robot control, the understanding of hardware characteristics is also very important.



Objectives

I can understand the design parameters and basic usage of MakeX 9g servos.

I can learn how to complete the connection of the servo through the RJ25 adapter to the mCore motherboard.

I can learn how to use the servo correctly and limit it in the program according to its environment.

I can use the Bluetooth controller to switch the steering angle and speed change

I can understand the concept of "debounce" in hardware control programs.

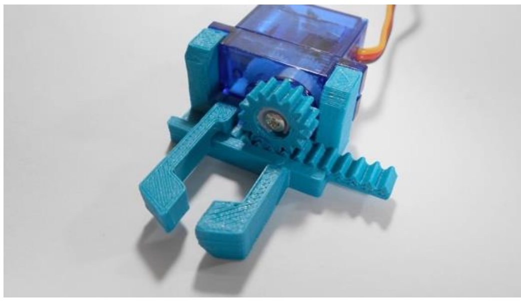
Part 2 Inquiry

Module Instruction

A servo motor allows a precise control of the angular position, velocity, and acceleration. It's like you're at the steering wheel of your car. You control precisely the speed of the car and the direction.

This very strict control of the angular position, velocity, and acceleration can't be done without a sensor for position feedback. This sensor sounds the alarm when the motor is spinning. But even so, there is something more sophisticated that controls all the stages of the servo motor. It's a dedicated controller that makes the tiny things inside the servo to move with military precision.

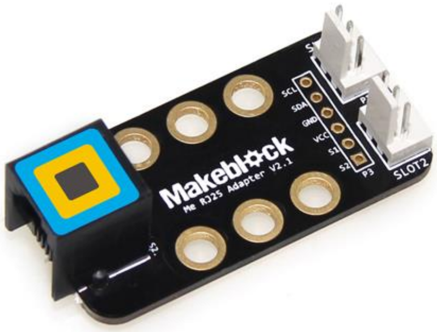
MakeX 9g Micro Servo



- MakeX 9g Micro Servo Pack is a servo pack for participants who need to make a rotating device , it contains a 9g servo, a servo hub, a servo bracket and hardwares.
- The MakeX 9g Micro Servo can rotate approximately 180 degrees, it works like standard servos but of course not as strong as a standard servo.

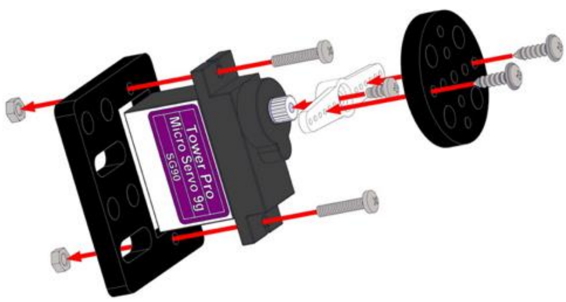
Connection Method

With the servo hub and servo bracket, it may be convenient to connect the servo with other Makeblock parts. A Me RJ25 Adapter also help you to connect the servo with mCore easily.

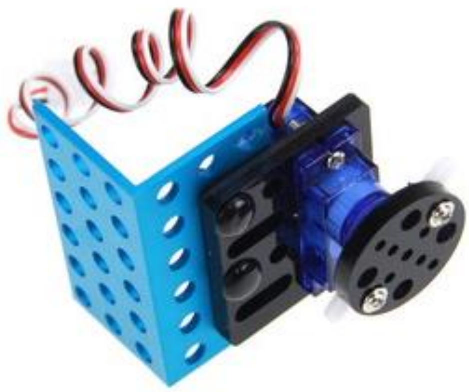


Application

These motors are classified into different types based on their application like Brushless DC , AC , continuous rotation, linear and positional rotation, etc. Typical servo motors contain of three wires such as, power control and ground. The outline and dimension of these motors depend on their applications.

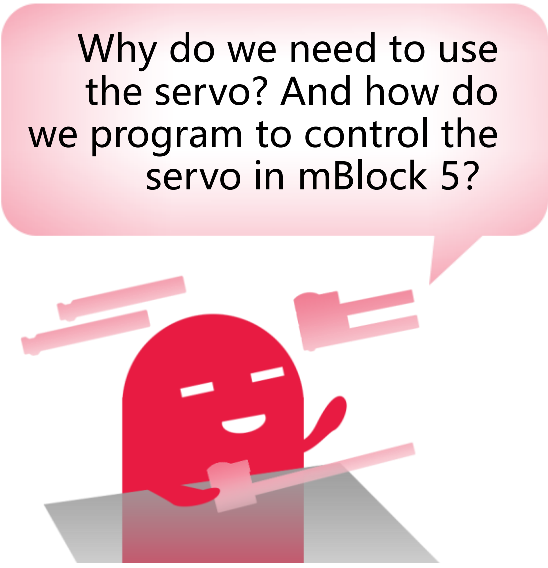


An exclusive design of this motor is suggested in controlling applications like the robotics. Basically, they are used to change the speed control at high torques and correct positioning.



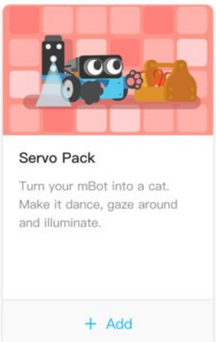
Part 3 Programming

Before to program the servo, we have some questions.

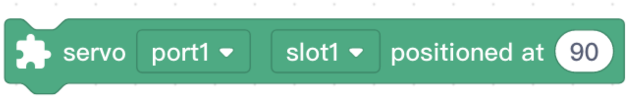


These servos are essential parts if we need to control the position of objects, rotate sensors, move arms and legs, drive wheels and tracks, and more.

And if you need to program to control the servo in mBLock, the first thing you need to do is to add the extension.



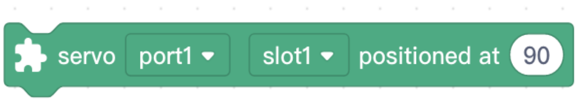
Then, you will find the following block which can be used to control the 9g servo.



Connect the servo on the RJ25 slot and mCore, set the servo to 90 degree to put on the servo arms, and we can have a go to start to program it.

mBlock Programming function

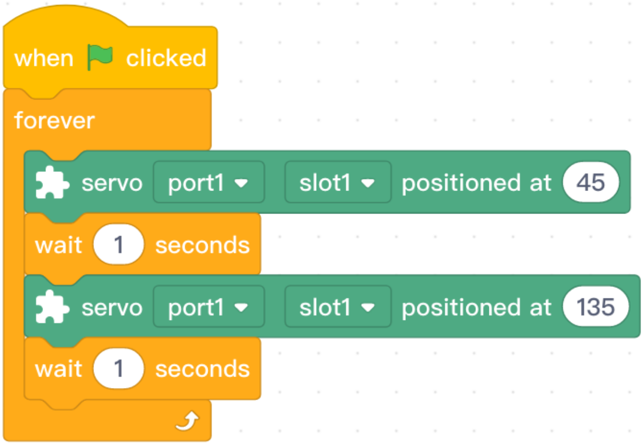
Set the servo to 90 degree

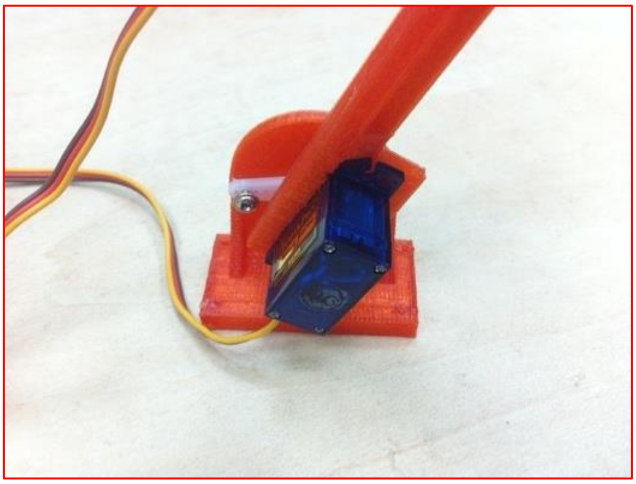


How to set the servo to the degree we want?

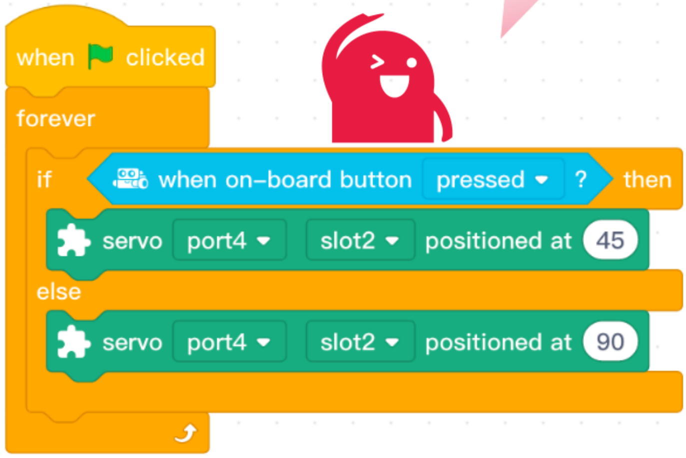
Check the port on mCore and the slot on the RJ 25 adapter. Make sure you have done the upgrade firmware of the mCore, and then double click on the block to set the angel of servo.

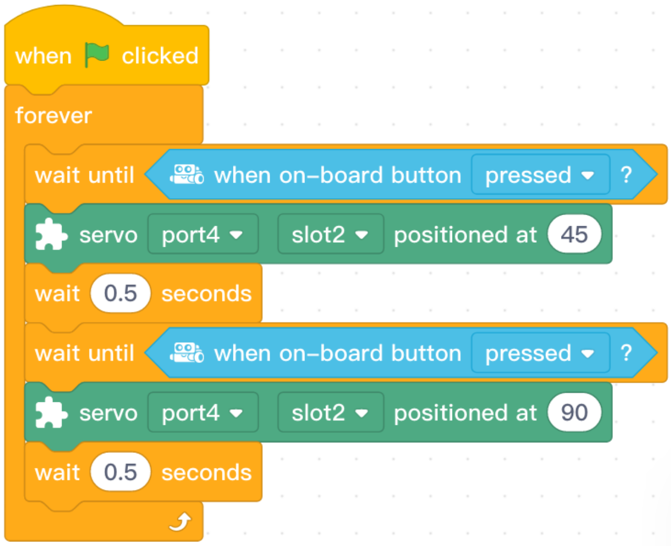
If we need something like a small robotic device to switch between two angles, how should we finish writing this control program?





What is the difference between these two programs?





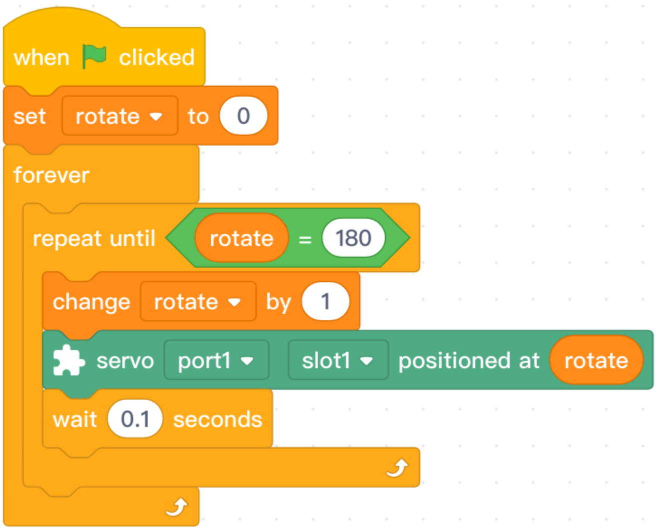
Why do you want to add a 0.5 second wait after each angle change in the second sample program?

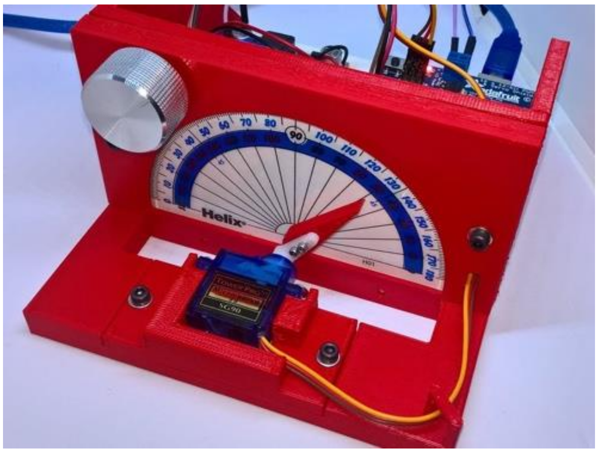
Debounce

Can we program to control the servo rotate from 0 to 180 degree by degree?

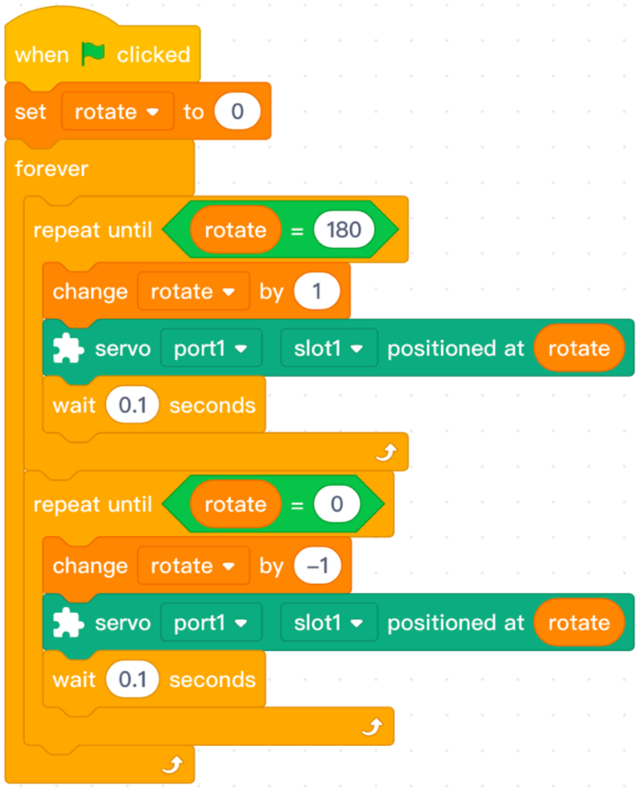
The 9g servo changes between two angles according to its own set speed. We can add variables to control its rotation speed and the rotation angle of a single motion.

How can we change the speed of servo rotation? What is the fastest speed of the rotation of 9 g servo?



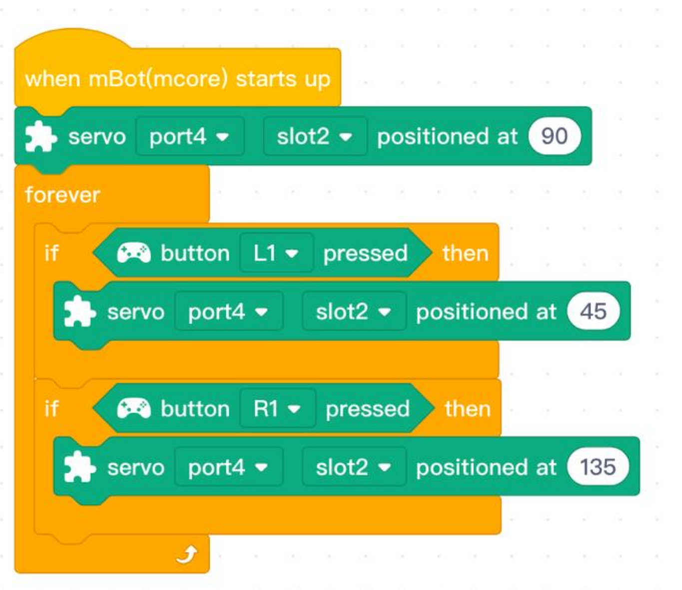


Can we control the 9g servo to rotate between 0 and 180 under the controllable speed conditions?



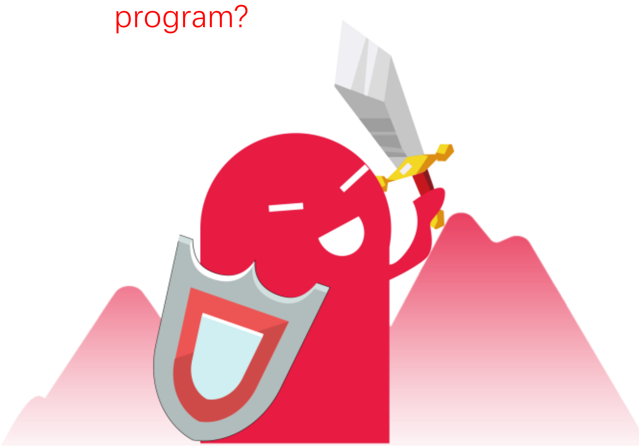
At the beginning of the program we set the variable to 0. After the program starts, because the initial value of the variable is 0, we enter the first "repeat until" loop until the variable equals 180 and jumps out into the second loop. In the first cycle, the incremental variable assignment gives the servo a set angle that has completed a 0 to 180 degree change. The second, on the other hand, is that the angle of the steering gear is gradually reduced to 180.

How do we use the buttons on the Bluetooth Controller to wirelessly control the steering of the servo between two fixed angles?



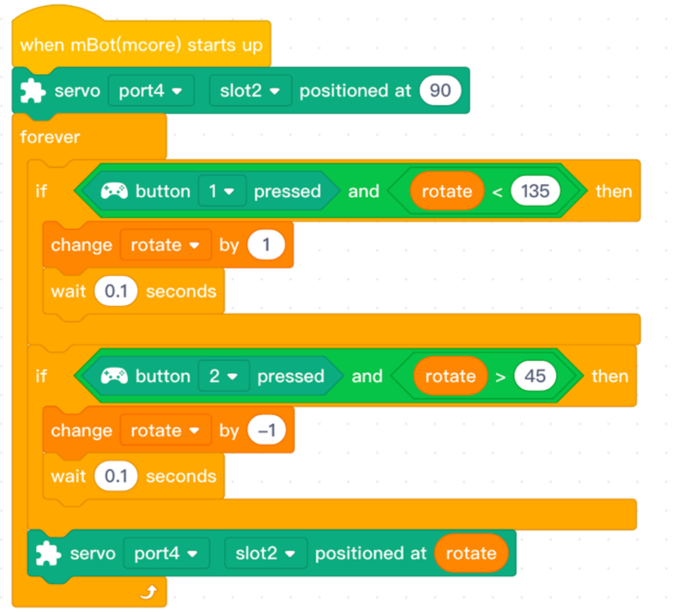
Challenge:

Can the student complete the switch of the steering angle by pressing the button on the remote control on the basis of the sample program?



Can we program to control the servo to move between two defined angles using the buttons of the Bluetooth controller, and the servo stops moving when the button is released?

Because of the structural design, sometimes the space that allows the servo to rotate can not reach 180 degrees. If the servo is stuck with the physical structure, it will cause great damage to the internal gear set and even the entire servo will burn out. So the program uses the "and" block to set the range of the rotation between 45 degrees and 135 degrees.

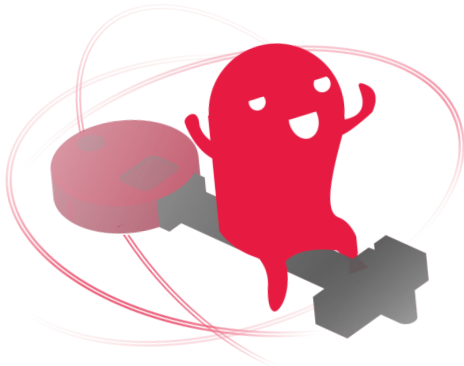


Challenge Program

Can we program to correlate the rotational speed of the servo with the displacement of the joystick?

Button Debounce

Pushbuttons often generate spurious open/close transitions when pressed, due to mechanical and physical issues: these transitions may be read as multiple presses in a very short time fooling the program. This example demonstrates how to debounce an input, which means checking twice in a short period of time to make sure the pushbutton is definitely pressed. Without debouncing, pressing the button once may cause unpredictable results.



Repeat Until ( )

The Repeat Until ( ) block is a Control block and a C block. Blocks held inside this block will loop until the specified boolean statement is true, in which case the code beneath the block (if any) will execute. This loop is in similar nature to a while loop in some other programming languages.

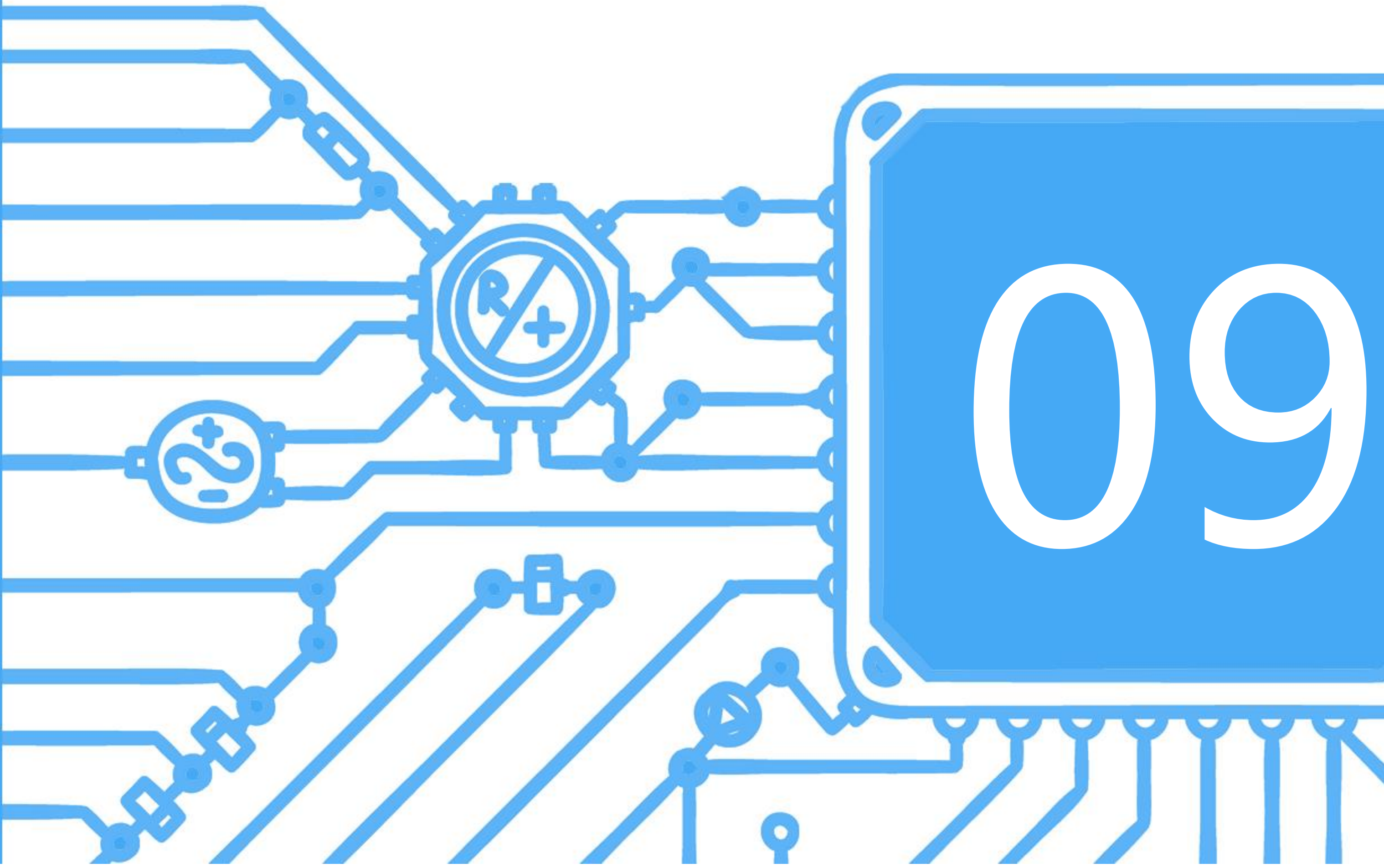
Part 5 Reflection

What is the working characteristic of the servo? How to protect the servo?

How to take into account the working environment of the servo to limit the operating range of the steering gear through the program?

What is the hardest part of the process you are studying today? How did you solve this part of the problem?

Advanced Programming Techniques



Teaching Suggestion

Lesson 9 Advanced Programming Techniques

Overview
25 mins

Kick off the class by a mission which provides a step by step guidance to students.
- 10 mins

Mission work reflection and lesson objectives introduction.
- 10 mins

Software and hardware cognition.
- 5 mins

Case Study
90 mins

Mission process breaking down and analysis.
- 10 mins

Five steps to implement the final program.
- 30 mins

Program test and debugging.
- 20 mins

Problems sorting and reflection.
- 30 mins

Reflection
5 mins

Part 1 Overview

In this lesson students will learn about several advanced programming techniques and tools utilizing the color sensor. These techniques and tools could be applied to other aspects of level one competition, given the reasonings and logics behind are well understood. Once mastered, students could produce much more efficient, reliable, and elegant programs that are easier to understand and reuse.

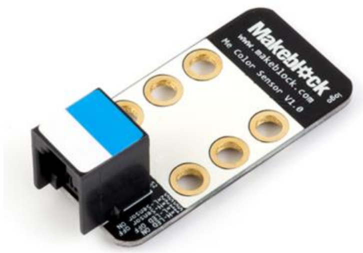
Objectives

Students can use the color sensor to count and memorize the number of appearances of a certain color.

Students can use a reliable technique to prevent overcounting in the use case above.

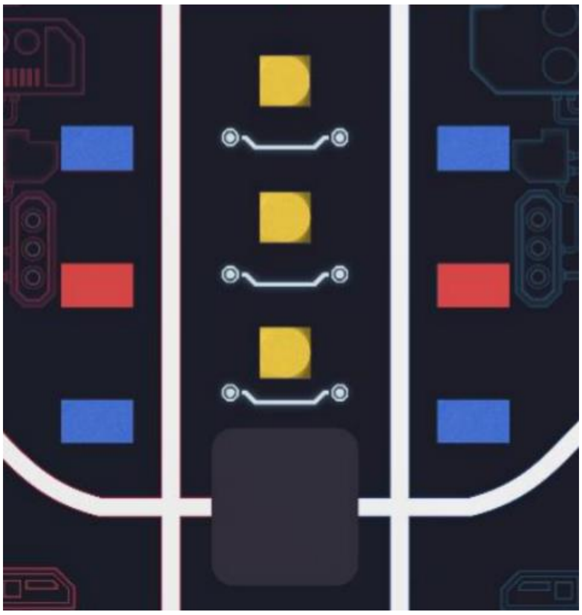
Students can understand the concept of logic gates.

Students can translate back and forth between if statements with logic gates and if statements without logic gates.



Part 2 Case Study

Imagine there is a sequence of blue and red cards alongside a track on the map:



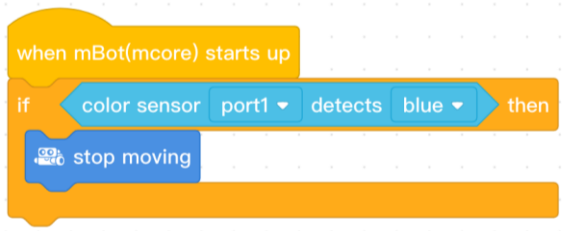
1. Please display a rolling count of the number of blue cards on the LED board

Before delving straight into the program, try decomposing the problem by thinking the following questions ——

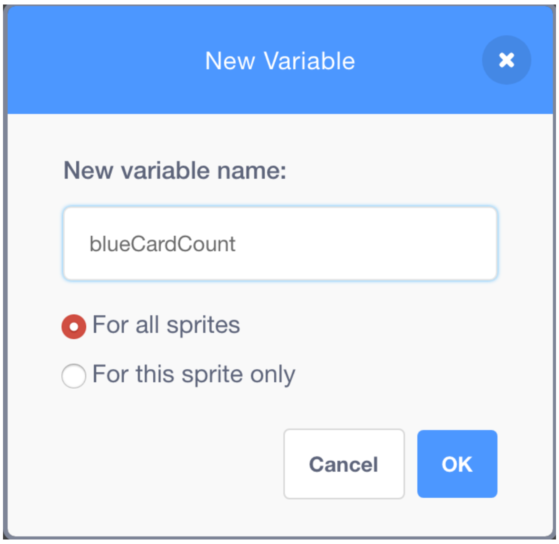
- How do you detect a blue card?
- How do you keep a count?
- When and where do you update the count?
- How do you display a rolling count?

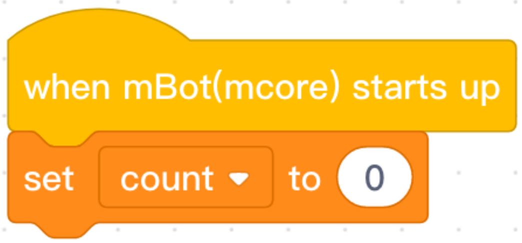
Being able to dissect a problem into these smaller, less complex questions is a crucial skill for programmers.

To detect a card of a certain color, it is obvious that a color sensor must be utilized. And according to the previous lesson on color sensor, using a simple if statement with the appropriate color sensor block can achieve this goal. The following program stops the mBot when a blue card is detected by the color sensor.

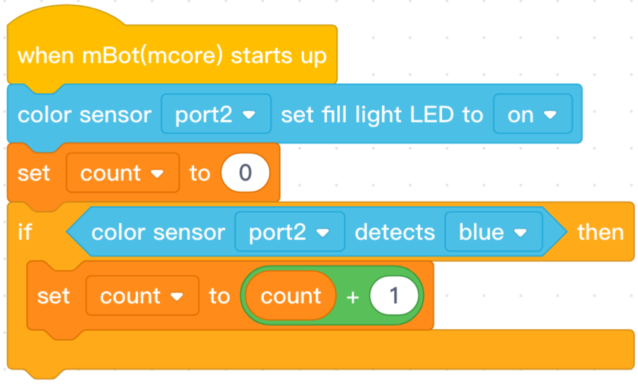


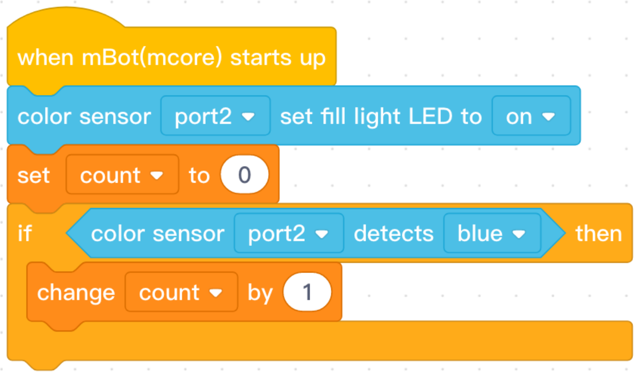
Since the count is a potentially changing number that must also be memorized, a variable is perfect for this job. To brush up, in mBlock5 a variable is a custom created block that stores numbers that could be dynamically referred to and/or changed during the execution of a program. First, a variable must a created and given a descriptive name. In the following example the variable is named 'blueCardCount'. Then, it is good practice to initialize the variable or in other words, set the variable to its initial value before the body of the program is executed.



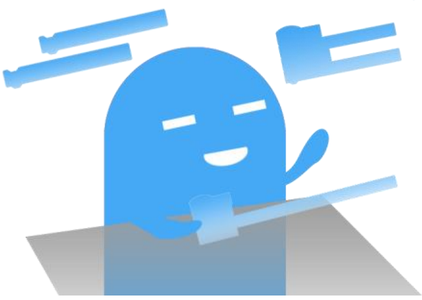


And regarding the question of when to update the count, it should be logically clear that the count should be updated inside the if statement after the target color has been detected. It should be noted that there are two methods to update the count, either by using the 'change by' block or by explicitly setting the count to the current count plus one. Understanding this equivalent relationship is a good exercise to further develop a programmer way of thinking.





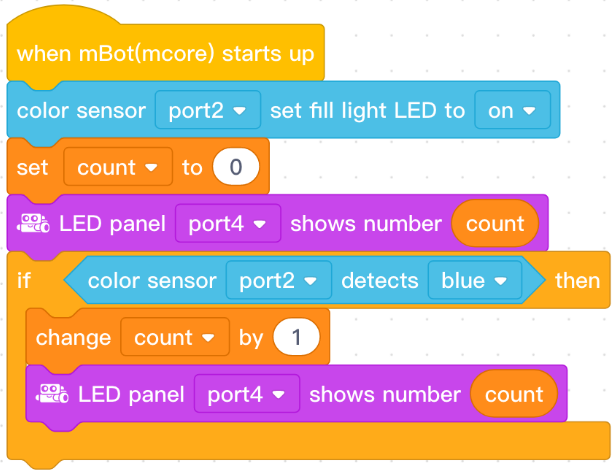
Although displaying a rolling count sounds relatively trivial, it is often unlikely for inexperienced programmers to get it right the first time.



Most would suggest putting a LED panel 'show number' block inside the if statement after the target color has been detected.

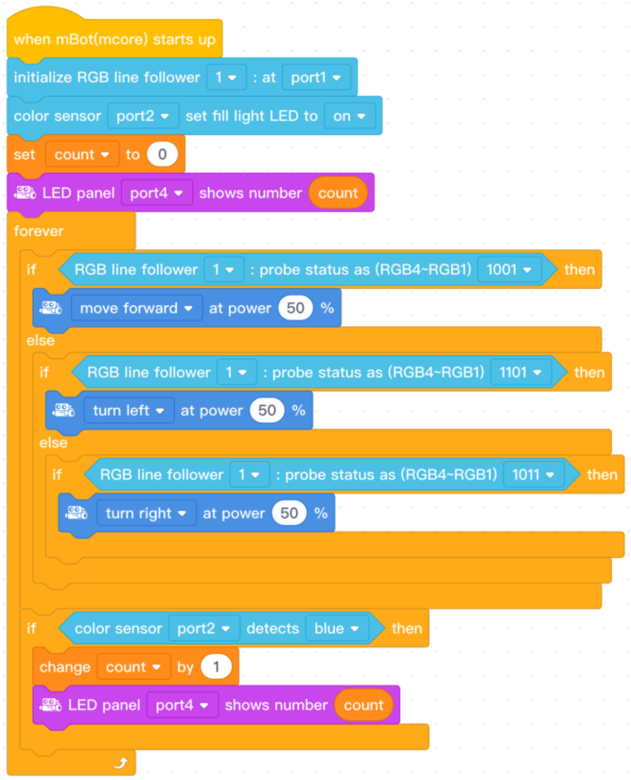
However, it is only correct if such a block comes after updating the count. Otherwise, the displayed number will always be off by one.

Moreover, the solution above is still not complete. If tested, the LED panel would never display count's initial value of zero. **Therefore, an additional LED panel 'show number' block needs to be placed outside of the if statement.**



Finally, to bring all the above together, the mBot must be in motion following the track.

Assuming there is no termination, the following program uses a forever loop. If the concept of execution flow is well understood, it should be clear that the line following logic should be placed inside the forever loop but outside the if statement, either before or after.



Now comes the testing phase!

Try to let the mBot run by only one blue card, see if the LED panel is displaying the expected count.

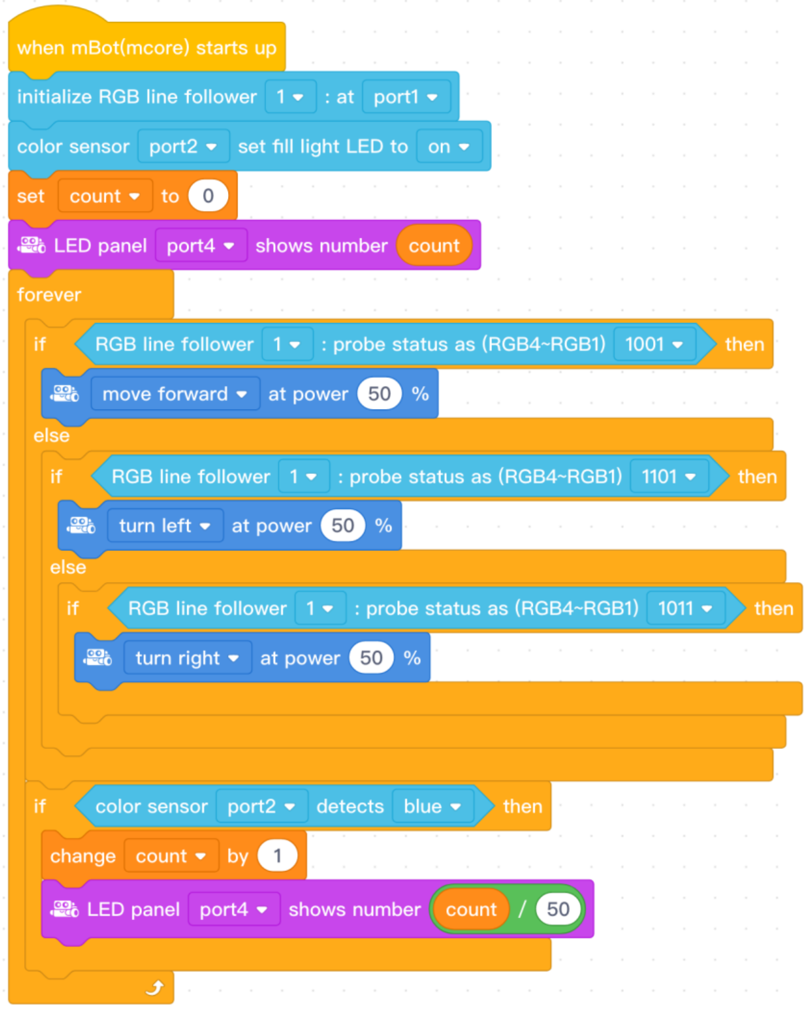
Depending on how fast the mBot is moving, any number in the range of lower tens to higher hundreds is possible. This type of issue is categorized as overcounting. Overcounting happens when the count variable is being unexpectedly and unreasonably updated more than it was supposed to.

In the case of the above program, the only possibility for overcounting to happen is the if statement' s condition being satisfied more than expected. Taking a closer look at the program and the problem itself, the if condition is met the moment the color sensor detects the blue card. And this condition is continuously being met the entire duration while the color sensor is on top the blue card. Since it takes time for the mBot to move past a single blue card and the execution flow being relatively speedy, the count variable is updated many more times than expected.

In the following section, four solutions to this issue will be introduced, each being more reliable and elegant than the previous one.

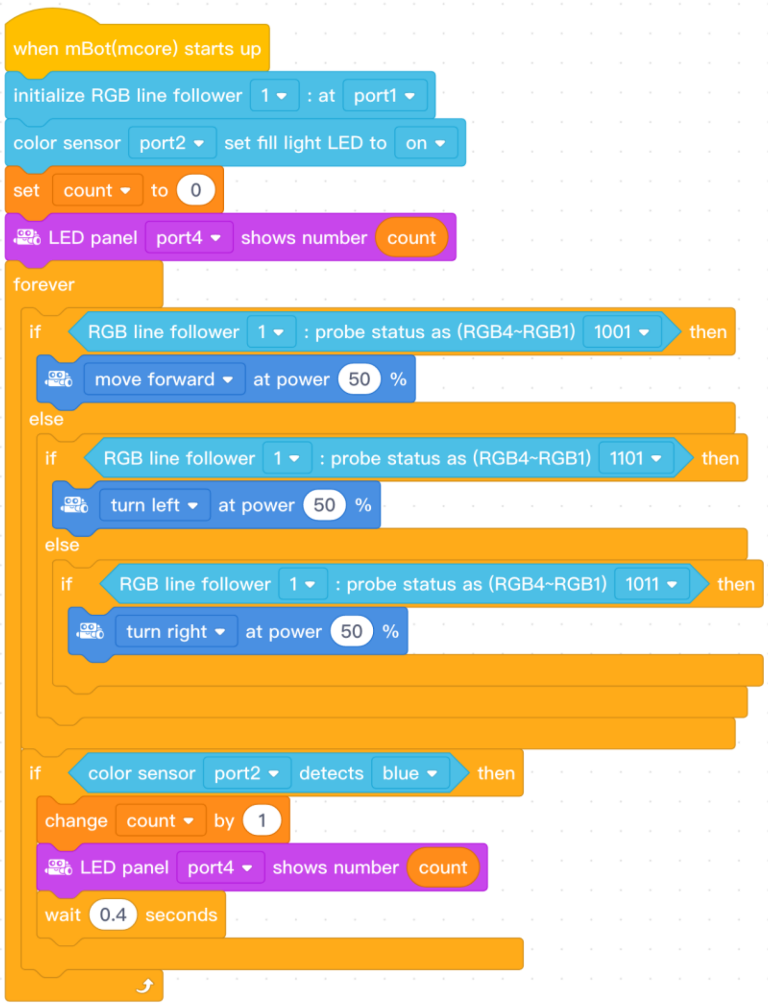
The first method
is to simply divide the count by the number of times each blue card is being counted.

For instance, if a single blue card produces a count of 50, then dividing the count variable by 50 should yield the correct number of blue cards. This method, in theory, should work provided a very reliable color sensor in a very controlled environment. Unfortunately, that is not the case. The color sensor is far from being consistent in its reading and the competition environment is far from being controlled, influenced by vibrations, lighting, etc. Therefore, the first method would not be an ideal solution in any measure.



The second solution has the advantage of being extremely simple to implement, which is to simply wait out the duration of the car passing a blue card.

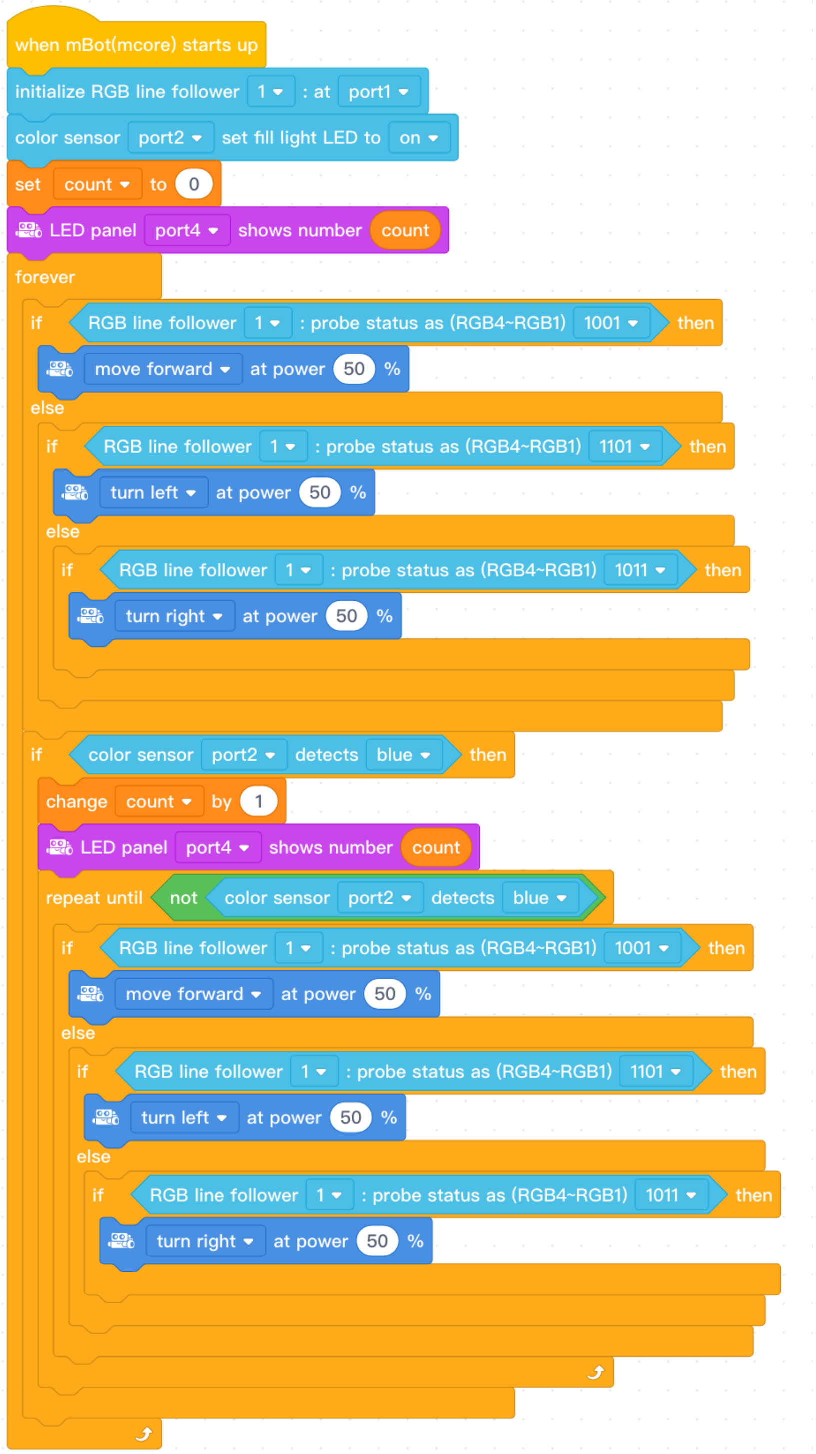
Once the blue card has been detected, a wait block is added for the appropriate number of seconds depending on the speed of the mBot, which is usually under 1 second. Despite its simplicity, the trade-off is costly. For this entire duration, the execution of the program halts, including the line following logic. The course of action right before the wait block will be carried out for this duration. This suggests that there is a high probability of going off track whenever the mBot passes through a blue card.



The third solution

why not replace the wait block with a repeat until block that resumes line following until the blue card has been past?

The insightful ones might have a quick fix for the method above. Instead of practically doing nothing for the wait duration, why not replace the wait block with a repeat until block that resumes line following until the blue card has been past? This indeed is a very reliable and rather easy solution to implement. The following program utilizes an operator block called 'not'. This is a logical gate block that, as its name suggests, negates the condition inside.

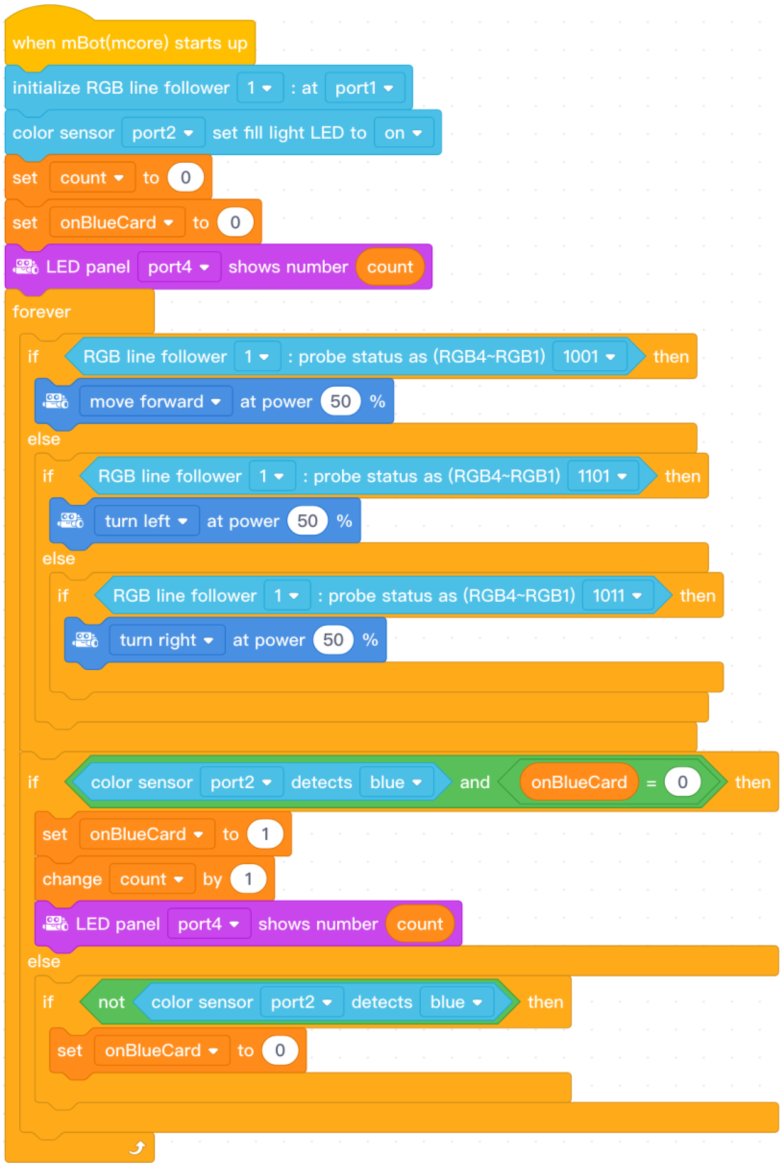


the final solution

A new variable named 'onBlueCard' is created.

Many would be understandably satisfied with the solution above. However, it is generally considered inelegant to have two segments of code with the exact same logic. Although the following final solution is not as straightforward at first glance, but it is elegant and introduces an important concept in programming – states. In this case study, the mBot could either be in the state of passing by a blue card, or in the state of not passing by a blue card. A new variable named 'onBlueCard' is created to represent these two states, where a value of 0 is equivalent to 'no' and a value of 1 is equivalent to 'yes'.

The initial state of 'onBlueCard' is 0, and the count should be updated when the state first changes to 1. While the state is in 1, the goal is to have the count not updated at all. And once the mBot passes the blue card, the state is changed back to 0 as well.



Part 3 Reflection

What is the initialization of a variable?

What is overcounting and what is your favorite solution?

What are states?

What is the most confusing part of this case study to you?

Advanced Line Following

10

Teaching Suggestion

Lesson 10 Advanced Line Following

Overview
25 mins

1. Introduce the lesson by adding new challenges about line following. - 10 mins
2. Let the students think about how to solve the new problems. - 10 mins
3. Show the content and learning objectives of this lesson. - 5 mins

Case Study
25 mins

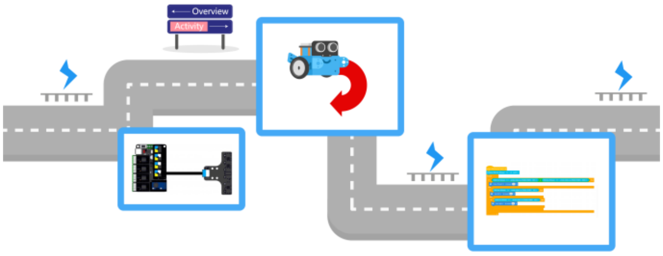
1. For the purpose of this lesson, first explain the turning radius to students. - 5 mins
2. Provide three programs for students to think about which one produces the largest and smallest turning radius. - 15 mins
3. Explain the importance of turning radius. - 5 mins

Techniques
65 mins

1. Four common but difficult situations about line following. - 5 mins
2. Program example for going through cross intersections. - 15 mins
3. Program example for turning at cross intersections. - 15 mins
4. Program example for stopping at Y-intersections - 15 mins
5. Program example for turning at Y-intersections - 15 mins

Reflection
5 mins

Part 1 Overview



This lesson delves deeper into some more advanced usage of the RGB line follower and various turning methods that go hand in hand with line following. As previously covered in the line follower lesson, the basic line following logic using either motor differential speed or sensor status works quite well on simple, uninterrupted tracks. On more complex tracks such as ones on the level one competition maps, certain tasks might simply be out of the basic method's depth.

On these tracks, the mBot could run into various types of crosses, intersections, or markings that exist to either purposely challenge or assist contestants. Knowing how to utilize these challenges as a helping hand is key to achieving more during competitions.

Objectives

Students can produce a complex line following program that are tailored to map features.

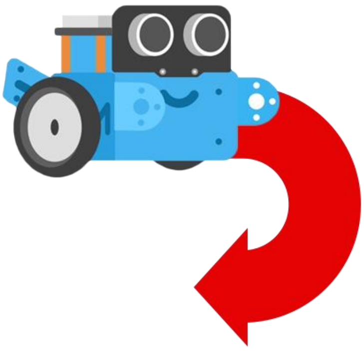
Students can understand the differences among three types of turning methods.

Students can appropriately utilize different turning methods depending on map features.

Students can identify, utilize, and resolve three common types of intersections.

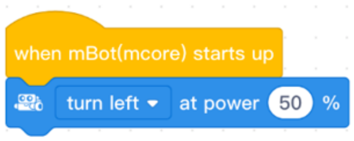
Part 2 Case Study

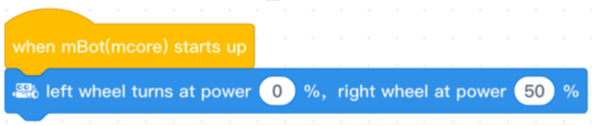
What is turning radius?

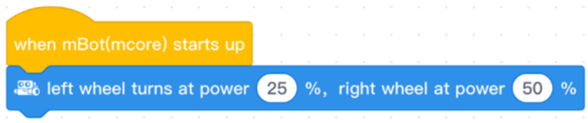


Turning radius or turning circle is a rather technical term that carries many different definitions in the automotive industry. For the purpose of this lesson, imagine the mBot is to make a circle, the turning radius is loosely defined as the radius of such circle.

Compare and contrast the following three types of turning methods, which one produces the largest and smallest turning radius?







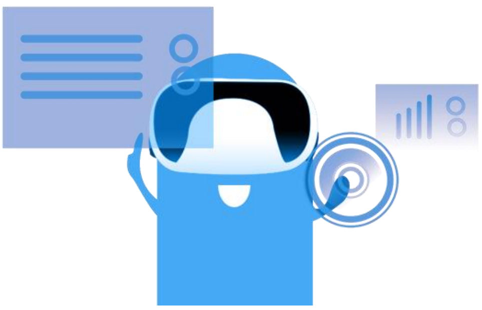
Why is turning radius an important topic?

Turning radius itself in an imaginary world is hardly important in any aspect.

However, in real world or especially in level one competitions, there are many constraints that limit the maneuvers of the mBot. Sometimes these are simply size constraints, other times it could be strategic, sensor, or efficiency limitations. In this aspect, using different methods to achieve different turning radii becomes a tool to better accommodate these constraints.

Part 3 Techniques

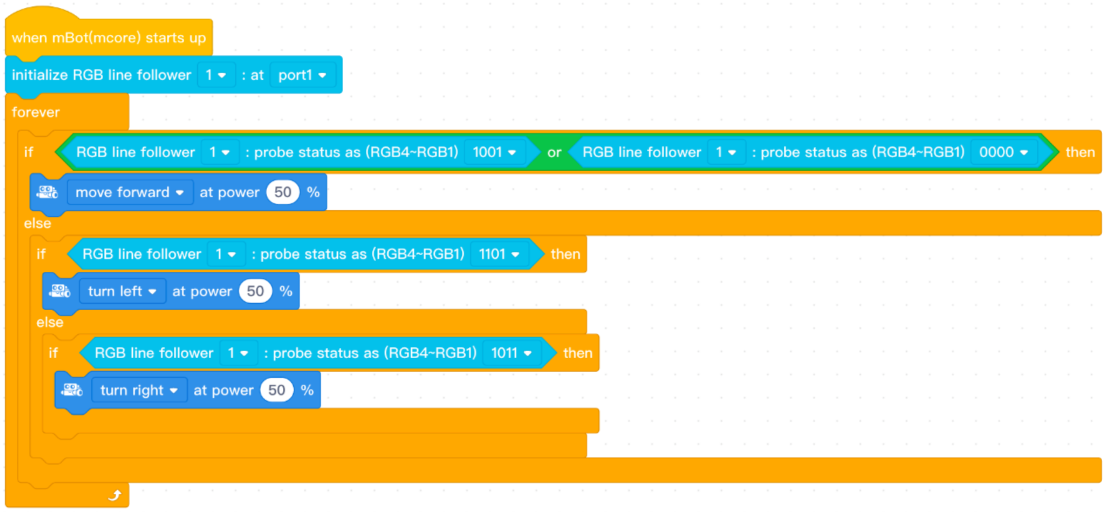
Advanced line following techniques



Now that students have been introduced to turning radius and three types of turning methods, it's time to take it to the track. In conjunction with the line follower, these turning methods affect the design of line following logic. Therefore, it is important for students to perform large number of tastings to develop a deep understanding their effects. In the following section, two common types of intersections will be introduced. Furthermore, additional constraints will be added to these intersections to challenge students to come up with the most reliable line following logic.

Going Through Cross Intersections

Going through a cross intersection is a rather simple problem. When passing a cross intersection, the sensor will run into a '0000' status. And all that needs to be done is to add an additional condition to the basic line following logic to also go forward when sensor status is '0000'. There are two ways to do this either with or without using logic gates.

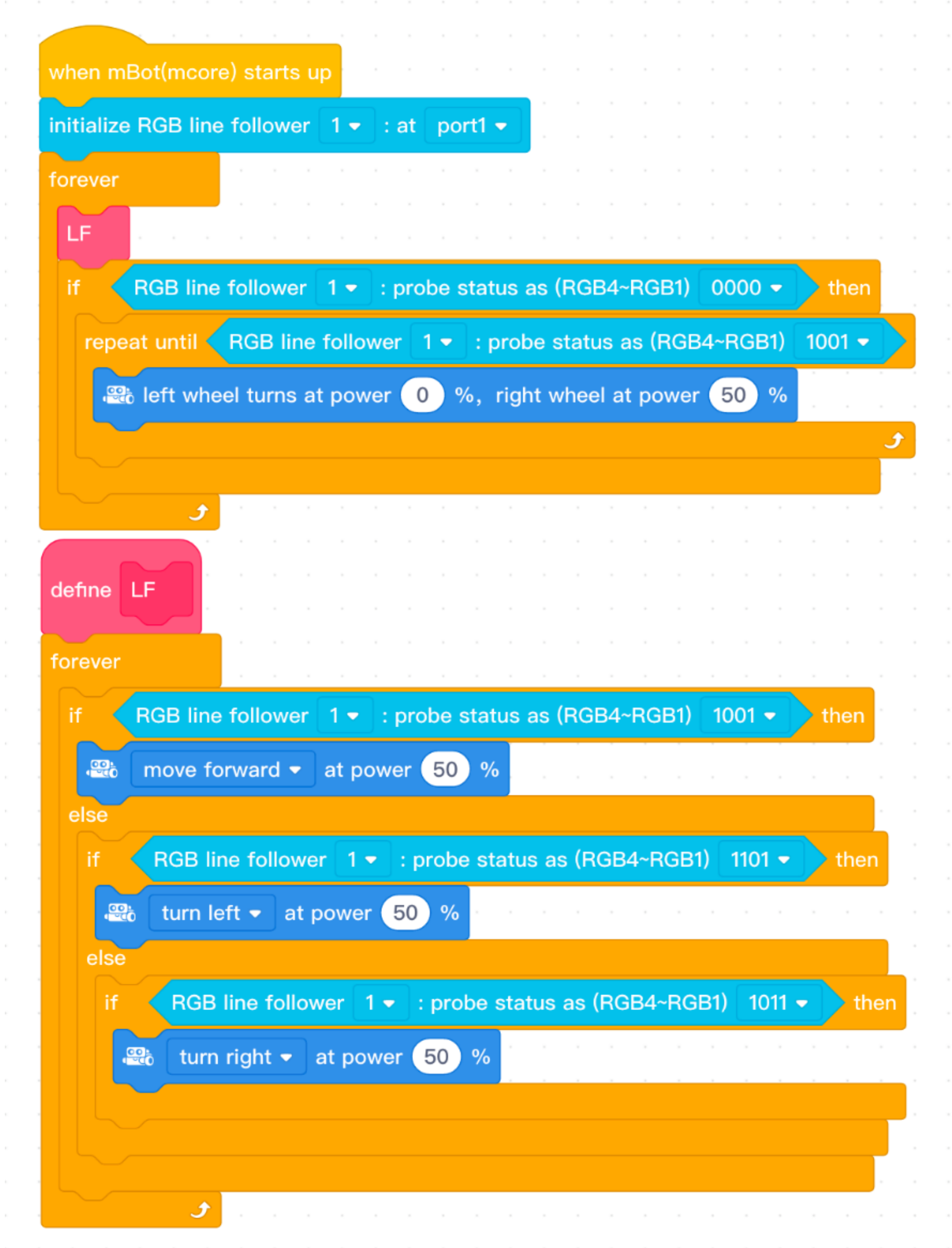


However, what would happen if the crossing is rather wide? Under this scenario, it would take the mBot much longer to go cross the intersection. And during this time period, it is likely the mBot wouldn't be able to maintain an absolute straight line due to friction, weight distribution, battery condition, and so on. Therefore, the longer it takes the mBot to cross over, the more off track the mBot would be as it reaches a non '0000' sensor status. Students would have to perform course correction depending on the sensor status as the mBot goes past the crossing.

Turning at Cross Intersections

Turning at a cross intersection presents an interesting issue which involves the choice of turning method and termination condition. It should be clear that unlike going through the intersection, an additional if condition wouldn't work since the turning wouldn't be considered complete until the follower is straight on the track again.

Since the turn at a cross intersection is 90 digress which is rather sharp, using the third type of turning method that produces the largest turning radius is ruled out. Despite the default turning method produces the smallest turning radius, it is so small that the sensor might never reach the '1001' completion status. What's left is the second turning method that fixes one wheel while turning the other where the fixed wheel becomes the center of the turning circle.



Stopping at Y-Intersections

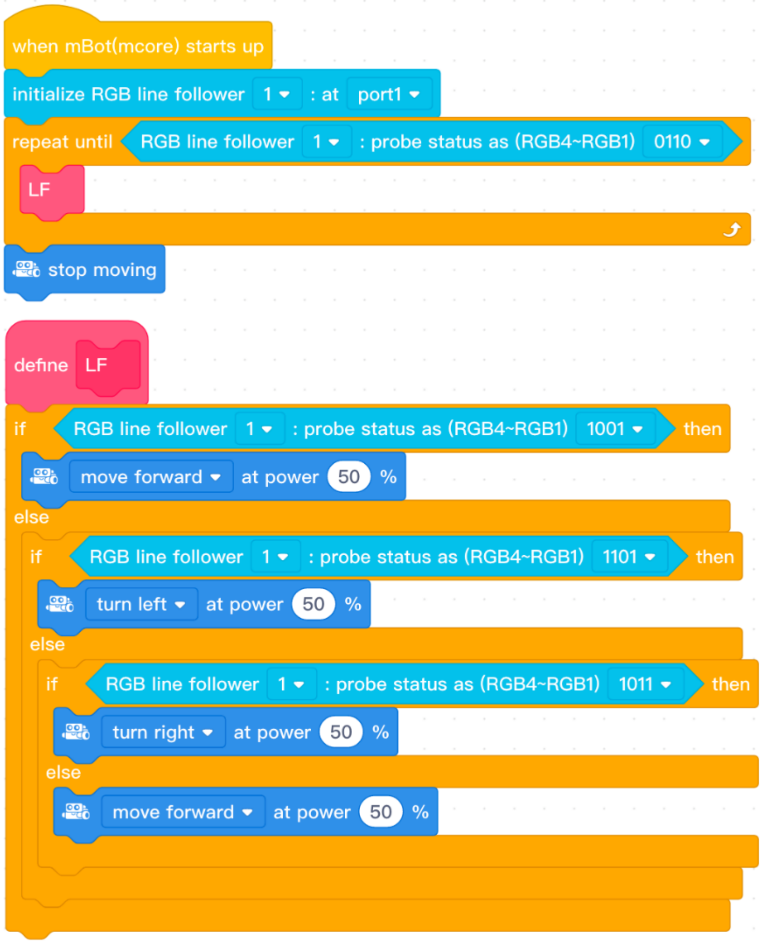
Y-intersections are more tricky compared to cross intersections just because they come in all shapes and forms. The split could be narrow or wide and the junction area could be small or large.

For this exercise, assume the junction area is not large enough for the sensor to read '1111' and that the split is just wide enough for the sensor to read '0110' as the mBot pass through the junction area.

Some might claim that the mBot

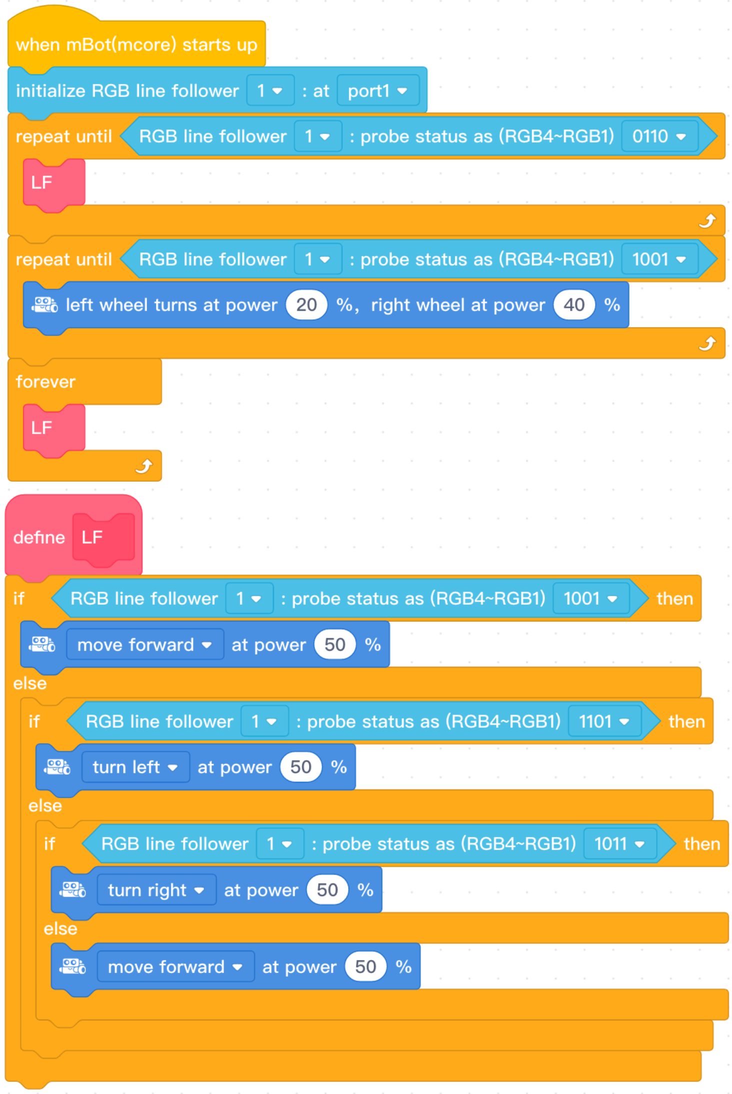
could simply turn at status '0110' but they ignore the fact that before the line follower could reach the '0110' status it first passes through some other potentially undefined status. And that is due to the form and shape of a Y-intersection.

Depending on the way the line following logic is designed, in most cases when an undefined status appears, the mBot resumes its previous course of action. Imagine the split second before the mBot gets to the junction area the sensor reads '1101'. The mBot would turn left as programmed. If the next sensor output, as the mBot is now on the junction area is undefined, then the mBot would keep turning left until the line follower reaches a defined status. Therefore, the correct logic is to let the mBot go forward in all undefined cases until '0110' is reached.



Turning at Y-Intersections

Now all that's left is to figure out the turning method and the completion condition for the turn. Although all three ways of turning could work, the simplest method is to let the mBot slowly merge onto the track until the sensor status reads '1001' again. Other turning methods could call for completion conditions such as '1111' or others, which are harder to deal with and requires more maneuvers to get straight back on track.



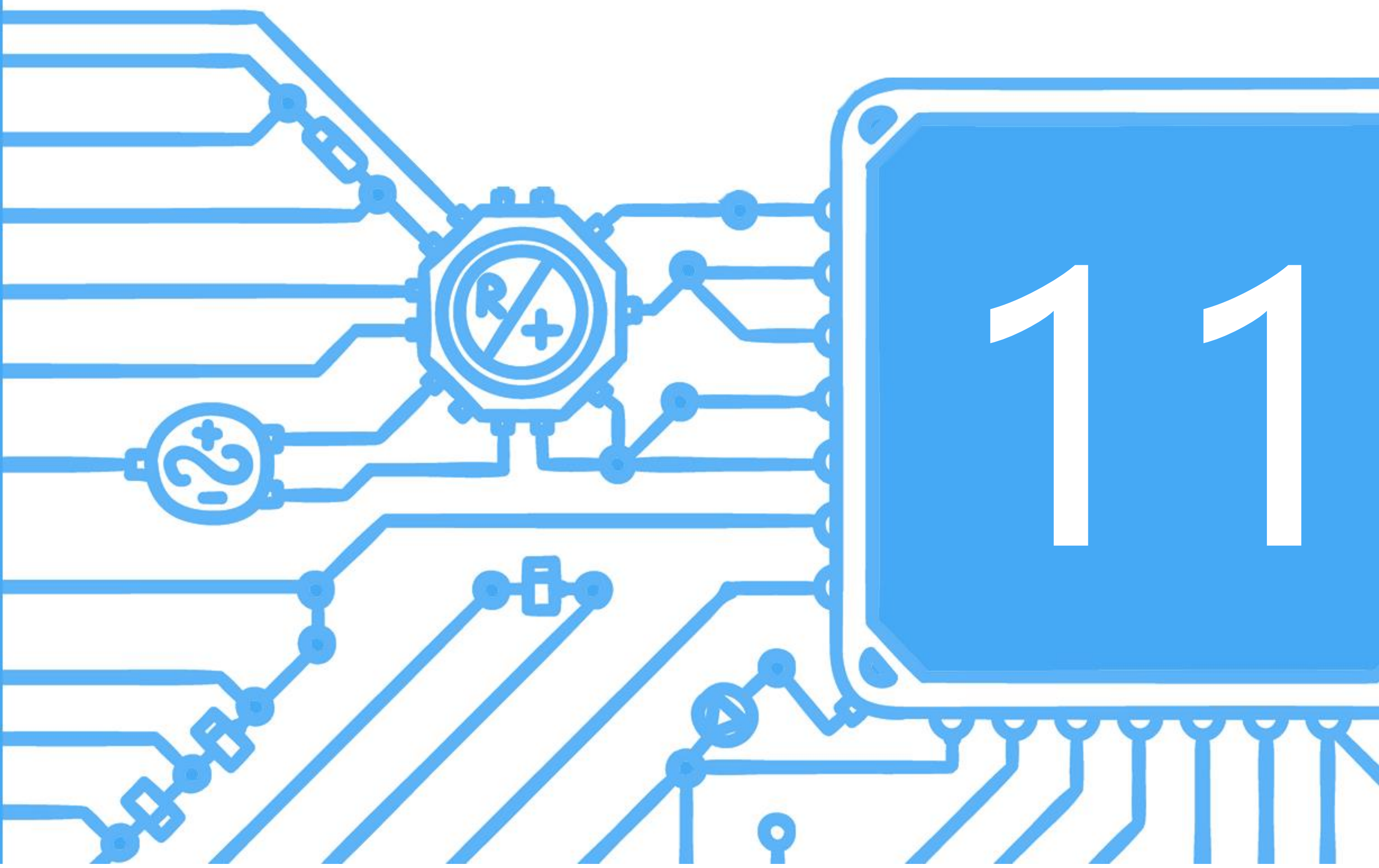
Part 4 Reflection

What are the most confusing topic for you in this lesson?

How does various turning radii affect the position of the mBot?

What would happen if the line follower runs into an undefined sensor status?

Advanced Programming Techniques II



Teaching Suggestion

Lesson 11 Advanced Programming Techniques II

Overview
20 mins

Mission introduction and rules explanation.
- 10 mins

Mission analysis and solution brainstorming.
- 5 mins

Learning objectives agreement.
- 5 mins

Case Study
55 mins

Let the students read the rules again and discuss how to complete the mission with teammates.
- 10 mins

Mission breakdown and workflow planning.
- 15 mins

Forming a formula concept for the relationship among speed, time and distance.
- 30 mins

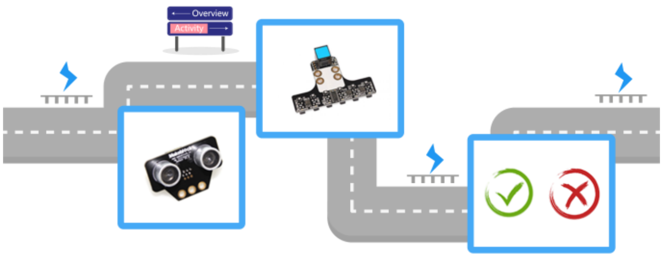
Programming
40 mins

Program
- 20 mins

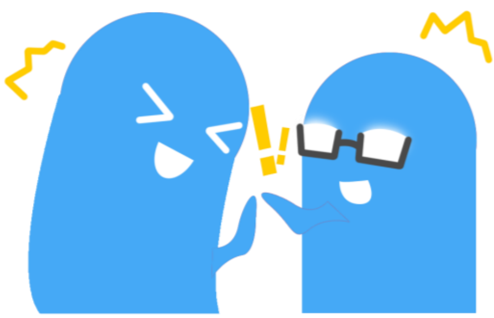
Case Analysis
- 20 mins

Reflection
5 mins

Part 1 Overview



In this lesson, students can try to interpret the task rules and split the workflow. The problems encountered in the objective physical environment are resolved and solved by program adjustment.



Objectives

I can understand the operational relationship between speed, time and distance.

I can calculate the time it takes for mBot to move 1CM.

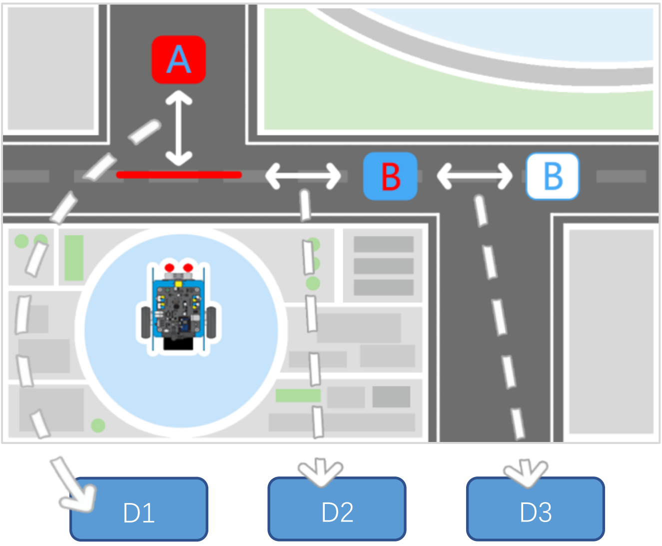
I can edit the program to record the sensor multiple times and return the value automatically, and the average calculation is done automatically.

I can consider the impact of the objective physical environment on the execution results of the program during the programming process.

I can make a reasonable workflow split for complex tasks.

Part 2 Case Study

Mission Introduction



In this mission, mBot starts from the starting point and needs to measure the distance from the red line to Block A. After that, turn to Block B and push it the same distance from the red line to Block A.

Mission Breakdown

Before we face any mission, we have to complete a detailed interpretation of the task rules and a breakdown of the task workflow.

After the students repeatedly interpret the rules, they try to complete the workflow decomposition of the task——

- **Step 1** - mBot move forward and stop at the red line.
- **Step 2** - Measuring the distance between the red line to the Block A and memorize it.
- **Step 3** - Turning right to find the Block B and stop to face it.
- **Step 4** - Measuring the distance between the mBot and the Block B.
- **Step 5** - Pushing the Block B to the same distance as the distance between the red line to the Block A.

Preparation

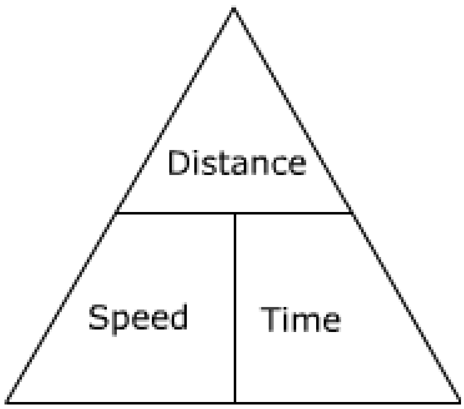
How long does the mBot need to move 1 cm?

Background knowledge

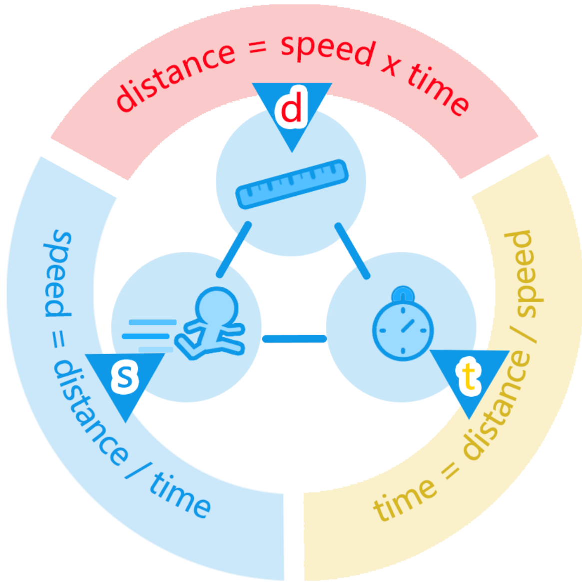
Distance Speed Time Formula

Speed is a measure of how quickly an object moves from one place to another. It is equal to the distance traveled divided by the time. It is possible to find any of these three values using the other two.

This picture is helpful:



The positions of the words in the triangle show where they need to go in the equations. To find the speed, distance is over time in the triangle, so speed is distance divided by time. To find distance, speed is beside time, so distance is speed multiplied by time.



s = speed (meters/second)

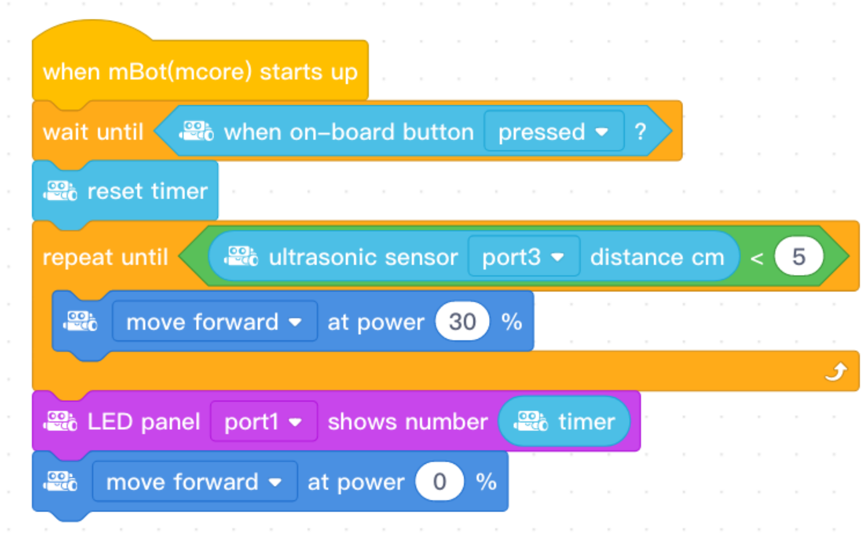
d = distance traveled (meters)

t = time (seconds)

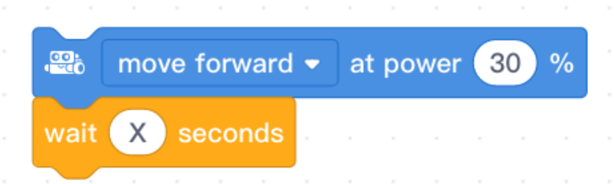
If we understand the above formulas for speed, time, and distance, then we can have a variety of methods to test and calculate the time it takes for mBot to walk 1cm. We can use the ruler and stopwatch, use mBot to test the parameters at the same time, and finally use the calculator to calculate the unit time used. This method is feasible, but since it is considered to be an operation at the time of timing, there may be a large error.

In the example below, we are only

providing a possibility. We placed a 15 cm position in front of the mBot with an obstacle of appropriate height and a line at a distance of 5 cm from the obstacle. Set the movement speed of the mBot to 30%, judge the stop by the returning value of the ultrasonic wave, and use the onboard timer to complete the 10cm travel time. Record the results of each time record, then add and divide by the test to get an average of 10 cm, and divide this value by 10 to get an average time of 1 cm.



Since many objective reasons, we only use "X" to represent the time when the mBot moves 1cm away at the speed 30%.

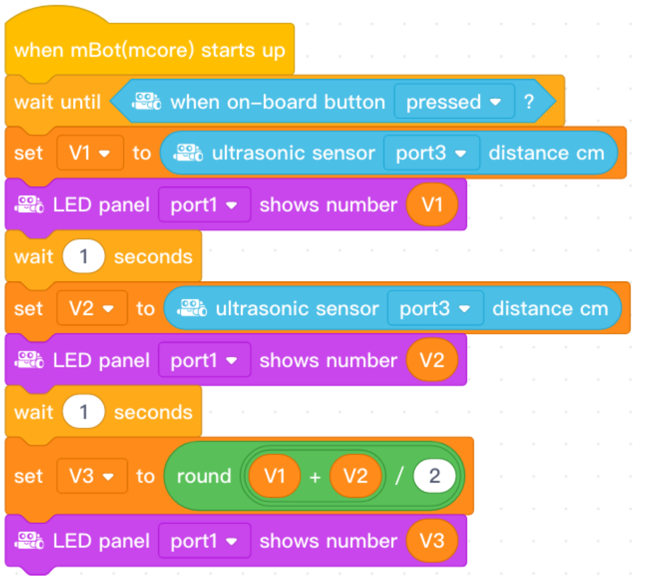


Part 3 Programming

How can we make our sensors return values more accurately?

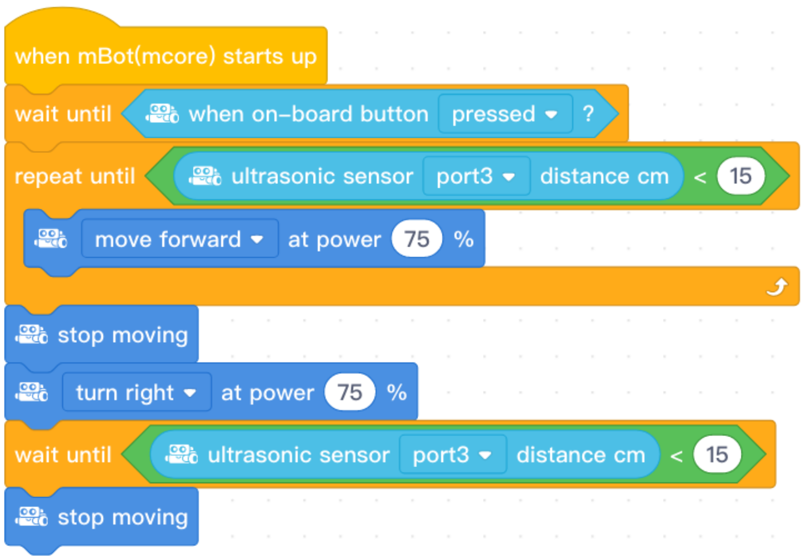
Here we use the multiple values collected to calculate the average value. Is it possible to use more programming methods to accomplish the functions similar to the sample program?

- **V1**= the result of the first test of the distance
- **V2**= the result of the second test of the distance
- **V3**= the average of the distance



Can this sample program complete the mBot move forward and stop on the red line? Can students find some bugs in this sample program?

Because we need to install the functional structure in the front of the mBot, here we extend the distance between the red line and the Block A to 15cm.



Did we use the same motor power value as the mBot move 1cm time program?

Whether to consider the larger forward displacement caused by the inertia of the car due to the increased speed?

Have you considered the effect of the width of the front obstacle on the steering conditions?

How can we turn mBot to find Block B?

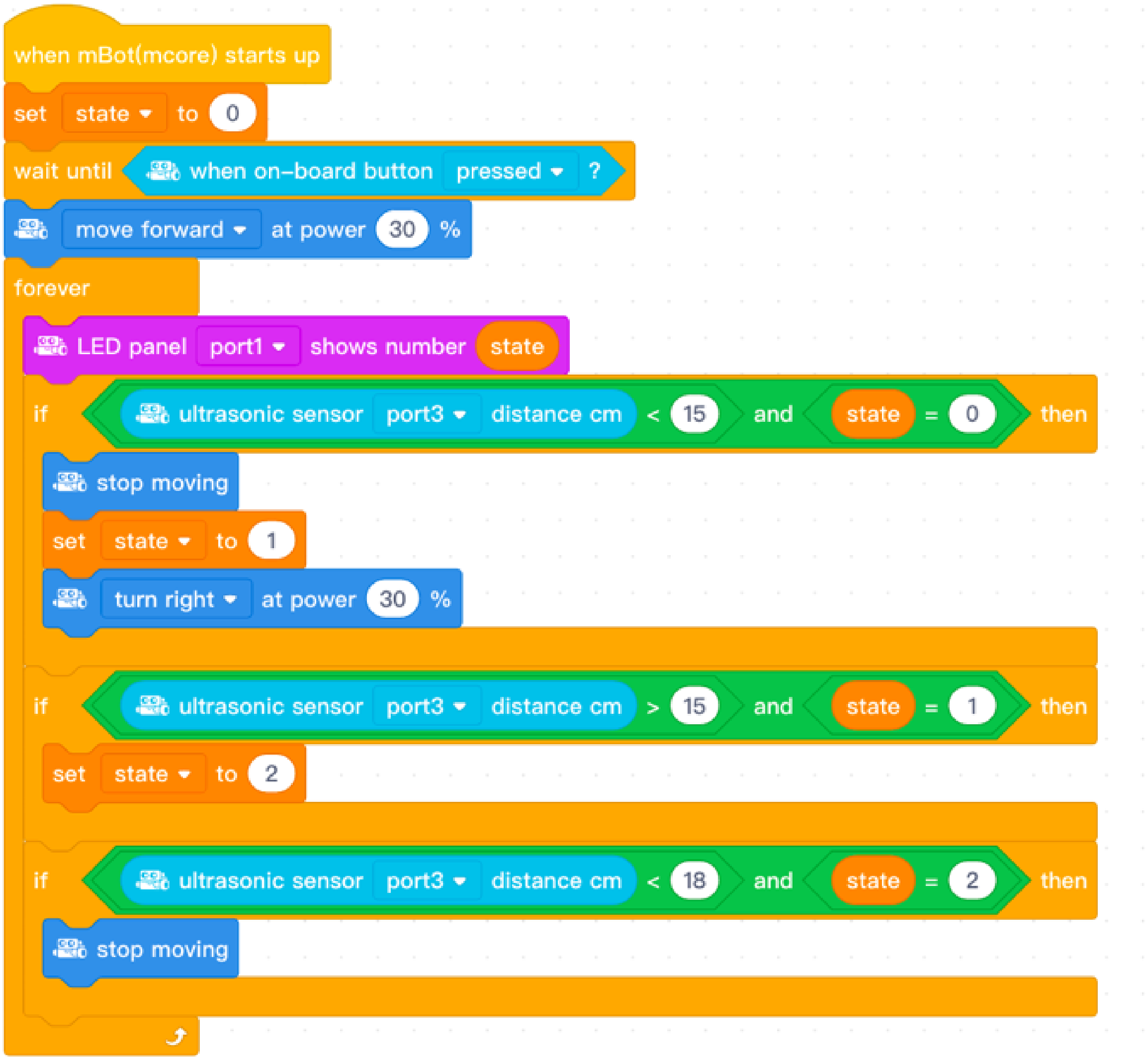
```
when mBot(mcore) starts up
wait until < when on-board button [pressed] ? >
repeat until < ultrasonic sensor [port3] distance cm < (15) >
    move forward at power (30) %
end
stop moving
wait (1) seconds
turn right at power (30) %
wait until < ultrasonic sensor [port3] distance cm > (15) >
turn right at power (30) %
wait until < ultrasonic sensor [port3] distance cm < (18) >
stop moving
```

When the first look at this question, most people will feel not very complicated. We can simply use time to control the moving forward distance or the angle of steering. However, in the extremely complex environment of such objective factors on the field, any program must ensure its stable enforceability.

- After waiting for the trigger button to start, the mBot advances at a speed of 30% until the distance to the obstacle is just less than 15 cm.

- Steering to the right with 30% power until the distance is greater than 15, continuing to turn until the distance is less than 18 stops.

Can we still complete this task without "wait until"?



In this program, we did not use "wait until" to get basically the same control results? Can students try to analyze whether the procedures in this paragraph and the above procedures have their pros and cons

Case Analysis

V1 - The result of the first test of the distance.

V2 - The result of the second test of the distance.

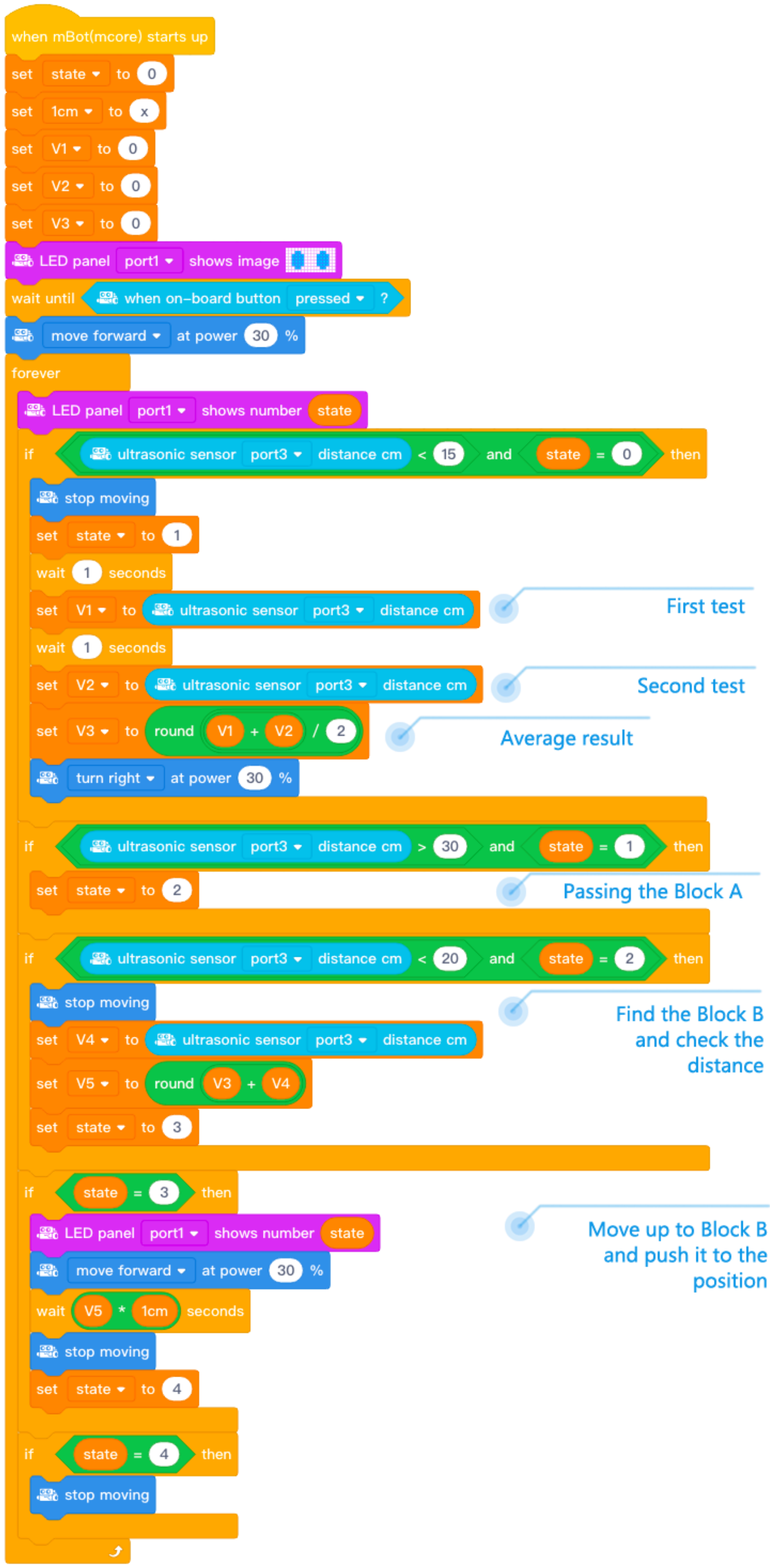
V3 - The average of the distance.

V4 - The result of the distance between mBot and Block B.

V5 - The distance mBot need to move to push the Block B.

State - mBot procedure state.

X - the time of mBot to move 1 cm.



Part 4 Reflection

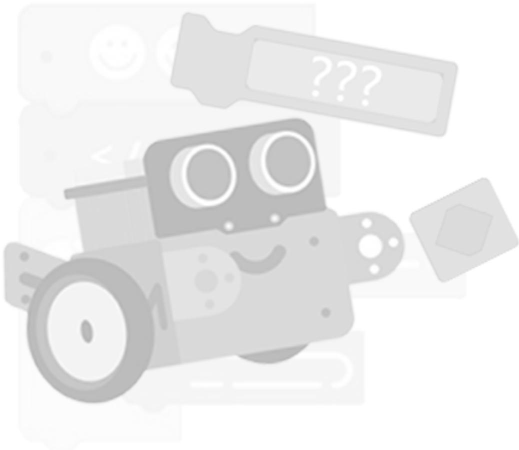
Which part of the program do you think is the most difficult part to understand?

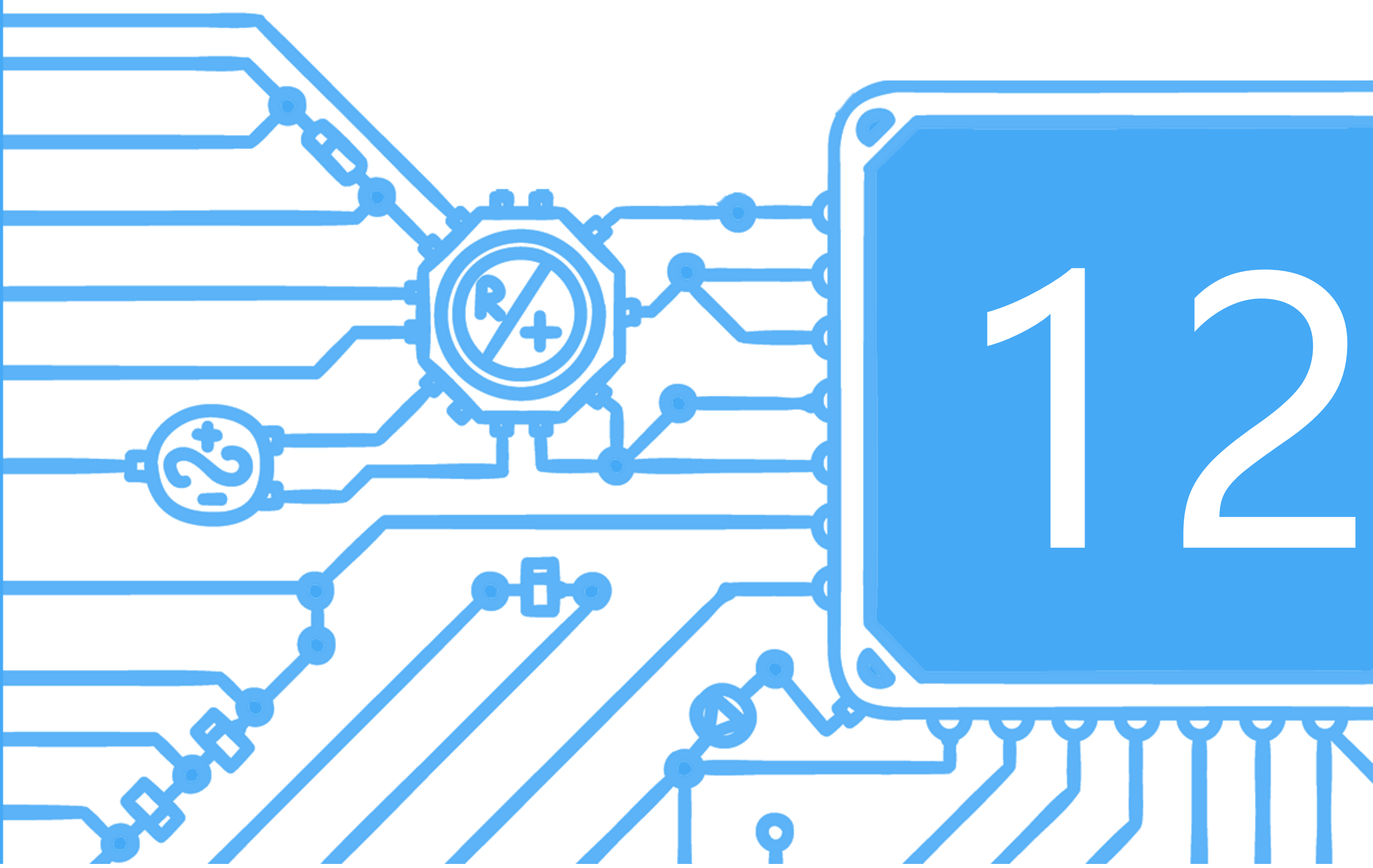
Can the way of the sample program be applied to other situations?

In the machine-controlled program, besides considering the logic of the program, what other factors should be considered?

What do we need to do between getting the mission and programming?

Program Picker





Teaching Suggestion

Lesson 12 Program Picker

Introduce the lesson by announcing a new working model. - 10 mins

Let the students think about how to write the program together in the new working model. - 10 mins

Learning objectives agreement. - 5 mins

Overview
25 mins

Custom blocks analysis. - 15 mins

Show the usage of the custom blocks with examples. - 30 mins

Have students try on their own. - 25 mins

Mechanisms
70 mins

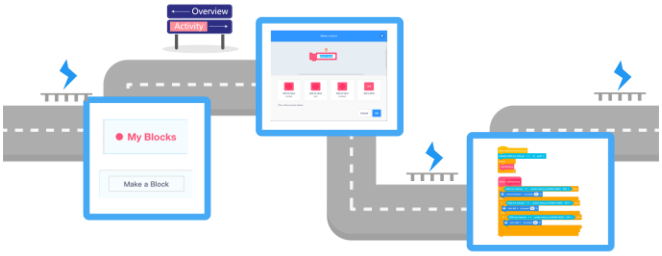
On- board button and its common mistake. - 10 mins

Timer. - 10 mins

Key Programming Topic
20 mins

Reflection
5 mins

Part 1 Overview



In the lesson, a special utility program called the program picker is introduced. Due to the nature of level one competitions, being able to pick and choose which automatic mission to carry out during a restart is almost a necessity for all contestants. The program picker alone is neither a particularly interesting nor a challenging program. Taking a closer look however, the mechanisms that must be in place in order for the program picker to function well are thought-provoking.

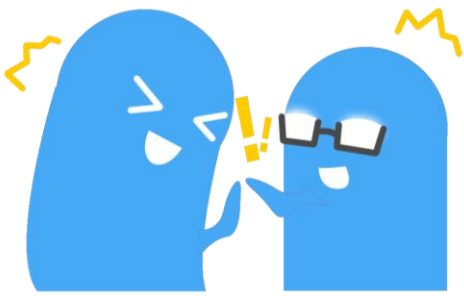
Objectives

Students can reasonably structure their program in blocks.

Students can describe what parameter is.

Students can utilize the on-board button.

Students can produce a program picker with a reliable method of selection.



Part 2 Mechanisms

Introduction

What are custom blocks?

A program picker, as its name implies, that there must be more than one program to choose from. These programs are called custom blocks in mBlock 5. The usage of custom blocks introduces a new way of thinking. Before this lesson, students wrote a single program to complete a certain task or mission. If the task or mission is rather complex and involves many different maneuvers, sensor logics, keeping track of variables, states, and more, this single program can easily become enormously lengthy. This way of programming has two major issues – first is low readability, second is low re-usability.

Why we use custom blocks?

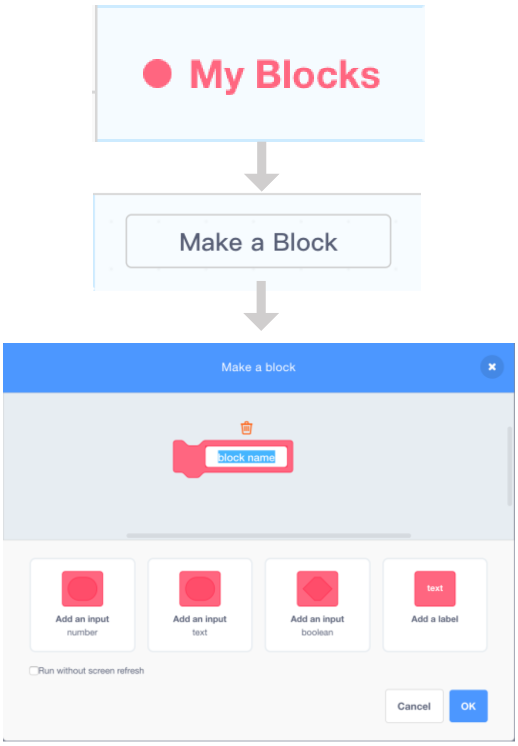
- Program readability is important because it is rare that a single programmer composes an entire program by him or herself. MakeX encourages teamwork where every contestant contributes to all aspects of the game. Therefore, a program is usually shared amongst several programmers to be worked on at different times. Low readability results in longer time for other programmers to understand the program. And since a program must be understood before it's improved upon, this leads to extreme inefficiency.

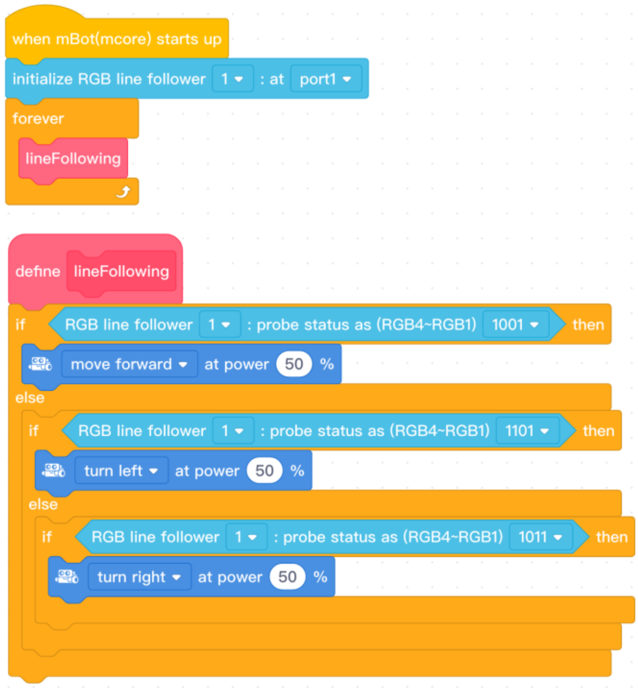
- A single long program is hardly reusable. Such programs are usually extremely specialized, designed to perform one complex task or a series of smaller tasks in series. Since the execution flows from top to bottom until there is no more instructions to be executed, if only parts of a program need to be used elsewhere, the programmer must pick out those parts and rewrite another program. This brings up the topic of program separation that leads to the reason why custom blocks are needed.

How to use custom blocks?

Think of the basic line following logic that has been covered many times in previous lessons. Since line following is at the heart of level one competitions, it is almost utilized in all missions during the automatic stage. For this reason, it could be thought of as a standalone component with a specific functionality. And if a custom block is created for this functionality, a reasonable program separation has been achieved.

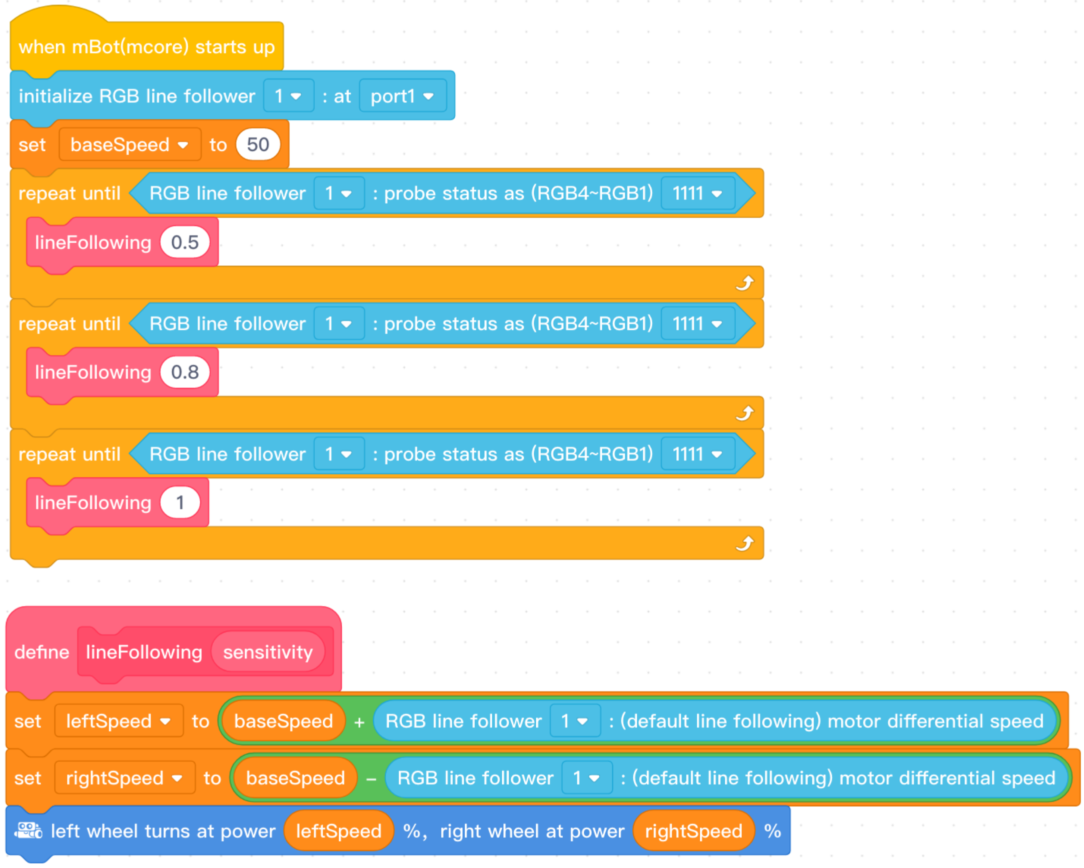
To do this, simply go under the 'My Blocks' section and click on 'Make a Block'. Now there's the option to name the custom block and add inputs to the block.



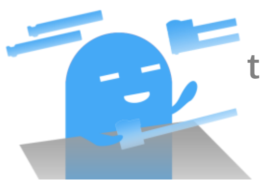


Inputs are sometimes more officially called arguments or parameters, provide more flexibility and room for customization to programmers.

Suppose the objective is to create a custom block that uses motor differential speed to perform line following. As mentioned in the lesson on RGB line follower, motor differential speed goes hand in hand with turning sensitivity. It is a reasonable assumption that for different missions or different sections of the track, different turning sensitivities are desired. And parameters make this possible. The following example program dynamically changes the turning sensitivity.



To summarize, it is good practice to look for either repeated instructions or instructions that carry out certain reusable functionalities and group them into custom blocks. These blocks can have parameters that could be used to do further customizations.

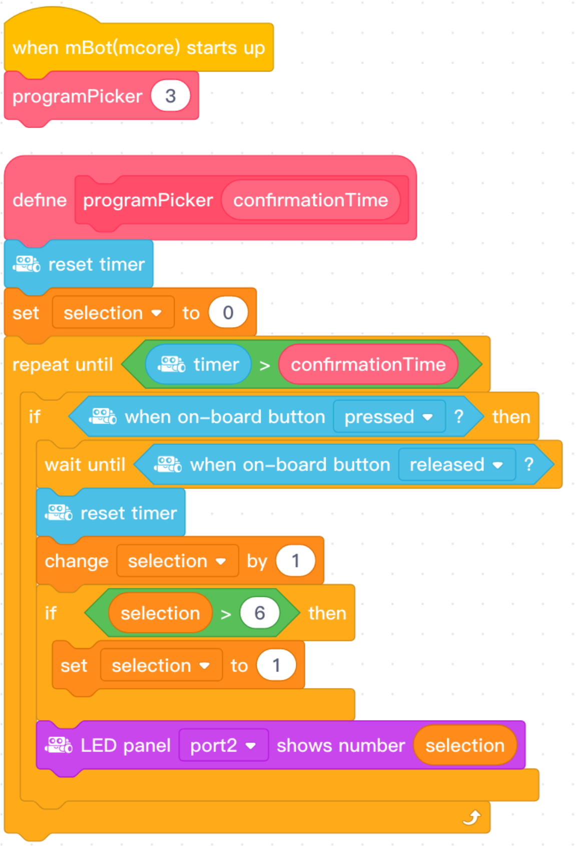


Now it's time to think about how to perform program selection!

For the purpose of demonstration, assume six custom blocks are made each designed to carry out its own mission and that the on-board button is used to perform the selection process. It is worth mentioning that using the on-board button is not the only option. In fact, almost any sensor could be used to perform this task.

Since there are six programs in total, a variable could be used to keep track of which program is currently selected.

In the example program below, a variable called 'selection' is created for this purpose. And each time the on-board button is pressed changes the selection. Once the desired program is selected, the method used to confirm the selection is where students can get very creative. Perhaps the simplest and most intuitive solution which is implemented in many modern electronics is to set a 'confirmation time' where if the button hasn't been pressed for a certain number of seconds, the selection is confirmed.



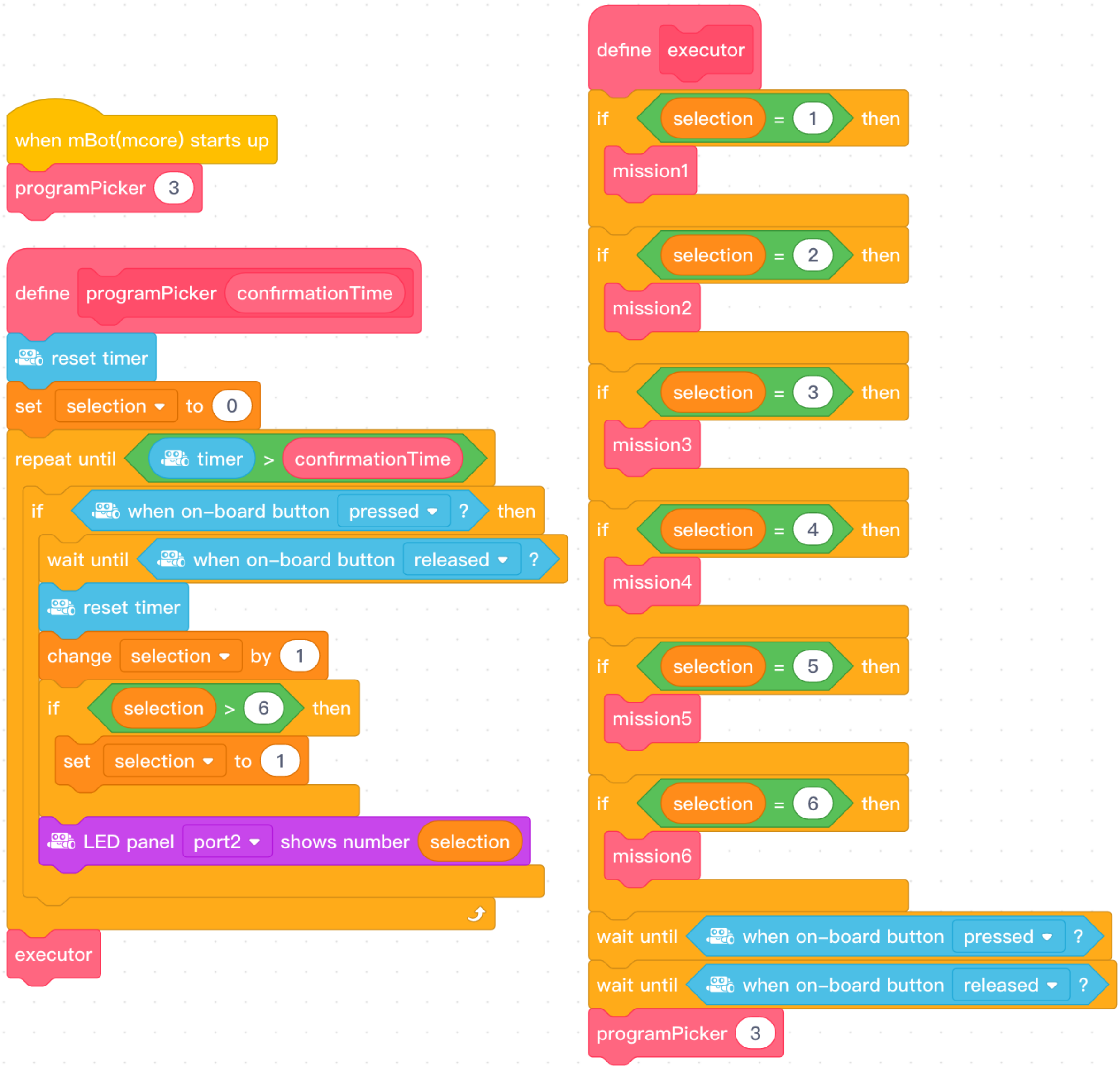
There are two caveats to note –

First is that what happens if the selection variable goes past the total number of programs?

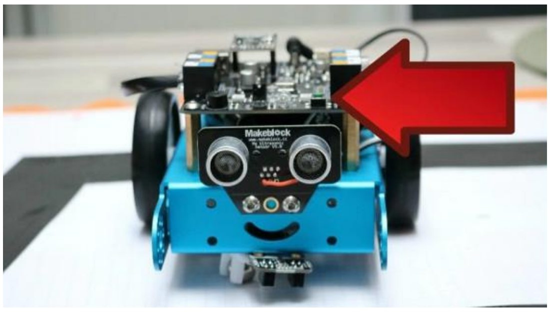
To address this issue, an if statement could be added to reset the selection variable back to '1', this creates what's called a rotating count. This method has the advantage that if the button has been accidentally pressed and the selection goes past the desired program number, user could simply keep pressing the button to rotate back.

Secondly, setting a confirmation time means using a timer block. Each time the button is pressed, the timer must be reset so that each selection receives the full duration of confirmation time.

And finally, once the selection has been confirmed, another custom block called the 'executor' is used to carry out the selected program. This program is essentially a series of if statements stacked together.


when mBot(mcore) starts up
programPicker 3
define programPicker confirmationTime
reset timer
set selection to 0
repeat until timer > confirmationTime
if when on-board button pressed ? then
wait until when on-board button released ?
reset timer
change selection by 1
if selection > 6 then
set selection to 1
LED panel port2 shows number selection
executor
define executor
if selection = 1 then
mission1
if selection = 2 then
mission2
if selection = 3 then
mission3
if selection = 4 then
mission4
if selection = 5 then
mission5
if selection = 6 then
mission6
wait until when on-board button pressed ?
wait until when on-board button released ?
programPicker 3

On-board button



When using the on-board button, there's a common mistake of not using the 'button released' block. Despite not a particularly severe mistake, it is logically incorrect to not do so. In natural language, pressing the button implies first pressing down the button and then releasing it. If only the 'pressed' block is used and put into an if statement, for the very short period of time that the button is pressed down before it's released causes the if statement being satisfied many more times than expected resulting in unreliable programs. That's why in most cases students are encouraged to use the 'pressed' block and 'released' block in pairs.

Timer

The timer is started the moment when the mBot is turned on. It counts up in seconds and could be reset any time during program execution. Each time the timer is reset, it counts up from 0 second again.

Part 4 Reflection

Why is program picker important for level one competitions?

What are some methods of selection confirmation that you can think of?

What are some caveats related to the program picker?

What is misleading about saying ‘press the button once’ in natural language?