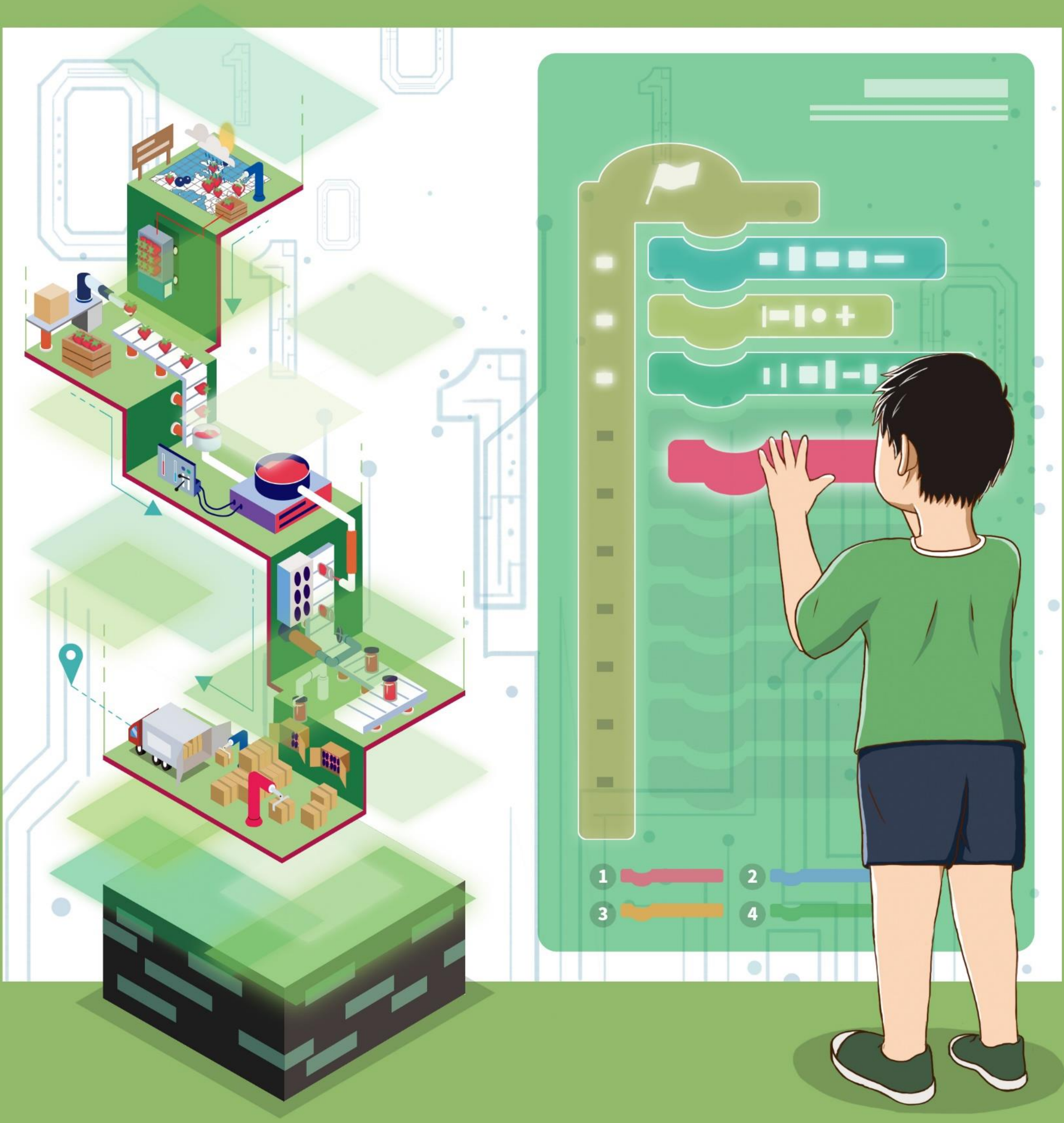
Graphical Programming & Robots



Contents

Chapter 1 Introduction to Programming
- John's Smart Supermarket

- John's Smart Supermarket

The rapid development of science and technology has made artificial intelligence (AI) gradually become part of our daily life. Now we can see AI products everywhere, including smartphones, smart watches and smart cleaning robots. John always wants to open a supermarket. He particularly expects that his supermarket can provide all kinds of AI products to serve customers. But he does not understand what instructions enabled these AI products to complete specific work. Through survey and consultation, John knows that engineers can program AI products to direct them to complete work. Programming not only helps AI products complete work, but also makes them smarter. But what is programming?

In this chapter, we will understand programming software, learn how to program, and thus start a programming journey!

Learning Target

- ☼ Students will understand DobotBlock software and its interface.
- ☼ Through writing and running programs, students will develop their interest in learning how to program.

Introduction to Programming

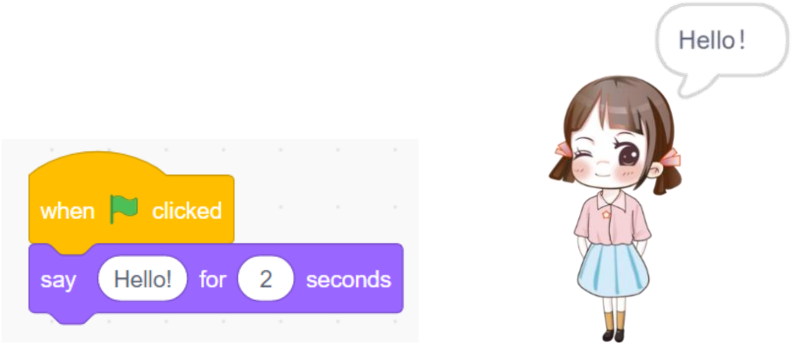
- Smart Assistant

I plan to open a smart supermarket. I want to complete some of my work by programming. Kelly, have you heard of programming?
It's great! Well, I have. Many students around me have learned it. But I cannot program. Let's study it together!

1.1.1. Introduction to Programming Software

Boys and girls, to learn programming well, we must grasp some handy software tools and effective learning methods. Generally, we must write computer instructions with a programming language. John invited the programming expert Mr. Lee. According to Lee's introduction, John knows that the programming software in this textbook is DobotBlock.

As a type of simple graphical programming software, DobotBlock is easy to operate. By dragging some graphical blocks and combining them according to a certain logic, we can finish programming. For example, to have Kelly greet you by saying "Hello!" on the stage, what blocks should we drag and combine? We just need to combine two blocks, see Figure 1.1.



```
when 🏁 clicked
say Hello! for 2 seconds
```

Program

Running Result

Figure 1.1

In Figure 1.1, the yellow block and the purple block mean instructions, and the girl Kelly is a set sprite. We can randomly set this sprite. It can understand and execute the instructions from the yellow block and the purple block in the program.

After we select a sprite and write a program by drawing blocks, we can click the green flag button to run the program. Then, the purple block will instruct Kelly to show "Hello!" in the bubble form.

Let's think about it

✧ Where can we find the block instruction and the sprite in the programming software?

1.1.2. Programming Software Interface

Let's find it

✧ Find and double-click DobotBlock on the computer, and now, we come to the interface; watch carefully, and do you find anything new?

There are Menu, Stage, Device, Sprite and Stage List, Block Area, Coding Area and Tabs on the DobotBlock interface, see Figure 1.2.

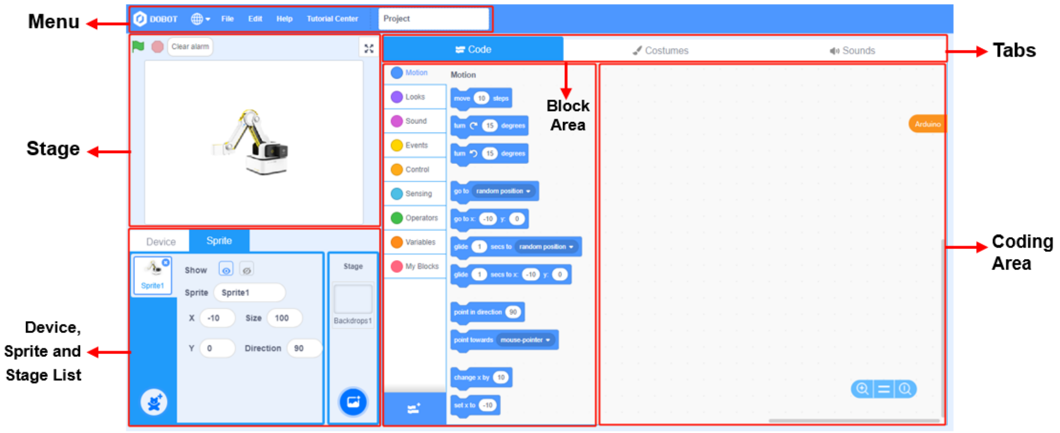


Figure 1.2

Let's do it

✧ Freely click icons on the interface, read and share them with others.

(1) Menu

We can see File, Edit, Help and Tutorial Center on the DobotBlock interface; each of them has different menu items for different functions. For example, there are New, Save, Load from your computer, Save to your computer and Recent under File, see Figure 1.3.

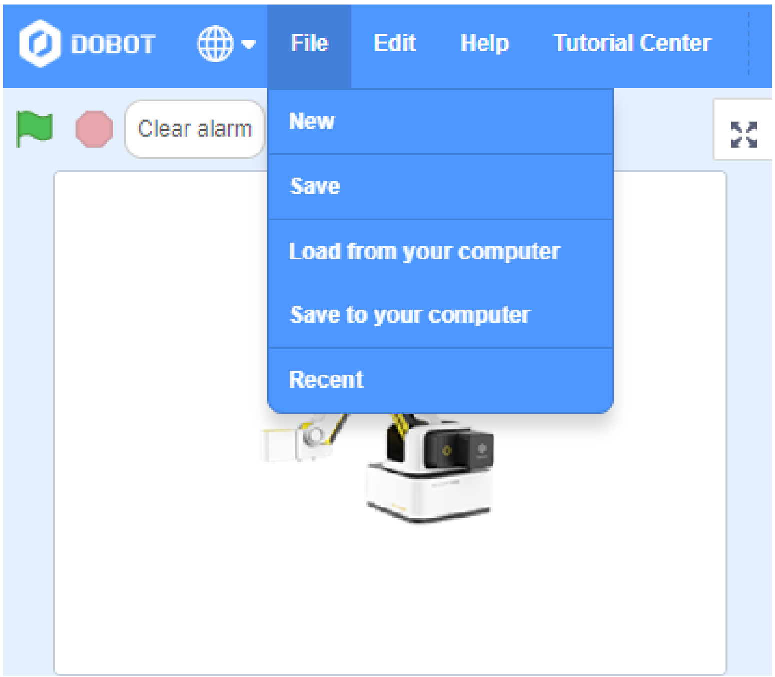


Figure 1.3

(2) Stage

We can move or interact sprites (or devices) on the Stage, see Figure 1.4.

Do you still remember Kelly? She said hello on the Stage. Yes, we can move or interact sprites (or devices) on her Stage, see Figure 1.4.

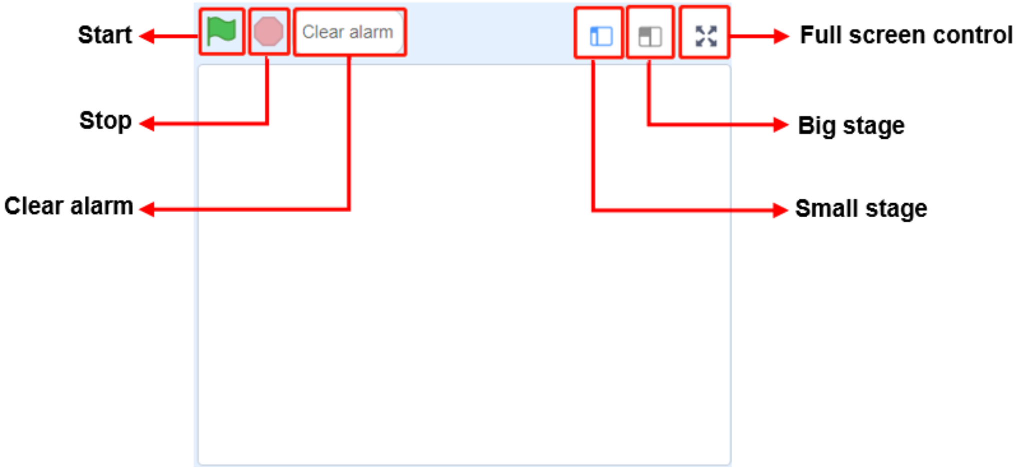


Figure 1.4

We can see 6 icons on top of the Stage. Boys and girls, let's guess which is Start and which is Stop on the left? What do the three stage modes mean on the right?

Let's read it

Stage Icons		
Icon	**Name**	**Function**
⚑	Green flag button	It starts the program.
⬣	Stop button	It stops the running program.
Clear alarm	Clear alarm button	As the connected device gives an alarm, we can clear the alarm by clicking this button.
◧	Small stage mode	It scales down the stage in the software interface.
◩	Big stage mode	It scales up the stage in the software interface.
⛶	Full screen control mode	It hides all scripts and programming tools, and enlarges the Stage to the full screen mode.

(3) Device List

By default, a new item includes a device. For the device list, see Figure 1.5.

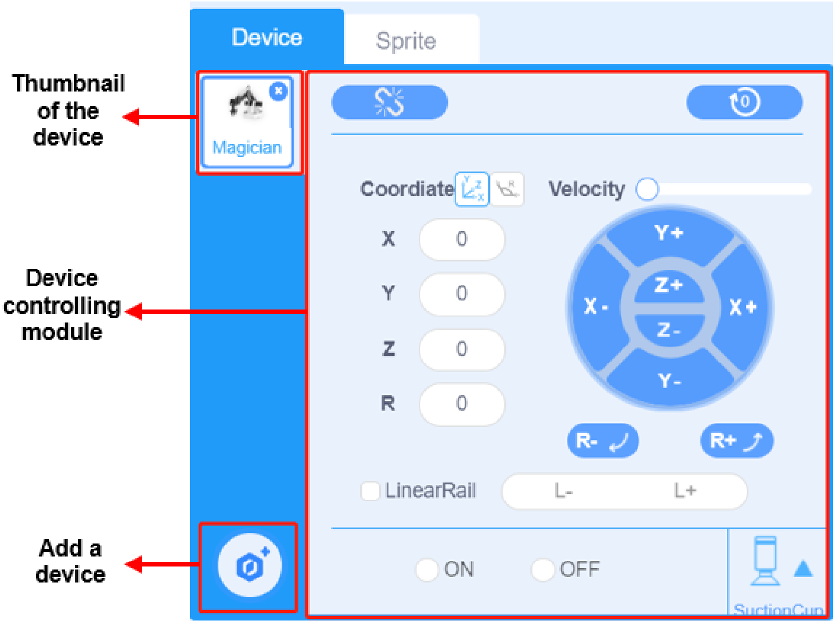


Figure 1.5

Let's read it

Device List Icon		
Icon	**Name**	**Function**
	Thumbnail of the device	Every time we add a device, the corresponding device thumbnail appears.
	Add a device	Click this button, and we can come to the device warehouse, and select the device we want to add.
	Device controlling module	We can see connection control, home, motion control, Linear Rail, end control and so on in this module.

(4) Sprite List

By default, a new item includes a sprite. For the sprite list, see Figure 1.6.

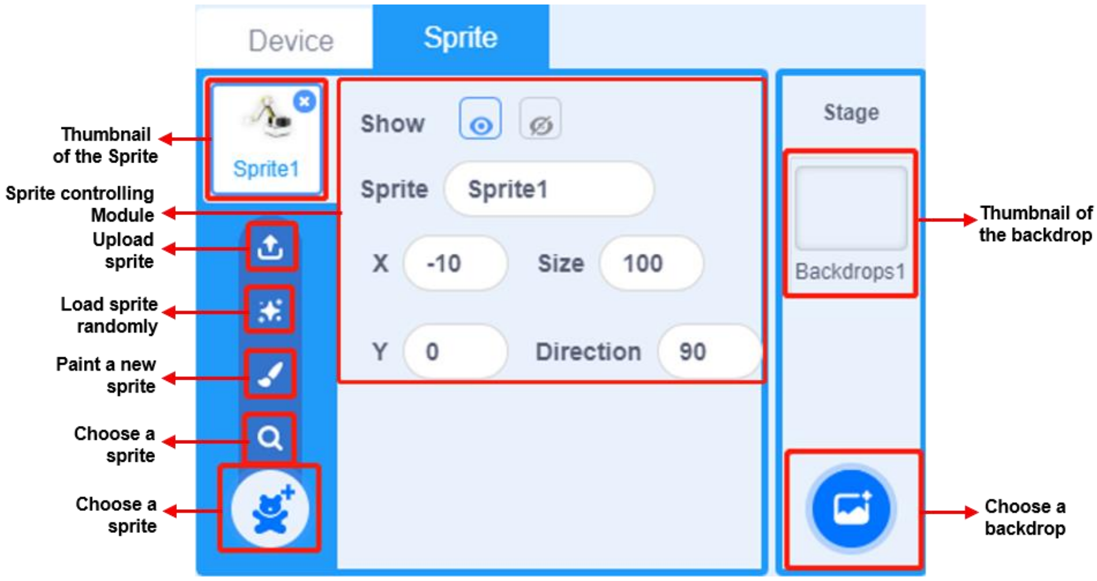


Figure 1.6

In the sprite control module, we can show or hide this sprite on the stage, or modify its name, or change its position by modifying X and Y values, or set its size and direction. For example, Figure 1.7 shows the sprite control module for the sprite "John (1)".

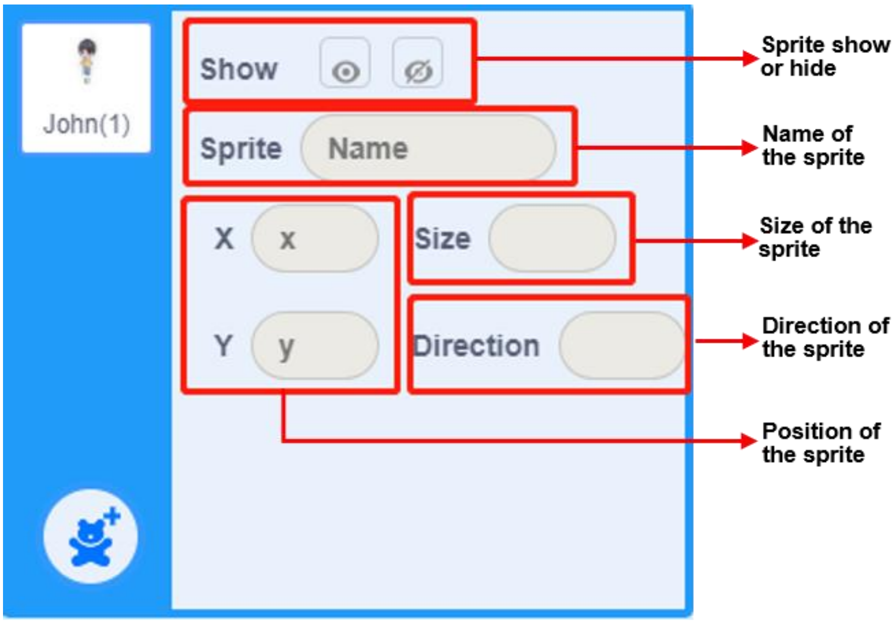


Figure 1.7

Let's do it

- ✧ Click the Sprite List in the software, and try adding a sprite "John (1)".
- ✧ Modify the name of the sprite name to "John", and set its size to 60, change the X and Y values to 0, and the direction to 0. Watch and describe the change in the sprite on the stage.

(5) Block Area and Coding Area

As shown Figure 1.8, the Block Area is on the left, and the Coding Area is on the right.

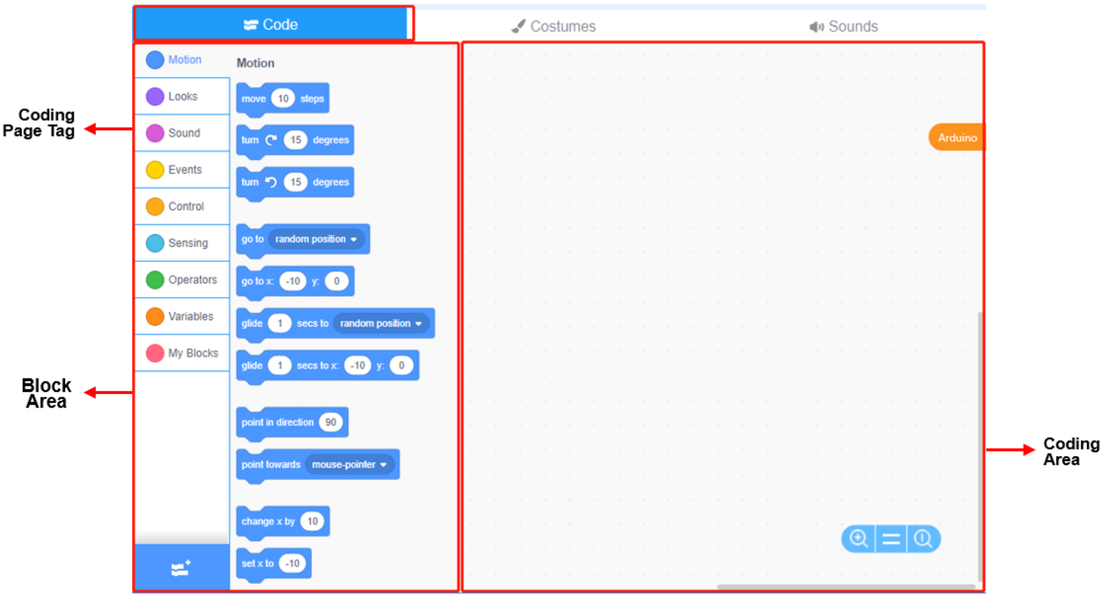


Figure 1.8

Block Area: This area provides blocks necessary for programming, and we can search for our desired blocks by class and color.

Coding Area: This area is for programming; we can drag blocks to this area to program.

From the block area in DobotBlock, we see different types of blocks and the Add Extension button, and different blocks in different modules. A module includes blocks with the same color, and different modules include blocks with different colors. We can add different extension modules by clicking Add Extension, for example, AI extension, music and pen modules.

As we use the sprite, the block area is divided into nine modules. Of them, the Motion module includes blue blocks, and the Looks module includes purple blocks, see Figure 1.9.

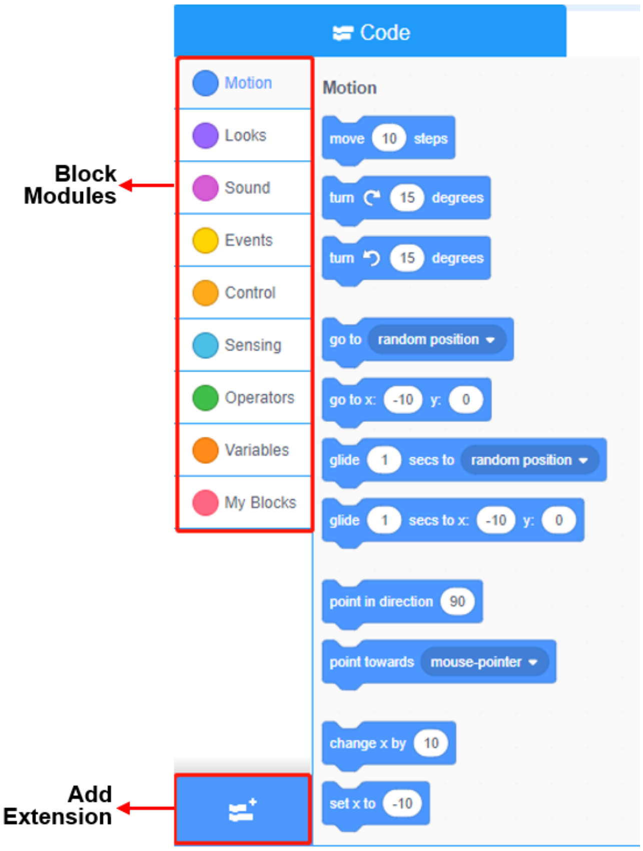


Figure 1.9

To create a program, first drag our desired blocks with a mouse to the coding area from the block area, and according to needs, then connect and combine these blocks just as we play puzzles. These connected and combined blocks are called scripts.

Let's do it

✧ After adding the sprite "John", find the "when the green flag clicked" block in the Events module and the "say 'Hello!' for 2 seconds" in the Looks module, and connect and combine these two blocks, see Figure 1.10.

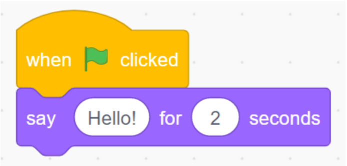


Figure 1.10

(6) Tabs of Costumes and Backdrop

In DobotBlock, we can process a sprite or backdrop picture. For example, after we select a sprite, we can change the color and size and so on by clicking the costume tabs. As shown in Figure 1.11, we can delete some costumes of the sprite "John" in the costume tabs, and modify the name of the costume.

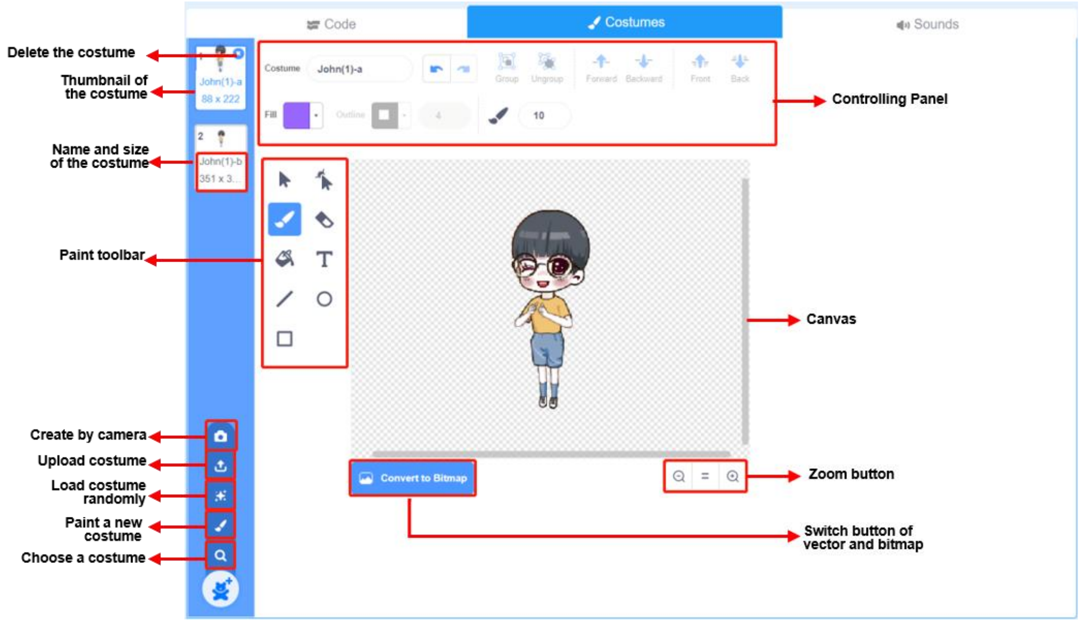


Figure 1.11

Let's read it

Sounds

Besides Code and Costumes, the sprite can play sound. This function makes the sprite lifelike. In Sounds, we can manage the sound played by the sprite. For example, we can edit the name and effect and so on of sound, see Figure 1.12.

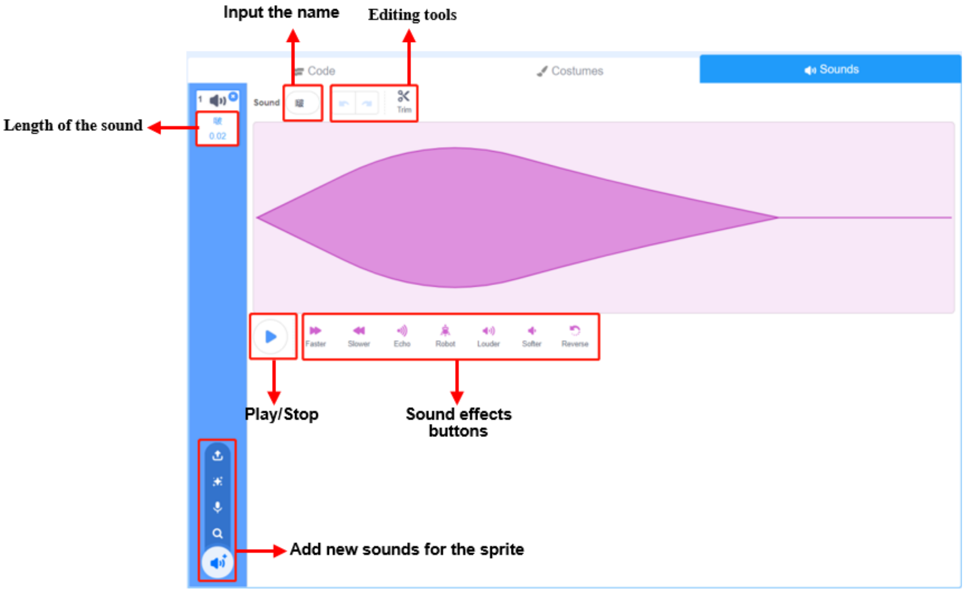


Figure 1.12

Research Laboratory

1. Task Release

(1) Task 1

In DobotBlock, we can upload a written program (Sing a song), and run it.

(2) Task 2

In DobotBlock, we can independently write a program, and try running it.

Processing Workshop

Upload a Program

Task 1

Step 1: Enable the programming software, click File in the menu, and Load from your computer, see Figure 1.13.

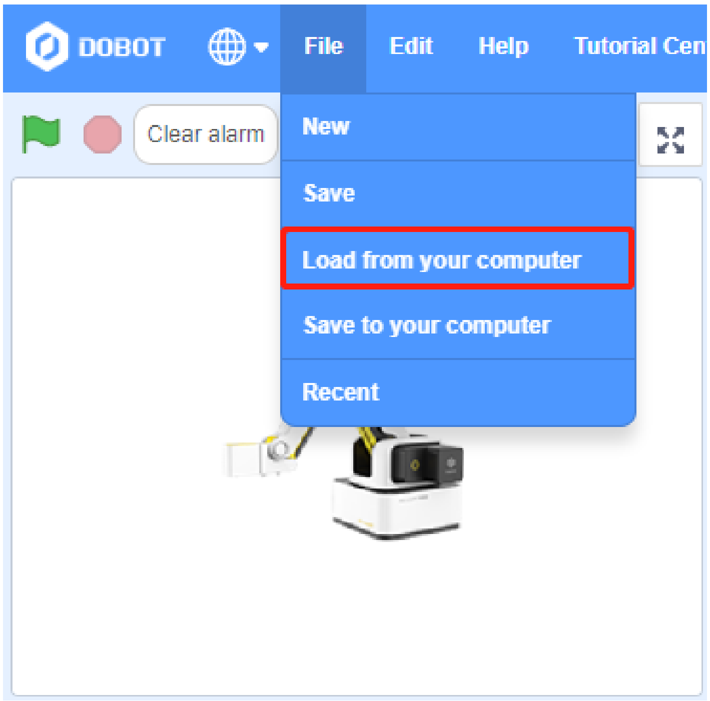


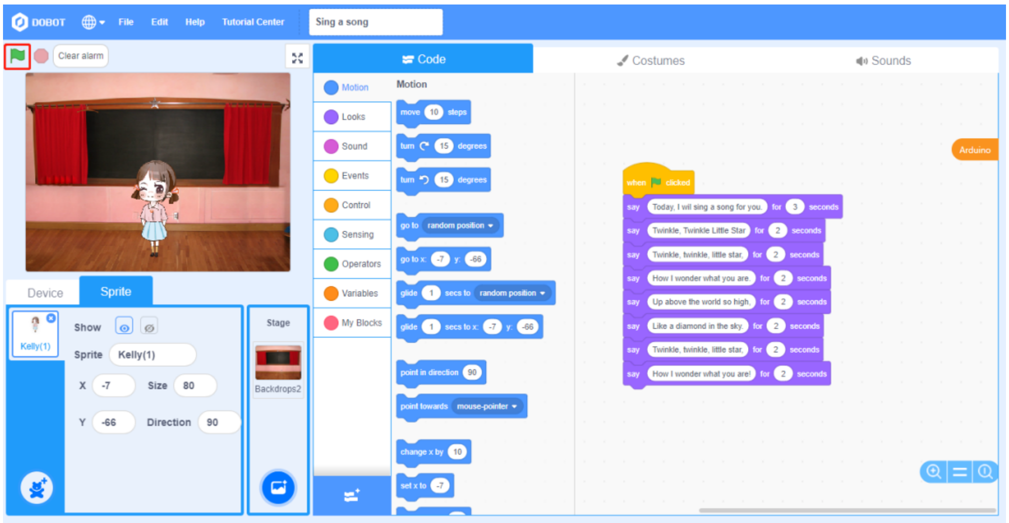
Figure 1.13

Step 2: Select the file "Sing a song.sb3", and finally click Open to upload the program.

Run a Program

Task 1

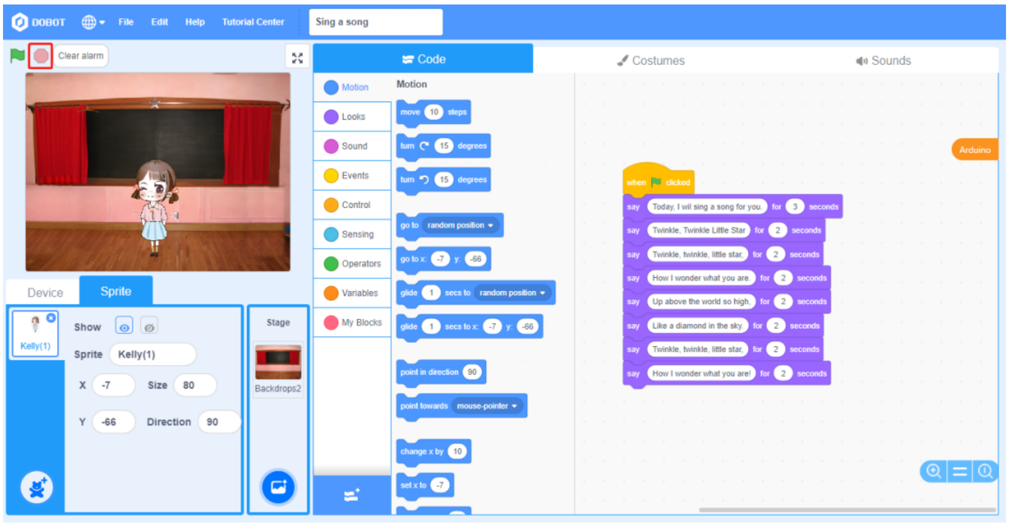
Step 1: Click the Sprite List, and click Green Flag on the stage to run the program, see Figure 1.14.



```
when ⚑ clicked
say Today, I wil sing a song for you. for 3 seconds
say Twinkle, Twinkle Little Star for 2 seconds
say Twinkle, twinkle, little star, for 2 seconds
say How I wonder what you are. for 2 seconds
say Up above the world so high, for 2 seconds
say Like a diamond in the sky. for 2 seconds
say Twinkle, twinkle, little star, for 2 seconds
say How I wonder what you are! for 2 seconds
```

Figure 1.14

Step 2: Click Stop on the stage to stop the running program, see Figure 1.15.



```
when ⚑ clicked
say Today, I wil sing a song for you. for 3 seconds
say Twinkle, Twinkle Little Star for 2 seconds
say Twinkle, twinkle, little star, for 2 seconds
say How I wonder what you are. for 2 seconds
say Up above the world so high, for 2 seconds
say Like a diamond in the sky. for 2 seconds
say Twinkle, twinkle, little star, for 2 seconds
say How I wonder what you are! for 2 seconds
```

Figure 1.15

Knowledge Base

Boys and girls, since we have known DobotBlock software and tried writing and running a program, could you tell me what is programming?

Let's read it

What is programming?

Programming is also known as program design. It is a program process to solve a specific problem and also a major step for software development. Program design generally uses a program design language as a tool, and gives a program of this language. Program design includes analysis, design, coding, testing, and debugging.

A program, a series of computer instructions arranged in order, is equivalent to the orderly task document of a computer. A computer instruction means an instruction or command that has a computer to execute a task.

Innovation Park

Modify the program "Sing a song": Ask the sprite to sing the song, then run the program, and watch the change in the sprite on the stage.

Self-Assessment Room

Content	Result
I've known the DobotBlock software and its interface	☆☆☆☆☆
I've preliminarily experienced what is programming	☆☆☆☆☆
I've written and run a program	☆☆☆☆☆

Chapter 2 Motion Control

- Smart Porter

John has recently opened a supermarket. He uses a smart robot as a helper. In this supermarket, the robot handles articles by motion. But how does the robot move and handle cargoes?

In this chapter, we will learn the Motion module through DobotBlock. The module can control the motion mode, position, direction and so on of a sprite or device. To control the motion of a sprite and a robot, we must have a command of some mathematical knowledge and logic. Then, how do we control the motion of a sprite and a robot. Boys and girls, let's explore this question!

Learning Target

☼ Students will learn how to control the motion of a sprite and a robot.

☼ Students will experience how to pleasurably control the motion of a sprite and a robot.

☼ Students will learn how to write a program to control the motion of a sprite and a robot.

2.1. Sprite Motion

- Robot Drawing Square

Kelly, my robot cannot now perform specific tasks. If I want it to move around along a route and direction, could you find a way to do this?

I think this is not a difficult problem. Mr. Lee must have a way. Let's go to ask him for help.

John and Kelly found Mr. Lee, and expressed their idea and requirements. Mr. Lee enabled DobotBlock in his computer, and asked John and Kelly to find Motion in the block area, see Figure 2.1. John and Kelly found many blocks in Motion of the sprite.

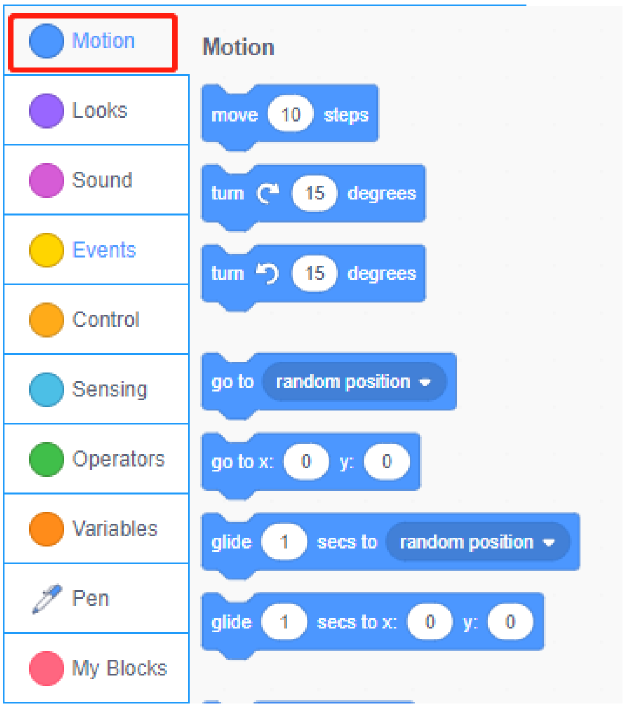


Figure 2.1

Let's think about it

✧ Which motions can these blocks in the Motion module help the sprite (e.g., a robot) complete?

John and Kelly find that these blocks can not only move the position of a robot, but also enable it to rotate. This is so fun!

2.1.1. Sprite Motion

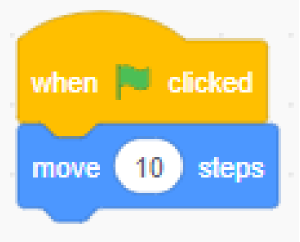
(1) Robot moving 10 steps

Let's think about it

✧ If we move a robot 10 steps, what blocks will we need? How do we combine them?

Let's do it

1. Click the ● Events module to drag the when ⚑ clicked block to the coding area.
2. Click the ● Motion module to drag the move 10 steps block to the coding area.
3. Combine these blocks, and run the program, see Figure 2.2.



```
when ⚑ clicked
move 10 steps
```

Figure 2.2

Boys and girls, we have finished running the program. Now, do you have any question?

After running the program, John and Kelly find a big problem, and ask Mr. Lee, "Mr. Lee, this robot turns right only. It cannot reach the designated position, nor turn around. What shall we do?"

(2) Robot moving forward or backward

Mr. Lee asks John and Kelly to continuously watch the blocks in the motion module, and to think about the function of the "go to x: (-10) y: (0)" block, see Figure 2.3. What does this block do?

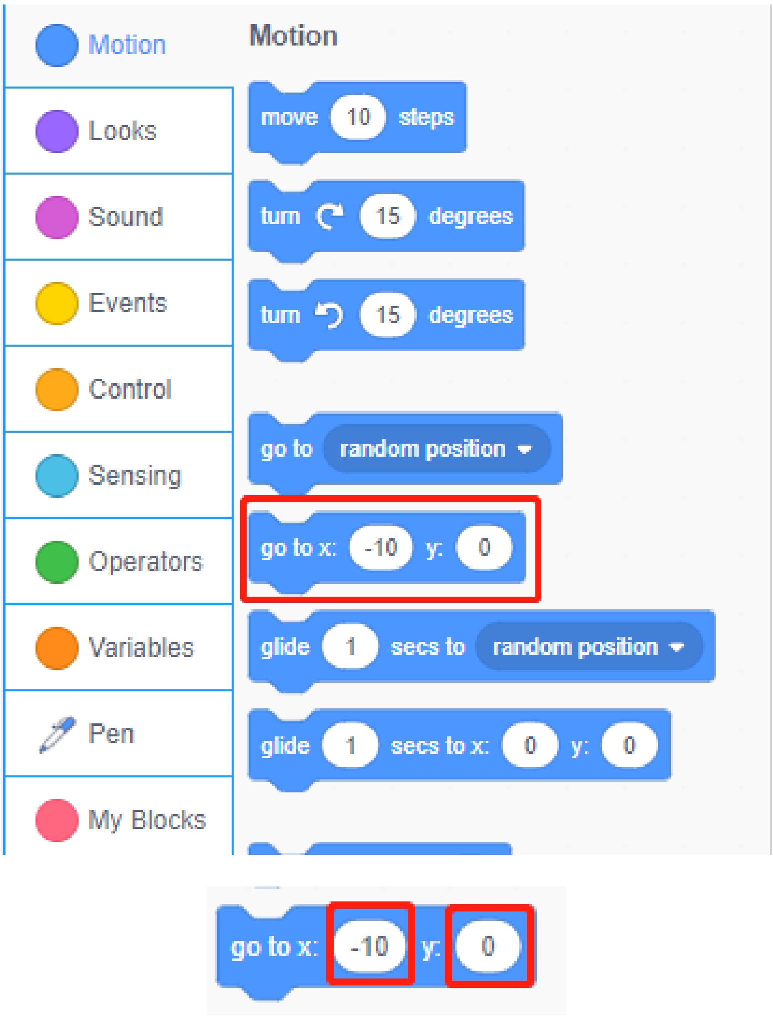
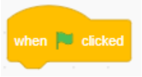
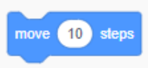
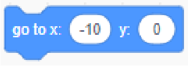


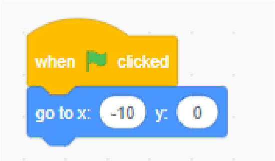
Figure 2.3

John and Kelly find two values users can enter in the "go to x: (-10) y: (0)" block. They understand the y value is 0. But they do not know why the x value is -10. This is a negative number in math. Boys and girls, do you know negative numbers? Let's guess what motions this block enables a robot to complete in this program.

Let's do it

1. Click the Events module to drag the  when clicked block to the Coding Area.

2. Click to replace  move 10 steps in the program moving a robot 10 steps with  go to x: -10 y: 0 and run the program, and watch how the robot moves, see Figure 2.4.

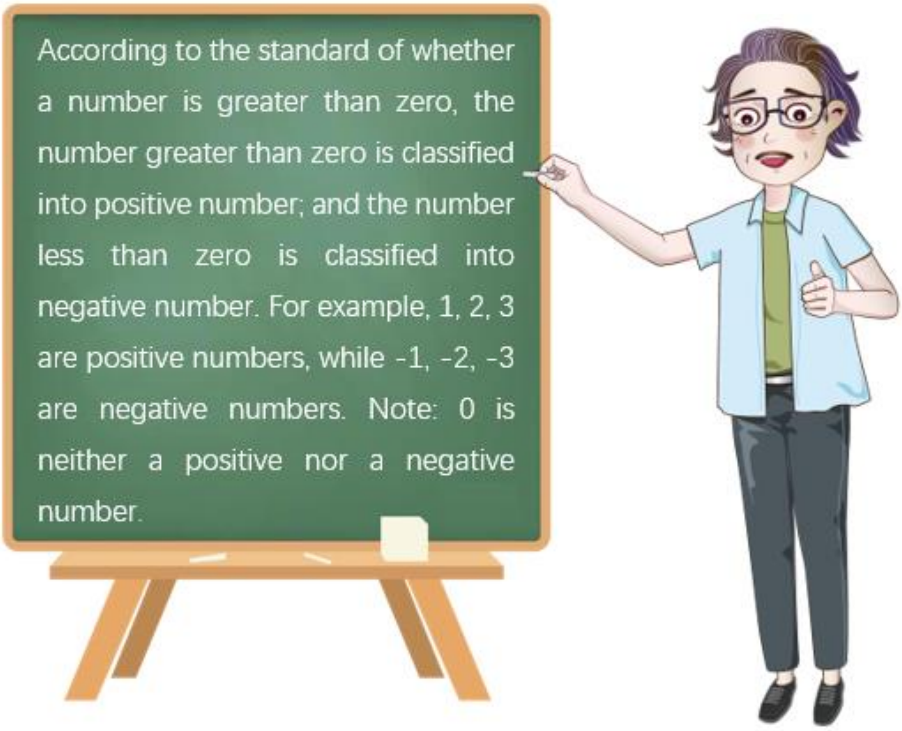


when clicked
go to x: -10 y: 0

Figure 2.4

Boys and girls, do you see other negative numbers in life? Let's get to know them.

(3) Positive numbers and negative numbers



According to the standard of whether a number is greater than zero, the number greater than zero is classified into positive number; and the number less than zero is classified into negative number. For example, 1, 2, 3 are positive numbers, while -1, -2, -3 are negative numbers. Note: 0 is neither a positive nor a negative number.

As shown in Figure 2.5, the part on the left of zero along the number axis refers to negative numbers, while the part on the right of zero refers to positive numbers. A positive number is greater than zero, a negative number is smaller than zero, and a positive number is greater than a negative number.

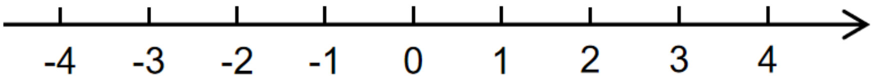
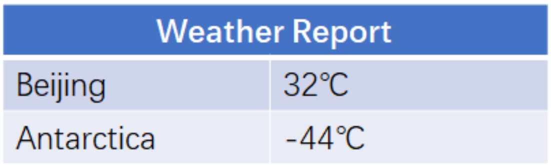


Figure 2.5

Positive numbers and negative ones are applied to not only math but also life. Boys and girls, as it is very cold, particularly in the case of icing, we often hear "centigrade degrees below zero" from weather forecast. For example, in Figure 2.6, it is "44 centigrade degrees below zero" in Antarctica, which is also written as "-44℃".

Here, the negative number indicates a temperature lower than 0℃ or a temperature below zero. In this case, there is a "-" (negative sign) before a number. A temperature higher than 0℃ is called a temperature above zero. In this case, there is a "+" (positive sign) before a number, but "+" is often omitted. 0℃ indicates a temperature at which fresh water starts to freeze. For example, as Figure 2.6 shows, it is a temperature above zero in Beijing, while it is a temperature below zero in Antarctica. Boys and girls, let's read and know these temperatures.



Weather Report	
Beijing	32℃
Antarctica	-44℃

Figure 2.6

Is a negative number presented in DobotBlock programming? What role does it play? To find out these two questions, we need to learn another mathematical concept, namely, rectangular coordinate system.

(4) Plane Cartesian coordinate system

There is a plane Cartesian coordinate system on the stage of DobotBlock. Then, what is plane a Cartesian coordinate system? This is a plane Cartesian coordinate system, see Figure 2.7.

Let's think about it

✧ What do you learn about from the plane Cartesian coordinate system? How many axes does it contain? What do x and y indicate respectively? What indicates the origin?

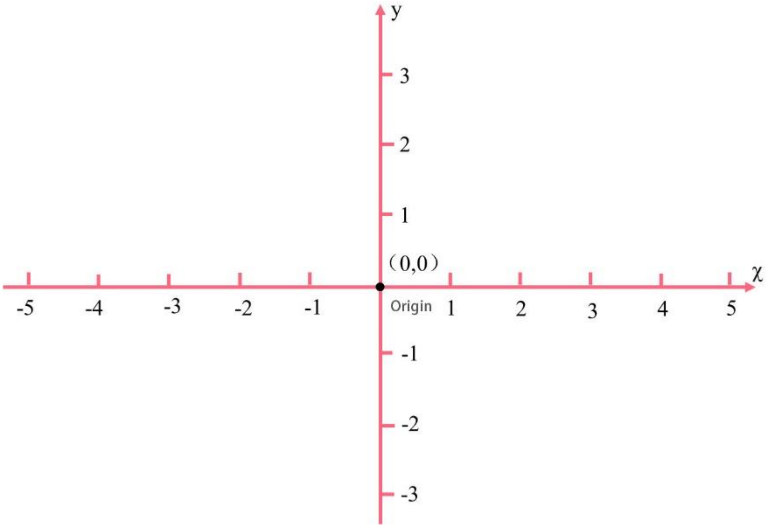


Figure 2.7

A plane Cartesian coordinate system means a coordinate system consisting of two mutually perpendicular axes with a common origin (the lateral axis is x-axis, and the longitudinal axis is y-axis) in the same plane. The intersection point of the two axes is called origin, whose coordinate is mathematically stipulated to be (0, 0).

In a plane Cartesian coordinate system, we can set the coordinate of any point in the plane. The coordinate of any point in the plane contains two value: one indicates the lateral coordinate or x-coordinate; the other one indicates the longitudinal coordinate or y-coordinate.

The coordinate of any point in the plane is written in a pair of brackets, where the x-coordinate is placed before the y-coordinate, and both are separated by a comma. As shown in Figure 2.8, the coordinate of point A is (3, 2), and that of point B is (-4, -2).

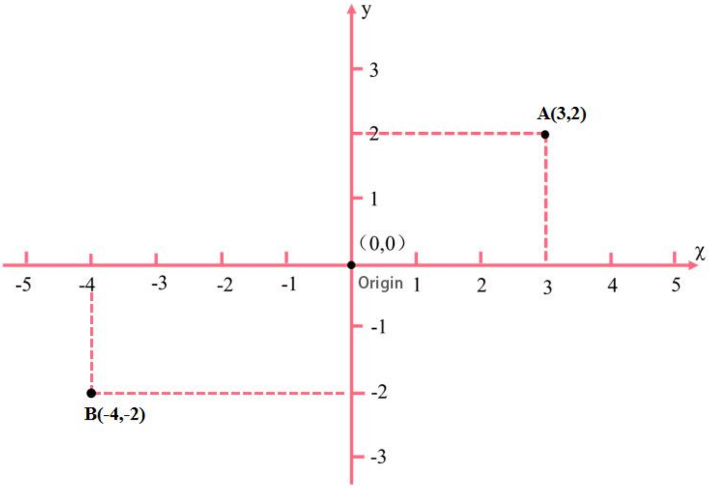


Figure 2.8

Boys and girls, can do you find the coordinate of any point in the coordinate system? Let's interact and play two rounds of the "report the coordinate by point" game!

Now, you have known the plane Cartesian coordinate system. But do you know who invented it?

Let's read it

The plane Cartesian coordinate system (or 2D coordinate system) in math is a great invention by French mathematician Rene Descartes.

Figure 2.9 Rene Descartes

(5) Cartesian coordinate system on the stage

There is a plane Cartesian coordinate system on the stage in the DobotBlock software. Then, how is the system presented?

Boys and girls, do you know what plane figure the stage of DobotBlock software has?

Yes, it is a rectangular area, with the center being the origin of the coordinate system. Can you read the stage scope? See Figure 2.10.

Let's describe it

What do you learn about from Figure 2.10? In this plane Cartesian coordinate system, how do we indicate the coordinate of any point?

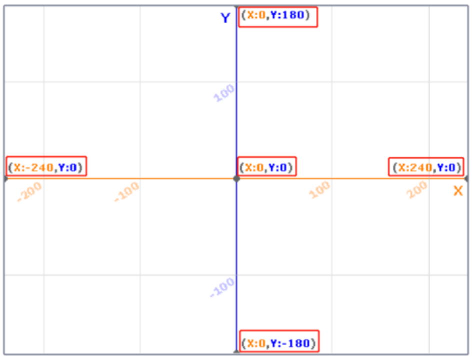


Figure 2.10

The stage of DobotBlock is a standard plane Cartesian coordinate system. Find and click the "Xy-grid" backdrop from the backdrop library, and now, we can see the plane Cartesian coordinate system on the stage. On the stage, the horizontal range is from -240 to 240, and the vertical range from -180 to 180. The stage center is the origin of this coordinate system, with a coordinate (0, 0).

The motion of the sprite is actually the motion of its coordinate point, during which the point moves to another position from one position. The position of the sprite depends on the x-coordinate and the y-coordinate of its center. As shown in Figure 2.11, the central position of the robot has a coordinate (0, 0), and the sprite robot stands at the center of the stage. If we move the central position of the robot to the coordinate (100, 100), the sprite robot will move to the top right corner of the stage, see Figure 2.12.

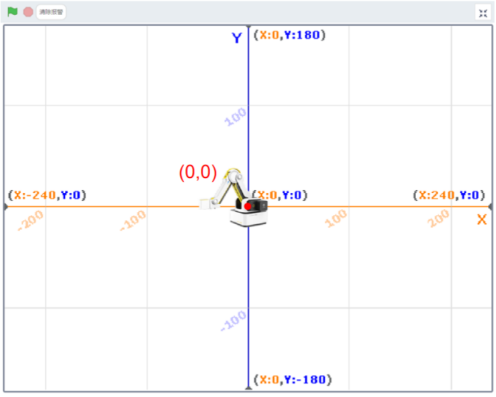


Figure 2.11

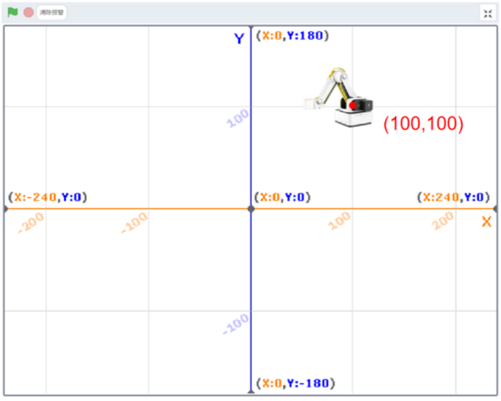


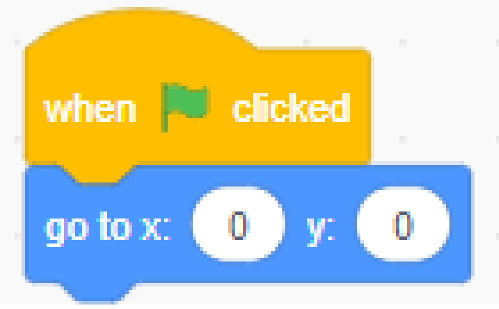
Figure 2.12

Let's think about it

- ✧ What is the coordinate indicated by the "go to x: (-10) y: (0)" block? If we need to return the sprite to the origin, how should we modify the value in this block?

Let's try it

To move a robot to the stage center through a program, we just need to change the values of the x-coordinate and y-coordinate in the "go to x: (-10) y; (0)" block to 0, and click the green flag button, see Figure 2.13.



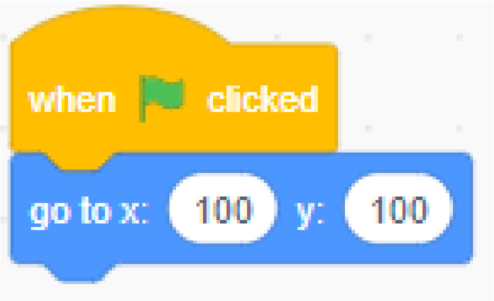
```
when 🏁 clicked
go to x: 0 y: 0
```

Figure 2.13

Boys and girls, do you find that the "go to x: (-10) y; (0)" block is to accurately move the sprite to a specific position on the stage? Yes, it is a practical and convenient block.

Let's try it

How do we move a robot to the top right corner of the stage through a program? Similarly, we just need to change the values of the x-coordinate and y-coordinate in the "go to x: (-10) y; (0)" block to 100, and click the green flag button, see Figure 2.14.



```
when 🏁 clicked
go to x: 100 y: 100
```

Figure 2.14

Let's do it

Use the "go to x: (-10) y; (0)" block to modify the x-coordinate and y-coordinate values:

Control a robot to move to the top right corner of the stage.

Control a robot to move to the position (145, -18) of the stage.

Control a robot to move to any set position of the stage.

2.1.2. Sprite Rotation

John and Kelly can move a robot to the designated position by programming. Yet the robot still faces the same direction only. They find some direction rotation blocks in Motion. Then, what do these blocks do? How do we use them?

In DobotBlock, each sprite is set a direction before it starts to move. By default, the sprite turns right.

Then, how do we choose the rotation direction of a sprite? First, click Motion, and find the "point in direction (90)" block, see Figure 2.15.

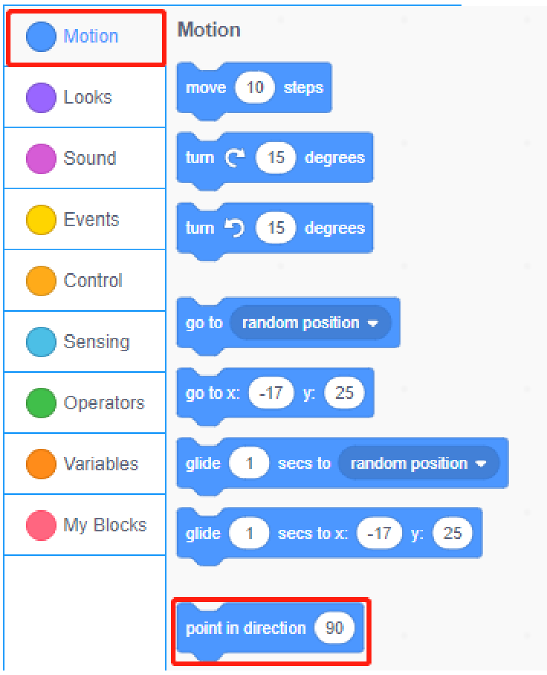
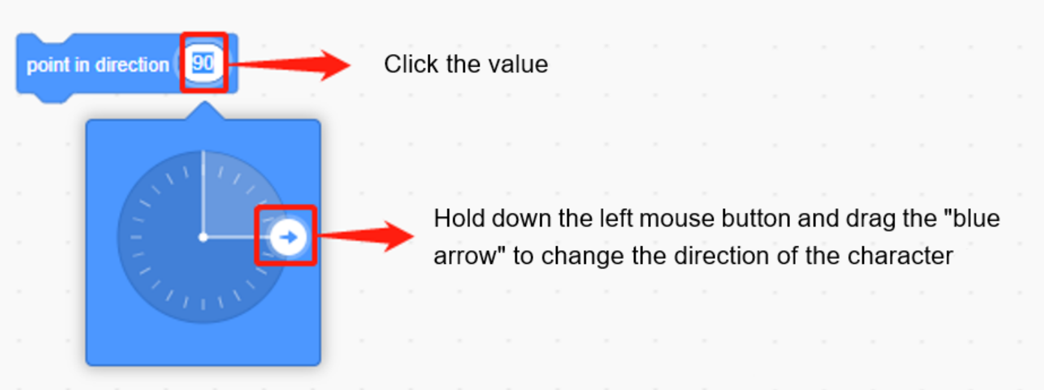


Figure 2.15

Drag the "point in direction (90)" block to the Coding Area, and click the value area of this block. Then, a disk like a dial pops up, see Figure 2.16.



Click the value

Hold down the left mouse button and drag the "blue arrow" to change the direction of the character

Figure 2.16

The arrow is upward = point in direction (0)

The arrow turns right = point in direction (90)

The arrow is downward = point in direction (180)

The arrow turns left = point in direction (-90)

Of course, besides the above four cases, we can choose many other directions of sprite rotation, see Figure 2.17. We can choose any direction from a 360-degree circle. See Figure 2.18 for the detailed angle corresponding to each direction.

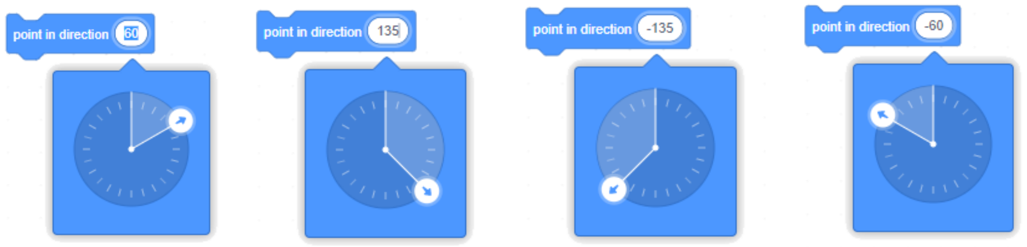


Figure 2.17

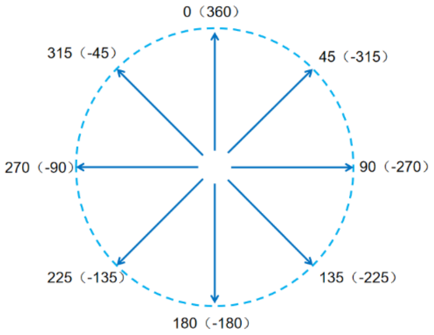


Figure 2.18

Boys and girls, have you learned sprite rotation? Let's perform a challenging task.

🔍 Research Laboratory

1. Task Release

Program the Robot to Draw a Square: Program a robot to draw a square on the stage, for example, draw a square with a 7 cm side.

2. Task Analysis

Boys and girls, if you want a robot to draw a square, what should you consider?

You need to set the stage backdrop to white;

You need to create a sprite robot;

From the Control module, you need to find the module that controls the waiting time every time the robot finishes drawing a stroke;

You need to set the position and direction of the robot;

You need to add Pen module in Add Extension to draw lines;

…

Carefully watch the following figure. We find this figure has four equal sides and four right angles. It is called a square, see Figure 2.19.

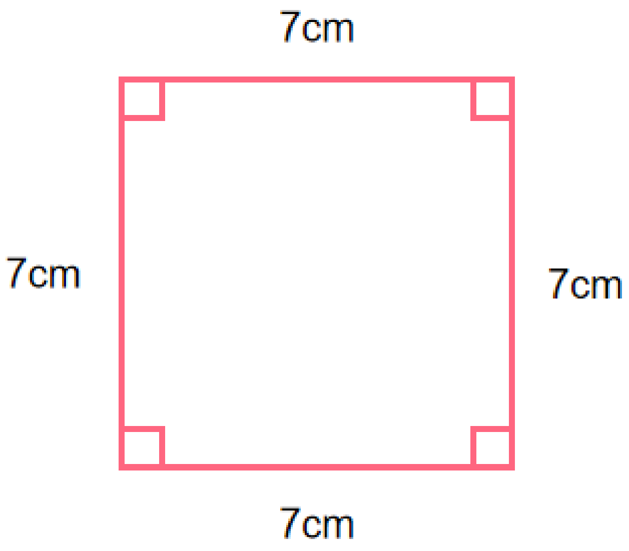
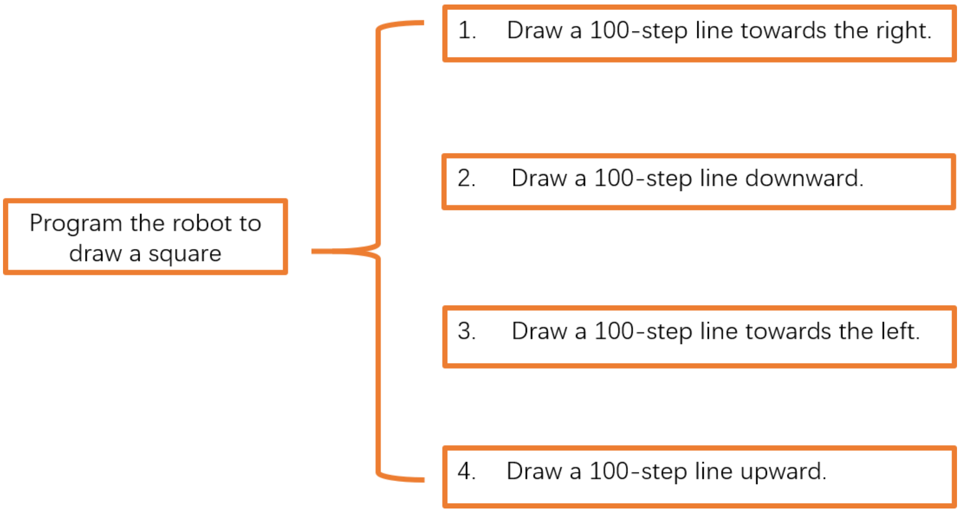


Figure 2.19

Obviously, a robot takes four steps to draw a square, see Figure 2.20:



Program the robot to draw a square

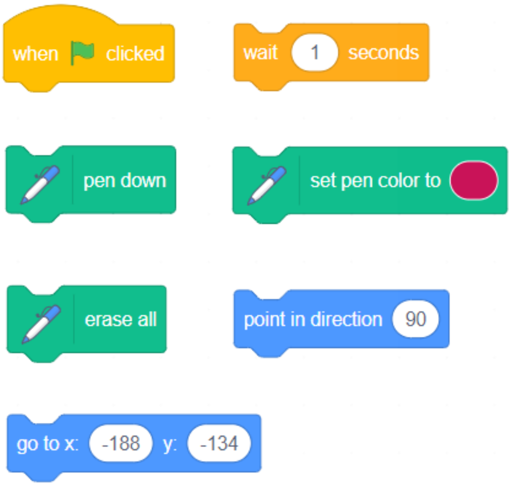
1. Draw a 100-step line towards the right.
2. Draw a 100-step line downward.
3. Draw a 100-step line towards the left.
4. Draw a 100-step line upward.

Figure 2.20

You usually draw a square with a ruler and a pen. How does a robot do so in the programming software? From the task analysis we can find that this job uses the Motion module and the Pen module.

2.1.3. Script Plan

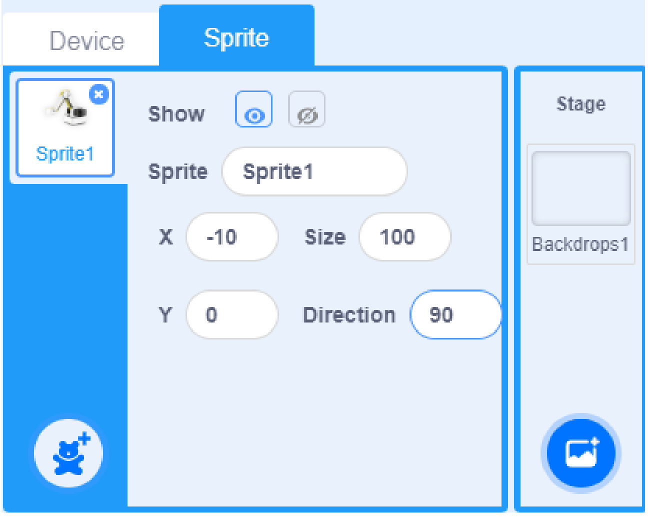
Before starting this operation, we can write a script plan to guide the programming.

Sprite	Task Description	Block
	A robot draws four lines to form a square, with each line being 100 long.	 when clicked wait 1 seconds pen down set pen color to erase all point in direction 90 go to x: -188 y: -134

Processing Workshop

Add a Sprite and Module

Step 1: Select the default sprite "Sprite 1", see Figure 2.21.



Show

Sprite: Sprite1

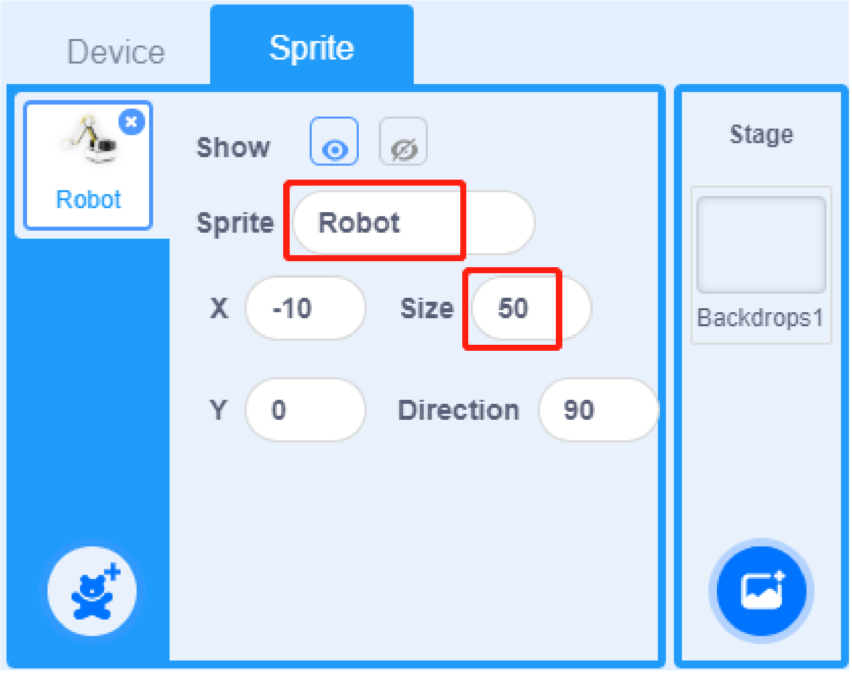
X: -10 Size: 100

Y: 0 Direction: 90

Figure 2.21

Add a Sprite and Module

After we add a sprite, we often need to adapt its size to the stage, or change its name to facilitate use. In this case, we also need to modify the name and size of the sprite, see Figure 2.22.



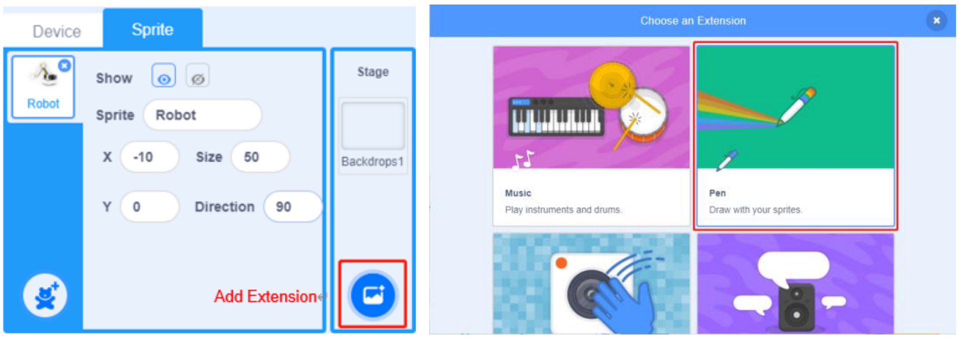
Sprite Robot

X -10 Size 50

Y 0 Direction 90

Figure 2.22

Step 2: Add a Pen module. Click the Add Extension button to add the Pen module, see Figure 2.23.



Sprite Robot

X -10 Size 50

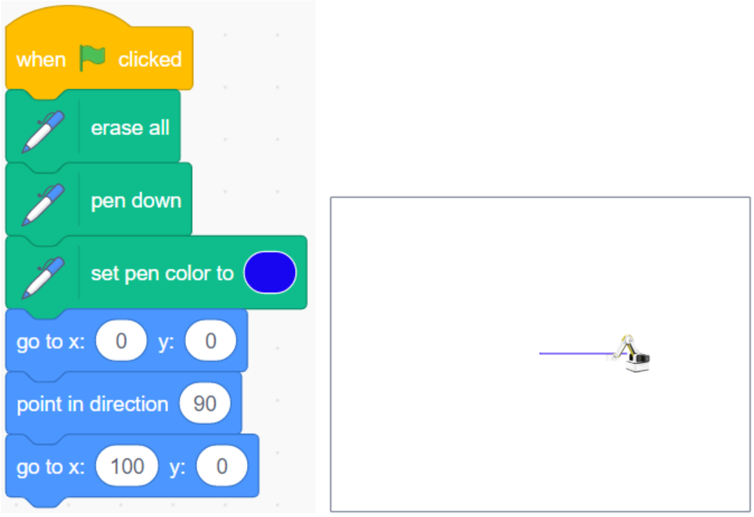
Y 0 Direction 90

Figure 2.23

Write a Sprite Script

Select the sprite robot in the sprite column. As per the script plan, drag a block, and program the sprite robot in the coding area to draw a square.

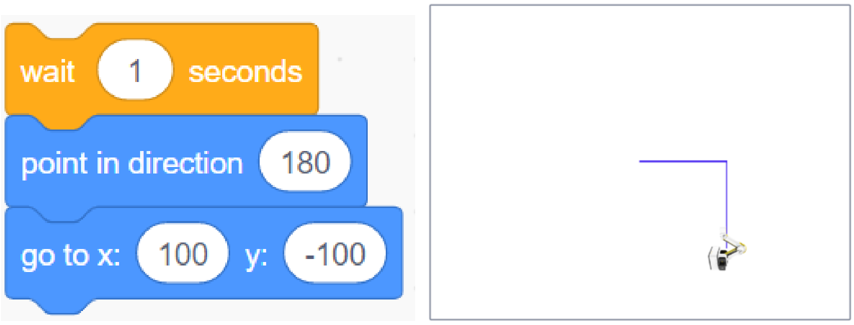
Step 1: Select the sprite robot, write a program shown in Figure 2.24, and draw the first line towards the right.



```
when 🏳 clicked
erase all
pen down
set pen color to ●
go to x: 0 y: 0
point in direction 90
go to x: 100 y: 0
```

Figure 2.24

Step 2: Select the sprite robot, write a program shown in Figure 2.25, and draw the second line downward.

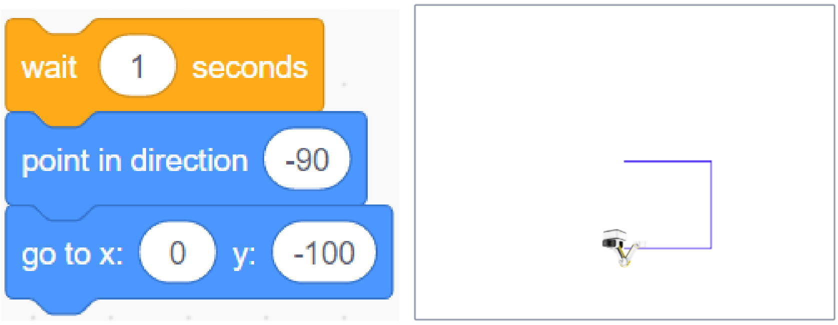


```
wait 1 seconds
point in direction 180
go to x: 100 y: -100
```

Figure 2.25

Write a Sprite Script

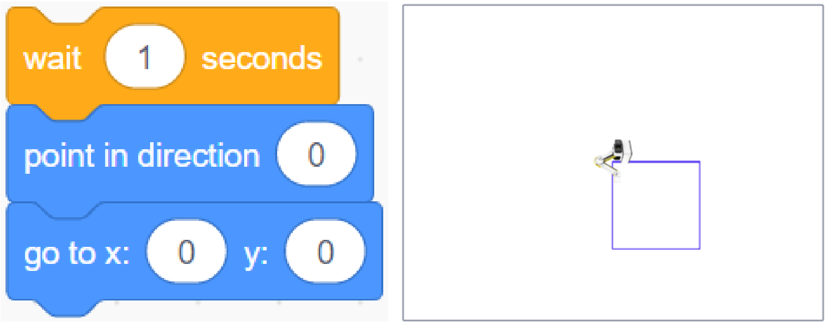
Step 3: Select the sprite robot, write a program shown in Figure 2.26, and draw the third line towards the left.



```
wait 1 seconds
point in direction -90
go to x: 0 y: -100
```

Figure 2.26

Step 4: Select the sprite robot, write a program shown in Figure 2.27, and draw the fourth line upward.

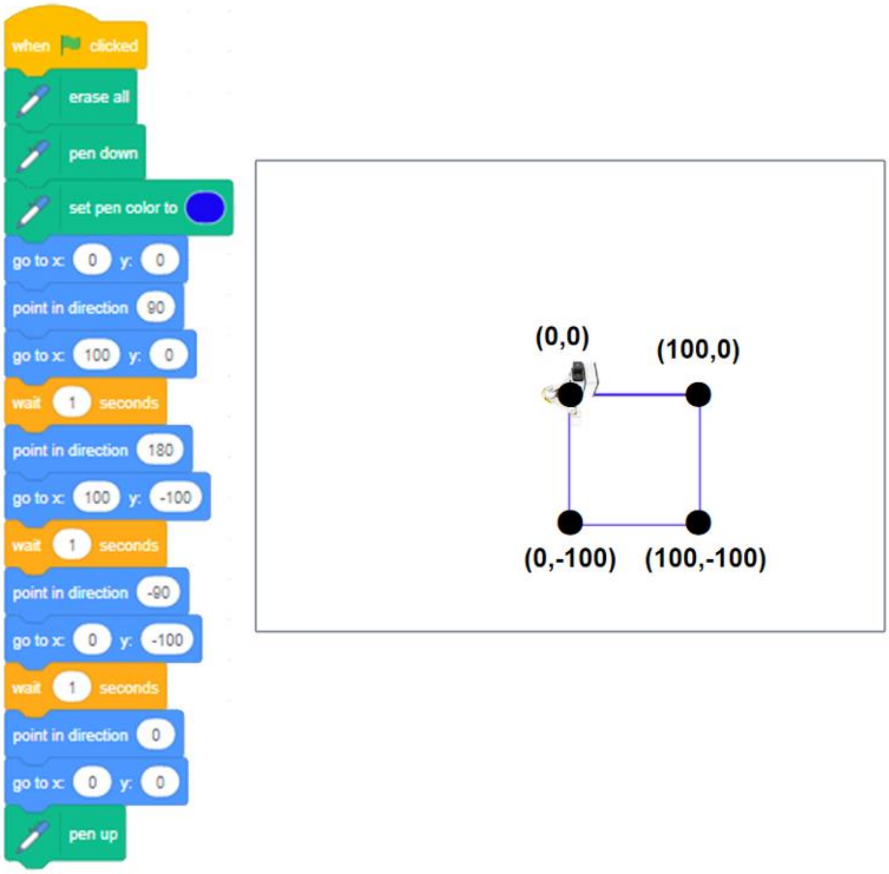


```
wait 1 seconds
point in direction 0
go to x: 0 y: 0
```

Figure 2.27

Write a Sprite Script

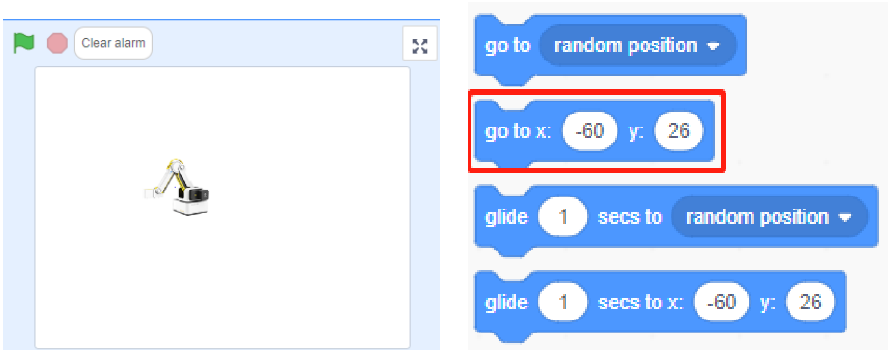
Step 5: Combine the above programs to allow the robot to draw a square. The relations between the coordinates of the four angles of the square and the coordinates set in the program are shown in Figure 2.28.



```
when 🏁 clicked
erase all
pen down
set pen color to ●
go to x: 0 y: 0
point in direction 90
go to x: 100 y: 0
wait 1 seconds
point in direction 180
go to x: 100 y: -100
wait 1 seconds
point in direction -90
go to x: 0 y: -100
wait 1 seconds
point in direction 0
go to x: 0 y: 0
pen up
```

Figure 2.28

Note: The initial position of the robot may be any point on the stage. In Figure 2.29, when we move the position of the robot, the coordinate point on the "go to x: (-10) y: (0)" block will change accordingly.



```
go to random position ▾
go to x: -60 y: 26
glide 1 secs to random position ▾
glide 1 secs to x: -60 y: 26
```

Figure 2.29

Innovation Park

In this course, we allow a robot to draw a square by calculating the coordinates of the four angles of the square. Boys and girls, now let's summarize how to calculate such coordinates.

If we want each side of a square to extend by 1.5 time to reach 150, what shall we do? Modify the program according to Figure 2.30.

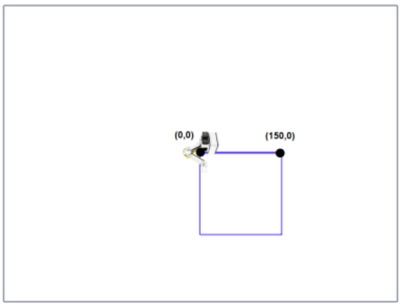


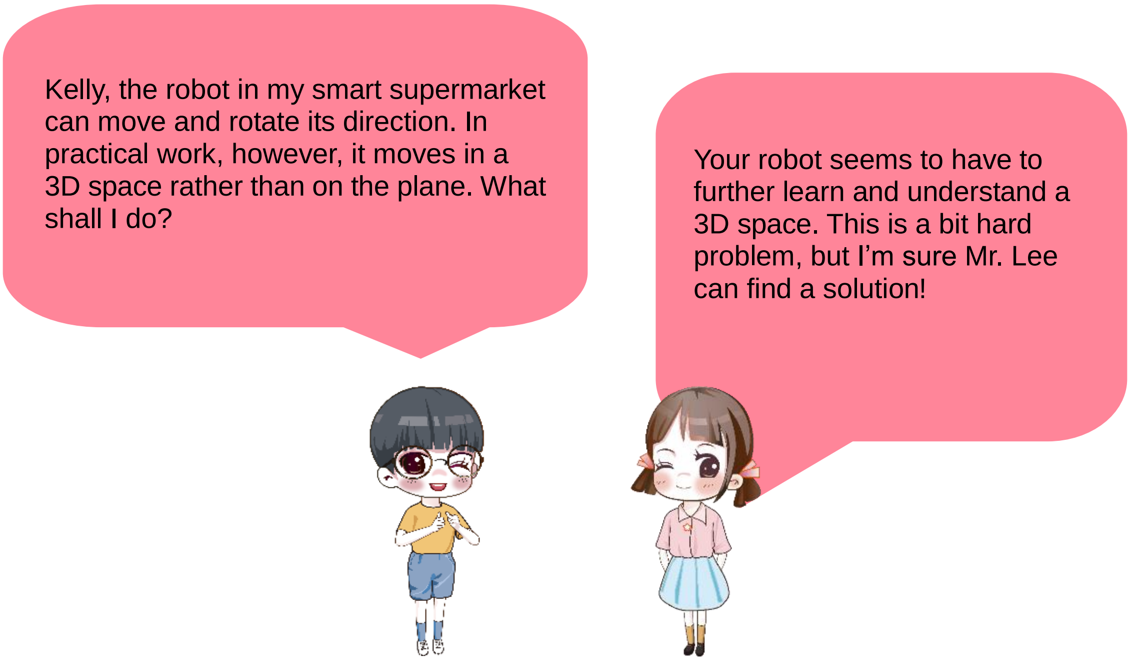
Figure 2.30

Self-Assessment Room

Content	Result
I've learned how to operate the "go to x: (-10) y: (0)" block	☆☆☆☆☆
I've known what is a Plane Cartesian coordinate system	☆☆☆☆☆
I've known how to control the motion direction of a sprite	☆☆☆☆☆
I've made a robot draw a square	☆☆☆☆☆

2.2. Robot Motion

- Cargo Handling


Kelly, the robot in my smart supermarket can move and rotate its direction. In practical work, however, it moves in a 3D space rather than on the plane. What shall I do?
Your robot seems to have to further learn and understand a 3D space. This is a bit hard problem, but I'm sure Mr. Lee can find a solution!

2.2.1. Space Cartesian coordinate system

John asks everyone to think about how many axes a plane Cartesian coordinate system have. How many values can set the coordinate of a point?

The coordinate system offers two axes, namely, x-axis (lateral axis) and y-axis (longitudinal axis). Two values (x, y) can set the coordinate of a point.

Let's think about it

- A space Cartesian coordinate system is a 3D space, as shown in Figure 2.31. How many axes does it have? How many values can set the coordinate of a point?

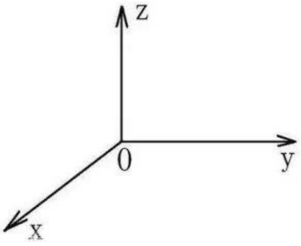


Figure 2.31

This coordinate system has three axes, namely, x-axis (lateral axis), y-axis (longitudinal axis), and z-axis (vertical axis), which are also called coordinate axes. The coordinate axes and the origin constitute a space Cartesian coordinate system 0-xyz. We can determine any point in space as 0, through which to draw three mutually perpendicular axes (Ox, Oy and Oz). These axes take 0 as the origin, and offer the same length unit.

In this coordinate system, three values can set the coordinate (x, y, z) of a point, with 0 point being the coordinate origin. For example, how should we denote the top position of Kelly through a space Cartesian coordinate system? As shown in Figure 2.32, point B' in her top position has the x-coordinate value as 1, the y-coordinate value as 1 and the z-coordinate value as 1. Thus, the coordinate of point B' is (1, 1, 1).

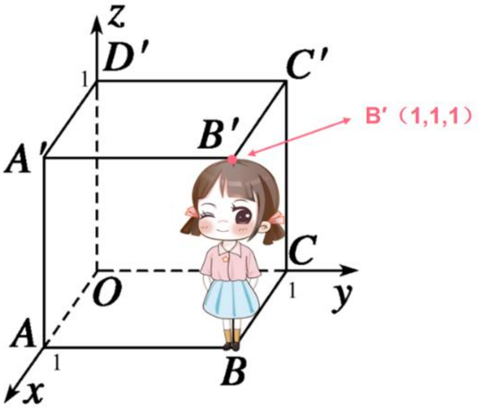


Figure 2.32

2.2.2. Robotic Coordinate System

Similarly, to determine the end position of a robot, we must know its space coordinate.

When we learn about the space coordinate of a point of a robot, we can accurately control the position of such a point. For example, to move right such a point 6 mm, forward 7 mm, and upward 5 mm, we just change the parameter of its coordinate. The space Cartesian coordinate system of a robot is shown in Figure 2.33.

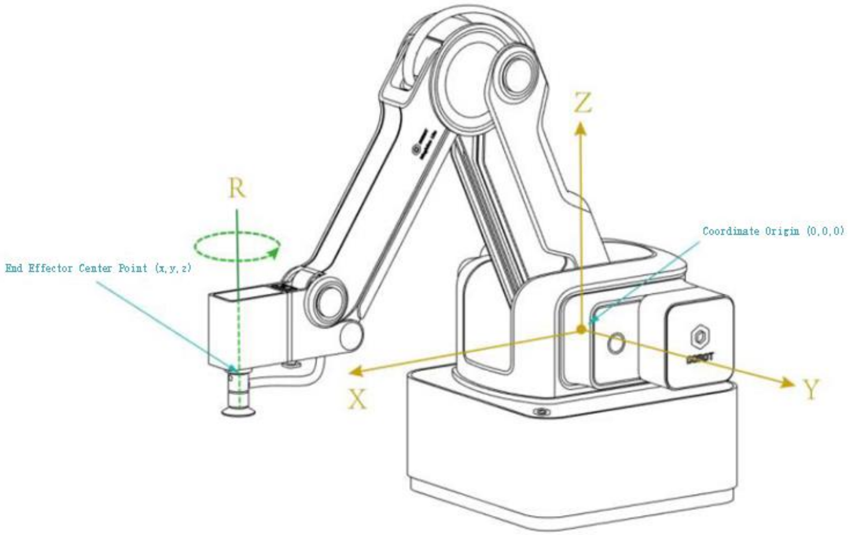


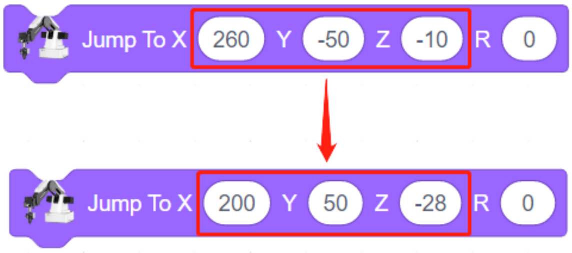
Figure 2.33

Tip: No R axis exists without an end kit carrying a steering engine on the robot. For example, the robot can rotate around R axis as it uses the suction cup and the soft gripper.

Let's think about it

✧ How do we use a program to move a robot to point B from point A?

For example, to control the end of a robot to jump to point B (200, 50, -28) from point A (260, -50, -10), see Figure 2.34. Boys and girls, do you understand this? You can try doing this.



Jump To X 260 Y -50 Z -10 R 0

Jump To X 200 Y 50 Z -28 R 0

Figure 2.34

Do you know how to move a robot to point B from point A along the door-shaped route?

Let's think about this question this way: First raise the robotic arm to a certain height from point A, then move it horizontally to the position above point B, and finally lower it to the position of point B.

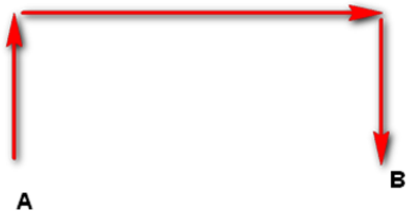


Figure 2.35

Research Laboratory

1. Task Release

Create a "Porter" Game: Program a robot to simulate a porter to carry a block.

Boys and girls, think back how a porter loads cargoes. Generally, what should a porter do before he carries cargoes to a truck from the storage zone? See Figure 2.36 for this process.

Figure 2.36

To improve efficiency and save manpower, how do we enable a robot to automatically carry cargoes?

In this task we should use a robot to simulate a porter to carry cargoes twice. In this course we will use a block to simulate a cargo, and carry two stacked blocks to the loading zone from the storage zone. We will also stack those blocks placed in the Loading Zone. For the arrangement of materials, see Figure 2.37.

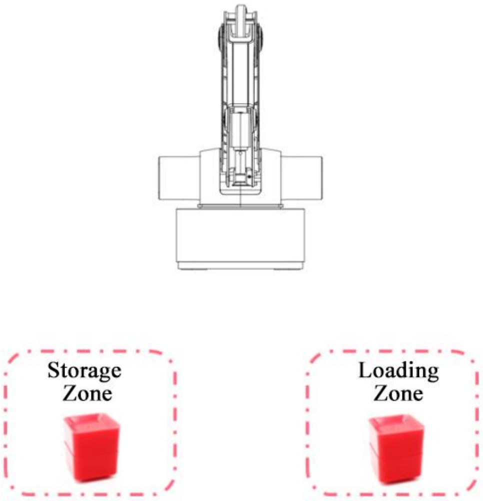


Figure 2.37

2. Task Analysis

To complete this task, you need to consider:

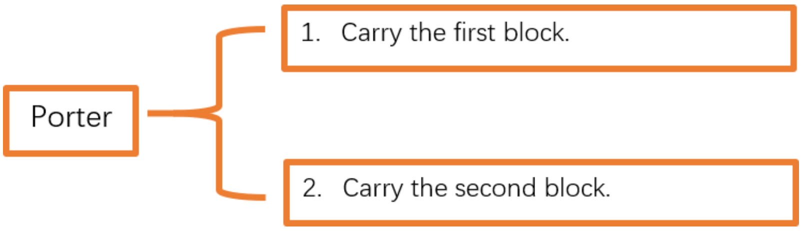
Arrange materials and devices, see Figure 2.37;

Add the robot device to Device;

Program a robot to carry blocks, and have a robot carry a block to the designated position with a suction cup;

…

This handling task needs two steps, see Figure 2.38:

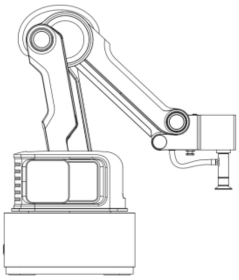

Porter

1. Carry the first block.

2. Carry the second block.

Figure 2.38

3. Script Plan

Device	Task Description	Block
	A robot carries a block to the loading zone from the storage zone.	when 🏁 clicked Suction Cup ON ▾ wait 0.5 seconds Jump To X 260 Y -50 Z -28 R 0

Processing Workshop

Add and Connect a Device

Click Add Device, choose Magician Lite from the device library, and delete the default device Magician, see Figure 2.39.

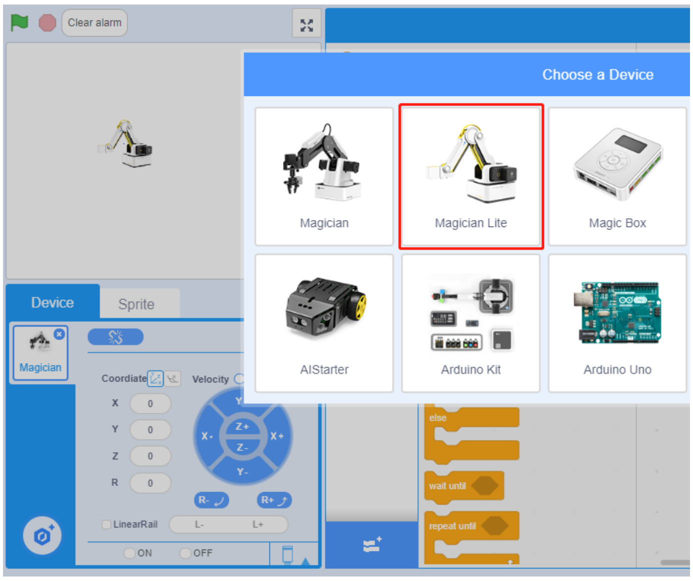
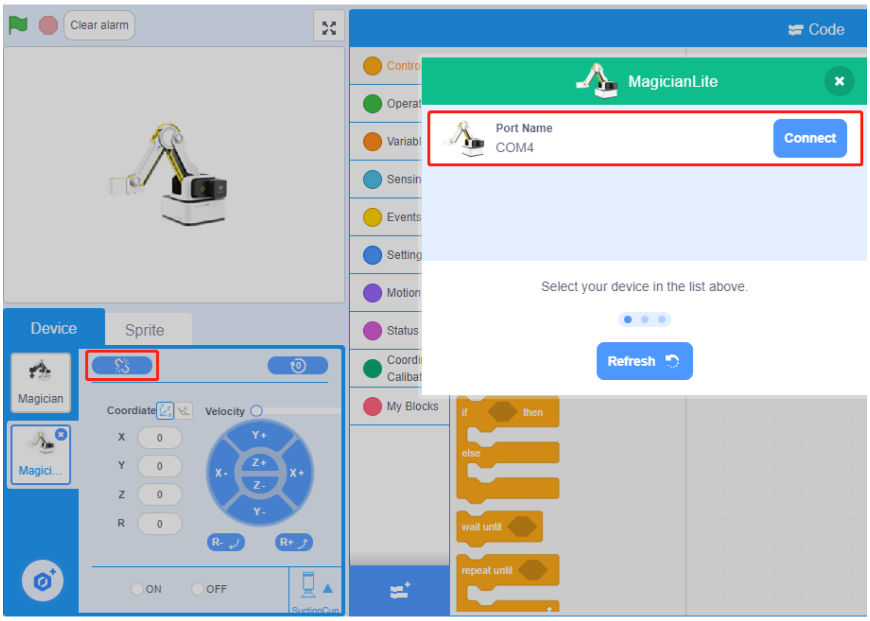


Figure 2.39

Add and Connect a Device

Steps for connecting a device: Click Connect Device, and a page appears. Then, select the corresponding port, and click Connect, see Figure 2.40.



MagicianLite

Port Name
COM4

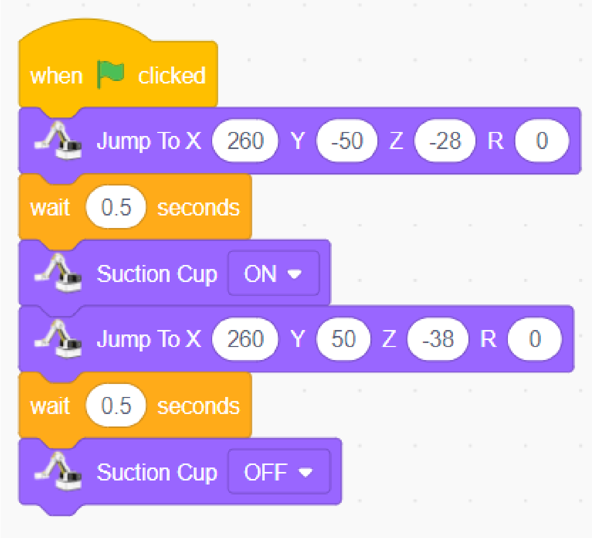
Select your device in the list above.

Figure 2.40

Write a Device Script

Select a robot. As per the script plan, drag a block, and program the Robot device in the coding area to carry a block.

Step 1: Select the Robot device, and write a program shown in Figure 2.41.



```
when 🏁 clicked
Jump To X 260 Y -50 Z -28 R 0
wait 0.5 seconds
Suction Cup ON ▾
Jump To X 260 Y 50 Z -38 R 0
wait 0.5 seconds
Suction Cup OFF ▾
```

Figure 2.41

Write a Device Script

Each block is fixed in height, and about 10 mm in concave height. As shown in Figure 2.42, after we determine the suction position of the first block, we can reduce the coordinate of z axis by 10 mm to determine the suction position of the second block.

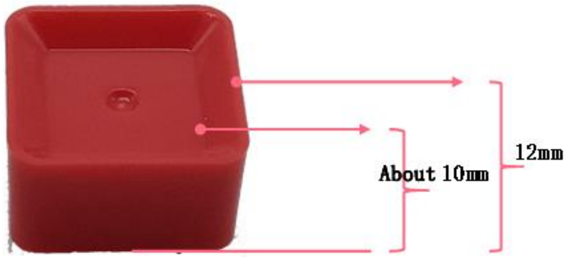
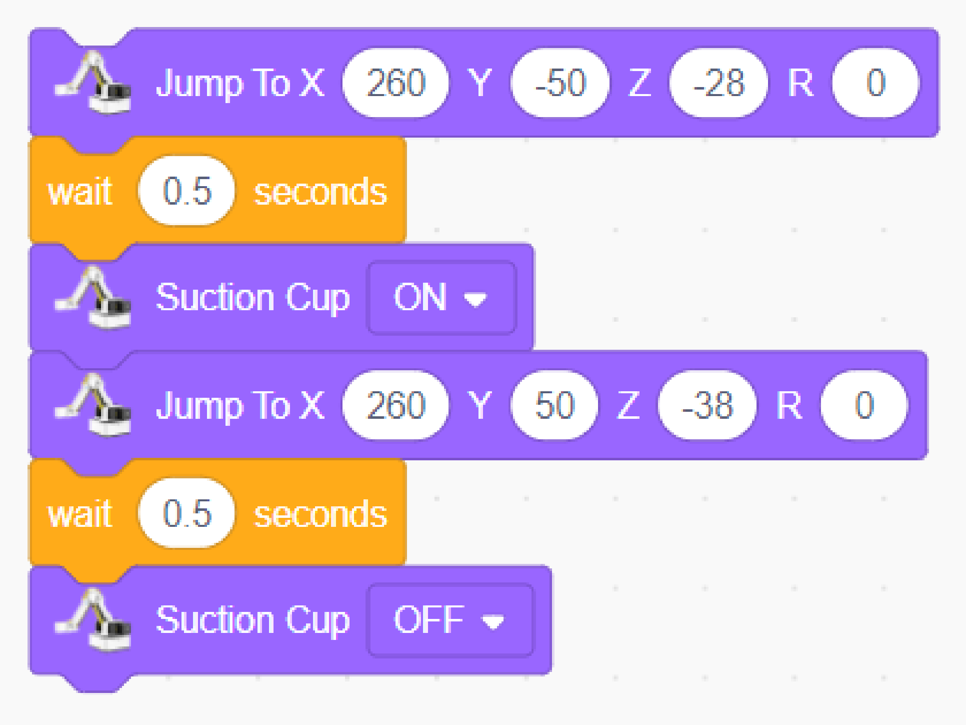


Figure 2.42

Step 2: Write a program for "carrying the second block". Select the Robot device, and program the blocks shown in Figure 2.43.

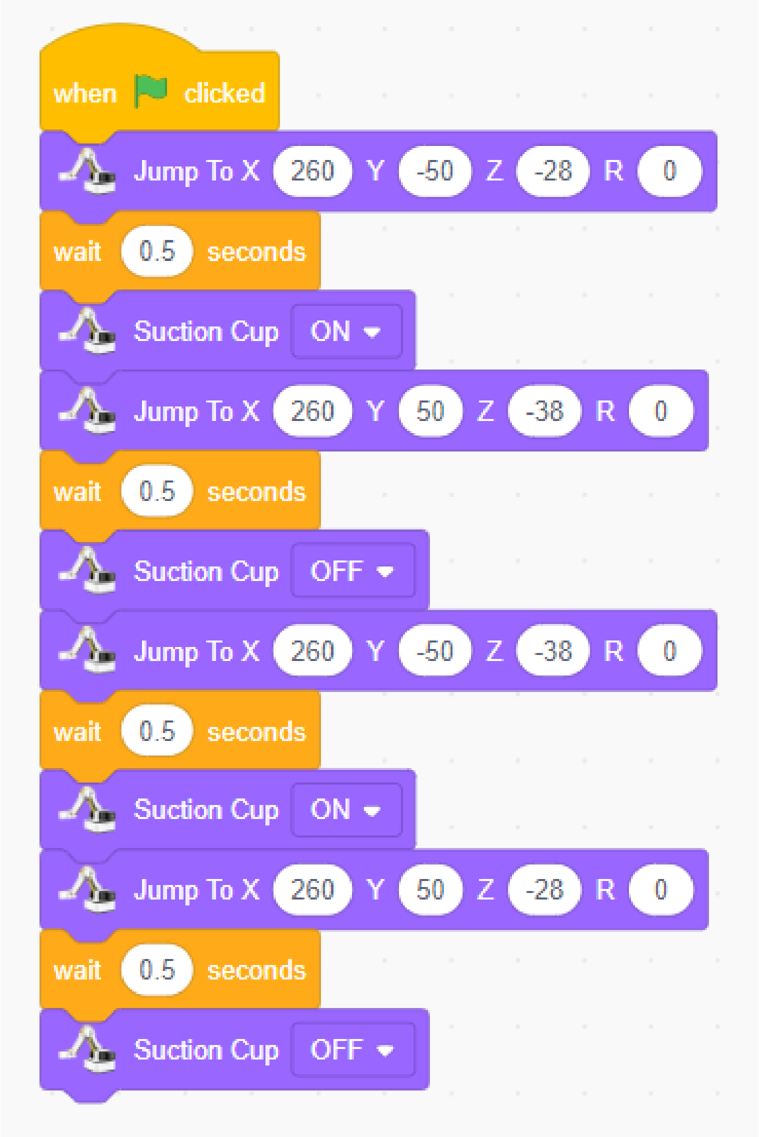


```
Jump To X 260 Y -50 Z -28 R 0
wait 0.5 seconds
Suction Cup ON
Jump To X 260 Y 50 Z -38 R 0
wait 0.5 seconds
Suction Cup OFF
```

Figure 2.43

Write a Device Script

Step 3: Splice and combine the program “carrying the first block” and “carrying the second block”, see Figure 2.44.

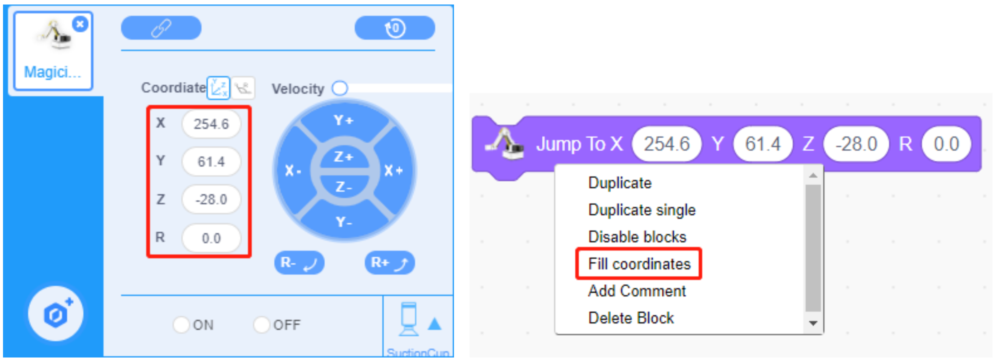


```
when 🏁 clicked
Jump To X 260 Y -50 Z -28 R 0
wait 0.5 seconds
Suction Cup ON ▾
Jump To X 260 Y 50 Z -38 R 0
wait 0.5 seconds
Suction Cup OFF ▾
Jump To X 260 Y -50 Z -38 R 0
wait 0.5 seconds
Suction Cup ON ▾
Jump To X 260 Y 50 Z -28 R 0
wait 0.5 seconds
Suction Cup OFF ▾
```

Figure 2.44

Let's read it

Move the end tool of a robot. In this case, the coordinate value on the control panel changes accordingly. We can quickly get the coordinate of the end tool of the robot by right-clicking Fill coordinates, see Figure 2.45.



Coordiate
X 254.6
Y 61.4
Z -28.0
R 0.0

Jump To X 254.6 Y 61.4 Z -28.0 R 0.0

Figure 2.45

Innovation Park

Blocks in the course are stacked. If we disperse them, what should a robot do to carry them? Write a program to carry these blocks to the loading zone from the storage zone. For the arrangement of blocks in the storage zone and the loading zone, see Figure 2.46.

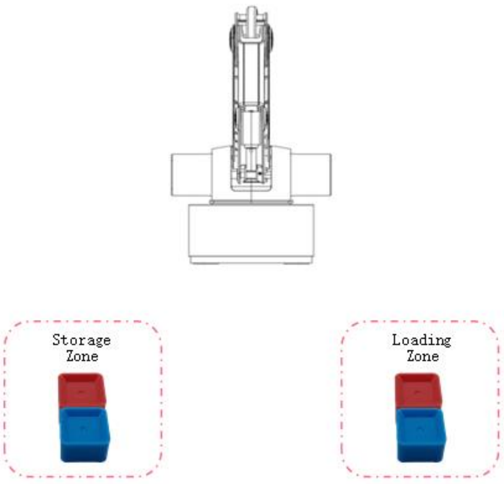


Figure 2.46

Self-Assessment Room

Content	Result
I've known the space Cartesian coordinate system	☆☆☆☆☆
I've completed the task of a "porter"	☆☆☆☆☆

Chapter 3: Data and Operation

- Smart Service of John's Supermarket

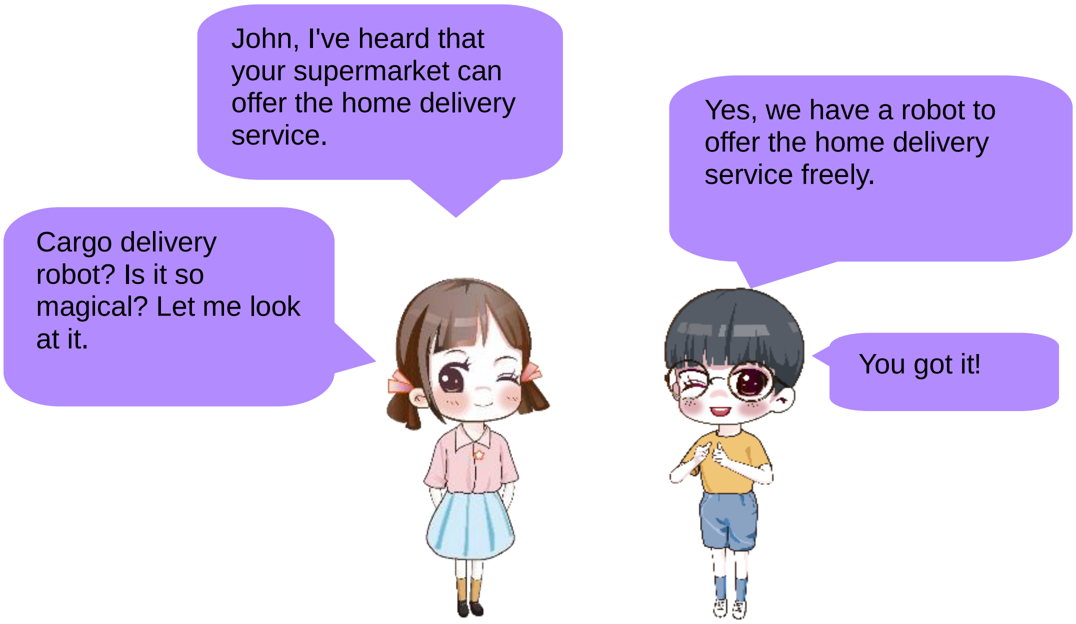
John is an IT fan. As he proudly says his supermarket is magical, his friends wonder why. Boys and girls, can you guess the answer? John has programmed cargo warehousing to automatically calculate the quantity of cargoes, and store opening to raise the supermarket popularity, and home delivery… His supermarket attracts people from the town to experience. They are curious about how John makes these functions happen.

Learning Target

- ☼ Students will learn how to store and access data in the computer.
- ☼ Students will learn how to perform data operations in the computer.
- ☼ Students will raise their ability to process and apply data, and experience the importance of data.

3.1. Program Data (1)

- Home Delivery Service


John, I've heard that your supermarket can offer the home delivery service.
Yes, we have a robot to offer the home delivery service freely.
Cargo delivery robot? Is it so magical? Let me look at it.
You got it!

Let's think about it

✧ John asks everyone to consider a question. His customers in the town live in different places. For example, Daisy's home is 140 meters from the east of the supermarket, Gary's 100 meters from the north of the supermarket, and Ella's 170 meters from the west of the supermarket. The robot remembers only one address at one time. Then, how can we make the robot remember so many addresses and deliver packages to the right home?

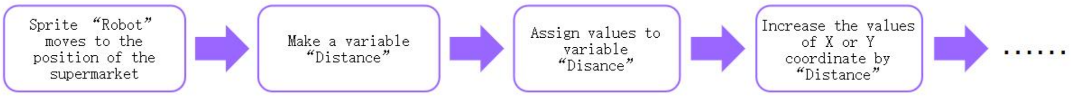
3.1.1 Variables

Today, we are going to learn variables. A varying value is called a variable. Delivery addresses change with directions and distances of customers' homes. We can indicate a distance with a variable. Thus, it will be OK if the robot knows a distance variable.

Let's do it

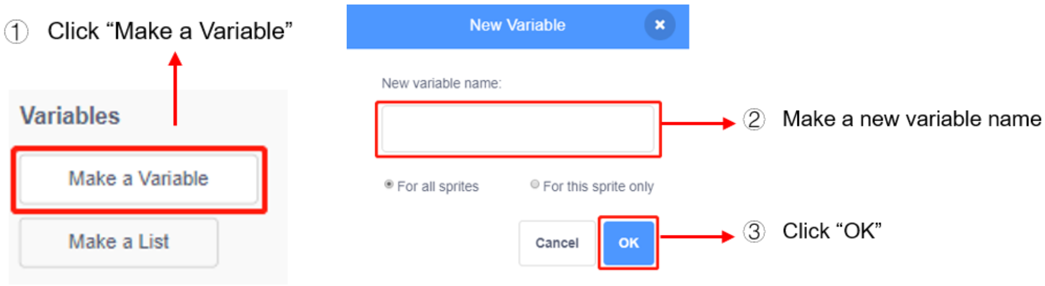
✧ Introduce the backdrop into the stage

Chart of thinking program steps:


Sprite "Robot" moves to the position of the supermarket
Make a variable "Distance"
Assign values to variable "Disance"
Increase the values of X or Y coordinate by "Distance"
……

3.1.2 Create a Variable

In DobotBlock, click Make a Variable in the Variables module. Enter a new variable name, click OK, and now we have created a variable, see Figure 3.1.



① Click "Make a Variable"

② Make a new variable name

③ Click "OK"

Figure 3.1

✅ **Let's do it**

✧ Create a "distance" variable.

Let's think about it

✧ What is a constant? Could you give any examples of variables and constants in life?

3.1.3. Assign a Value to the Variable

(1) Set the initial value of the variable

After creating a variable, we need to initialize it, assigning an initial value. We use the "set (Distance) to (0)" block to initialize the variable value, and set the initial value of the variable "Distance" to 100, see Figure 3.2.

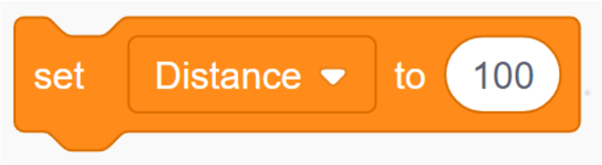

set Distance to 100

Figure 3.2

Let's do it

✧ Set the initial value of the "Distance" variable to 0.

✧ Try programming the cargo delivery by a robot.

(2) Modify the value of the variable

After a robot delivers a cargo, the device must return to the supermarket. In DobotBlock, we can modify the value of the variable in the "increase (Distance) by (1)" block and the "set (Distance) to (0)" block. Now, we can rely on the "increase (Distance) by (1)" block to increase or decrease the current value of the "Distance" variable.

Let's do it

- ✧ Write a program to allow the robot to return to the supermarket.

Tip: If the increased distance during cargo delivery is positive, the increased distance during return will be negative.

Knowledge Base

Today we've learned variables. A variable can assign a short, easy-to-remember name to each segment of data ready for use in the program to facilitate our use, so it is very helpful. The "Distance" variable used by a robot is the name of several distance data, such as 140 meters, 170 meters or 100 meters. In DobotBlock, data may fall into integer type, floating-point number type, string type and Boolean type.

For example, Cindy comes to the supermarket to buy 2 kilograms of bananas for 8.2 dollars, and she has made the payment. In this sentence, "bananas" belongs to the string type, "2" in 2 kilograms to the integer type, "8.2" in 8.2 dollars to the floating-point number type, and whether to complete the payment to the Boolean type. See Table 3.1 for four data types.

Table 3.1 Four Data Types

Data Type	Example
Integer type	0, -3, and 10
Floating-point number type	3.14, 6.18, and -2.1
String type	A, apple, I'm a student
Boolean type	True, and false

Let's read it

Boolean type originates from Boolean algebra. This type of data contains only two values: true and false. Boolean type of data usually acts as the result of a conditional judgment. For example, this expression, 5 > 3, is valid, so Boolean value returned by this expression is true. This expression, 5 < 3, is not valid, so Boolean value returned by this expression is false.

Research Laboratory

1. Task Release

John receives three orders today. Now, he must set his robot to deliver cargoes. The delivery destinations are Gary's home, Daisy's, and Ella's respectively. Every time the robot finishes delivering a cargo, it must return to the supermarket to pick up another cargo and deliver it to the next destination. Boys and girls, now let's design this program together.

2. Task Analysis

(1) Sprite interface

A robot picks up a cargo from John's supermarket and starts out. It determines the first destination as Gary's home, sets the "Distance" variable as a distance from Gary's home, moves the "Distance" toward Gary's home and reaches that destination. Then, it jumps back to the supermarket to pick up another cargo, and deliver it to the next destination.

John's home delivery service falls into three parts, see Figure 3.3.

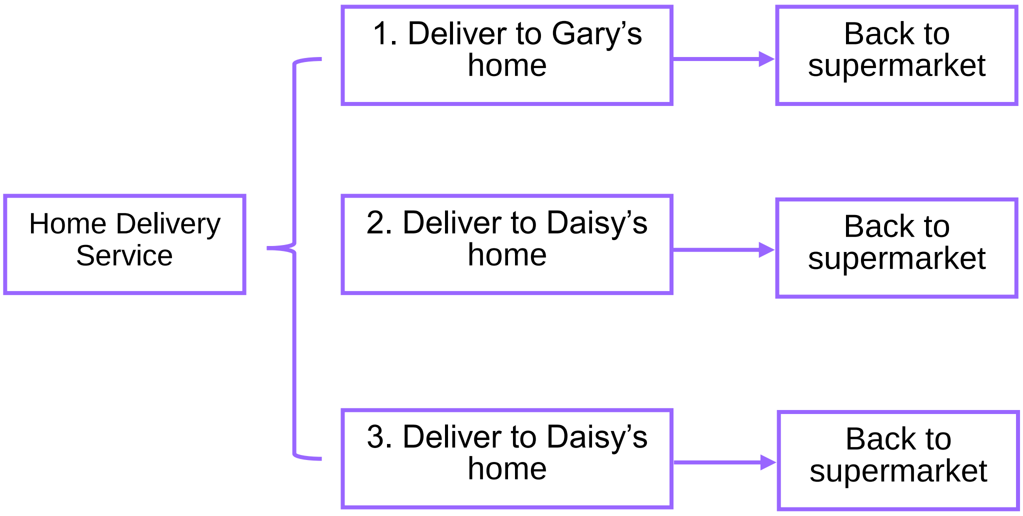


Figure 3.3

(2) Device interface

The initial position of the end of the robot is set in John's supermarket. The robot grips the cargo with its end gripper, and uses the "move along a door-shaped route to X()Y()Z()R()" block to carry the cargo to Gary's home, and release it. Then, the robot uses such block to jump back to the supermarket, and grip and carry another cargo to the next destination. See Figure 3.4 for the device arrangement position.

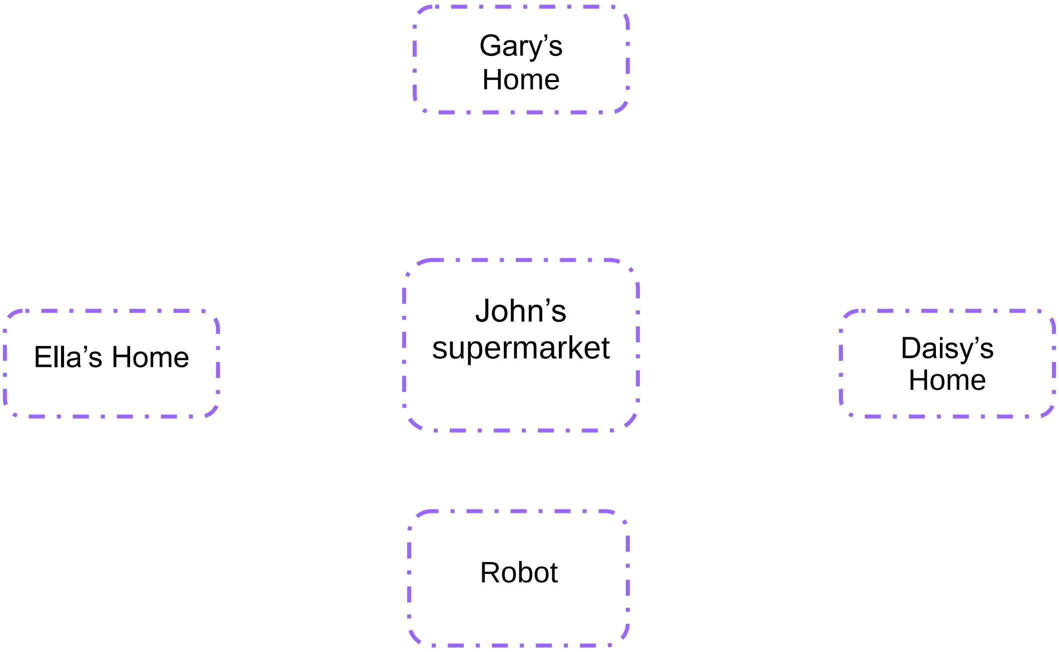
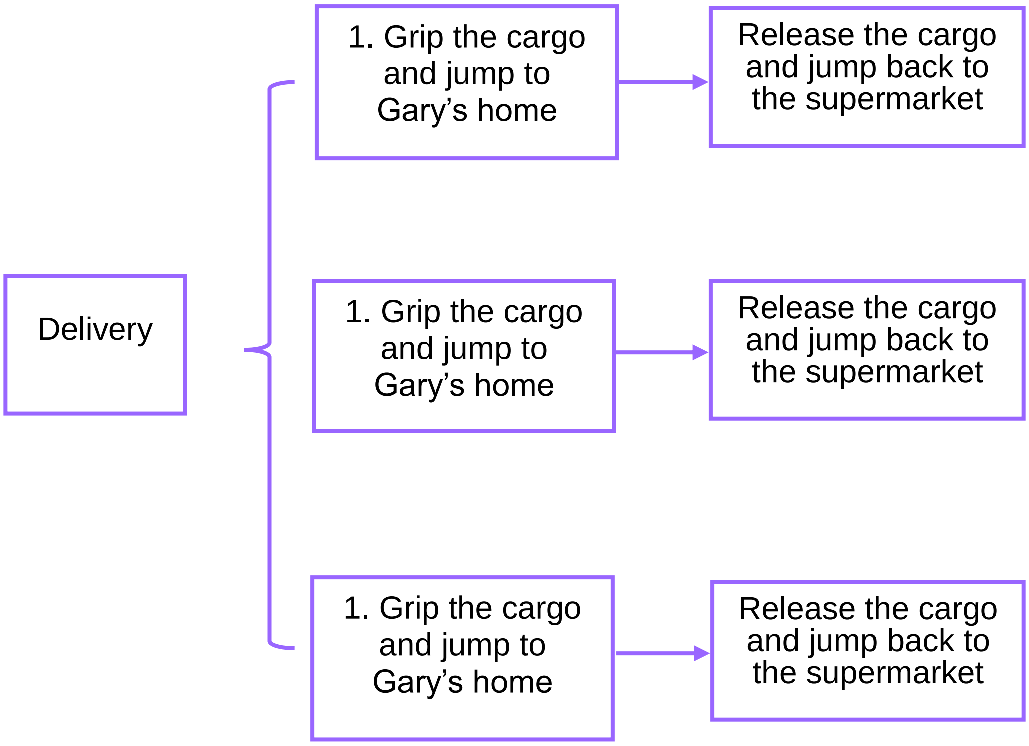


Figure 3.4

Home delivery service by the robot falls into three parts, see Figure 3.5.


Delivery

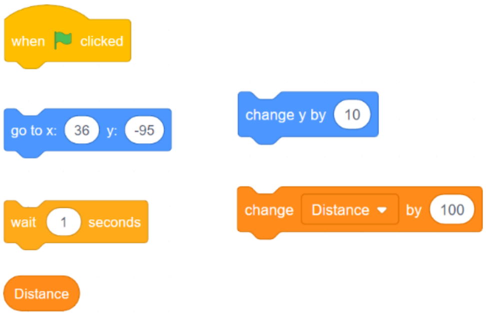
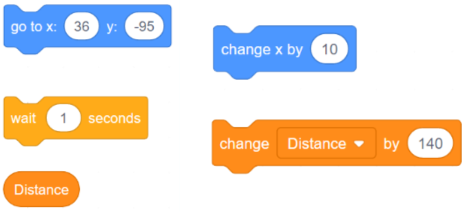
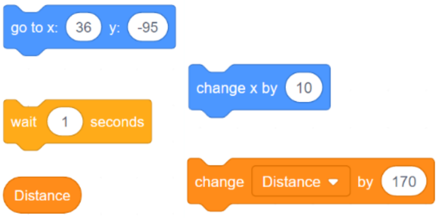
1. Grip the cargo and jump to Gary’s home → Release the cargo and jump back to the supermarket

1. Grip the cargo and jump to Gary’s home → Release the cargo and jump back to the supermarket

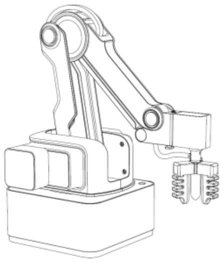
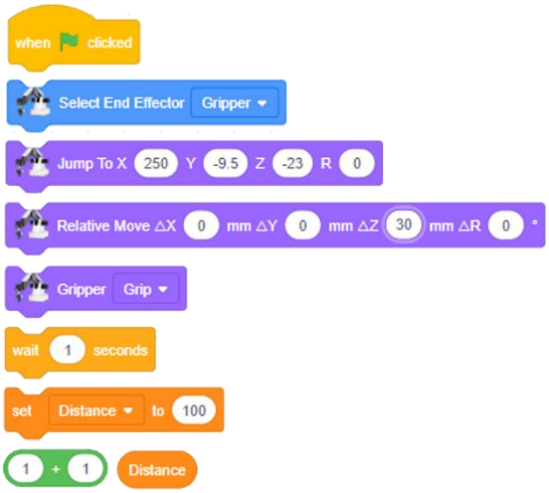
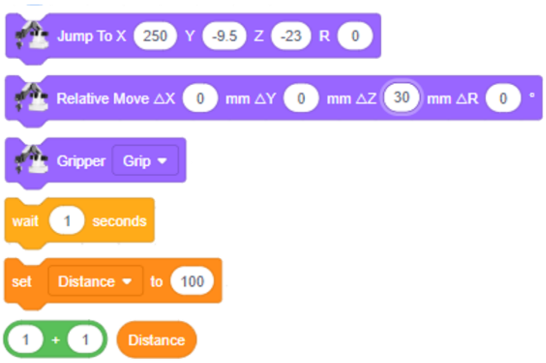
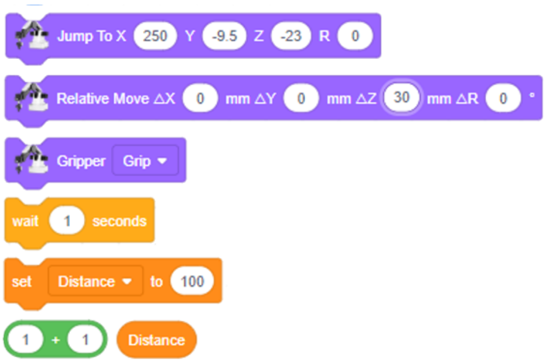
1. Grip the cargo and jump to Gary’s home → Release the cargo and jump back to the supermarket

Figure 3.5

3. Sprite Script Plan

Sprite	Task Description	Block
	1. Deliver the cargo to Gary's home	 when 🏁 clicked go to x: 36 y: -95 change y by 10 wait 1 seconds change Distance ▾ by 100 Distance
	2. Deliver the cargo to Daisy's home	 go to x: 36 y: -95 change x by 10 wait 1 seconds change Distance ▾ by 140 Distance
	3. Deliver the cargo to Ella's home	 go to x: 36 y: -95 change x by 10 wait 1 seconds change Distance ▾ by 170 Distance

4. Device Script Plan

Device	Task Description	Block
	1. Deliver the cargo to Gary's home	 when ⚑ clicked Select End Effector Gripper ▾ Jump To X 250 Y -9.5 Z -23 R 0 Relative Move ΔX 0 mm ΔY 0 mm ΔZ 30 mm ΔR 0 ° Gripper Grip ▾ wait 1 seconds set Distance ▾ to 100 1 + 1 Distance
	2. Deliver the cargo to Daisy's home	 Jump To X 250 Y -9.5 Z -23 R 0 Relative Move ΔX 0 mm ΔY 0 mm ΔZ 30 mm ΔR 0 ° Gripper Grip ▾ wait 1 seconds set Distance ▾ to 100 1 + 1 Distance
	3. Deliver the cargo to Ella's home	 Jump To X 250 Y -9.5 Z -23 R 0 Relative Move ΔX 0 mm ΔY 0 mm ΔZ 30 mm ΔR 0 ° Gripper Grip ▾ wait 1 seconds set Distance ▾ to 100 1 + 1 Distance

Processing Workshop

Upload a Stage Backdrop

Step 1: Add the backdrop. Click Upload Backdrop in the Stage List, and find the file "John's supermarket.png", and upload the file to the stage, see Figure 3.6.

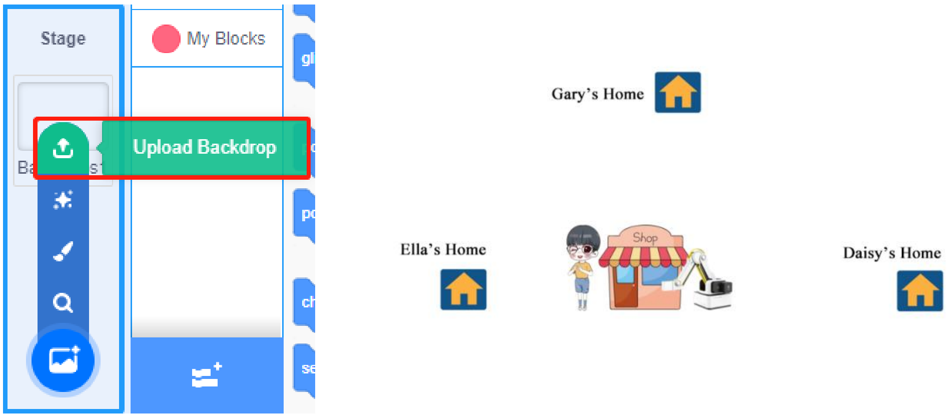
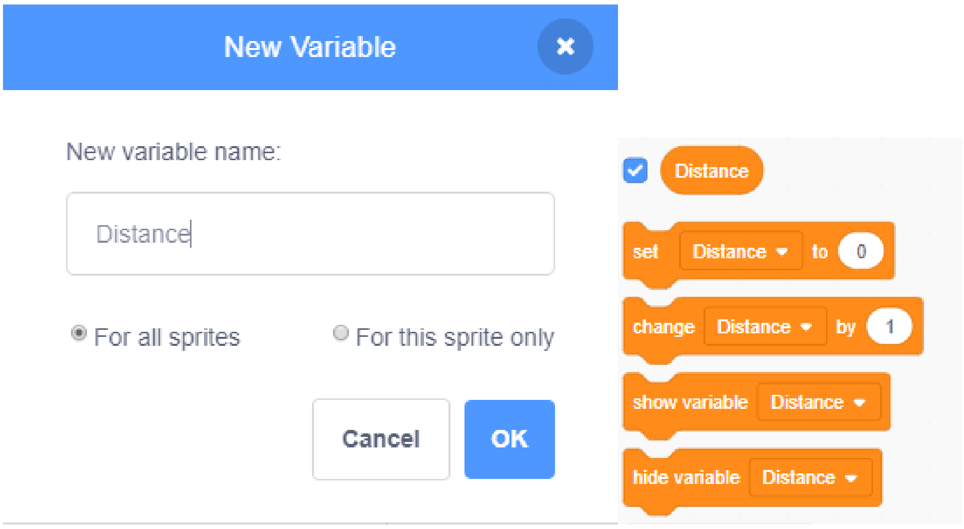


Figure 3.6

Create a Variable and List

Click Make a Variable in the Variables module, and fill in the variable name "Distance", see Figure 3.7.


New Variable

New variable name:

Distance

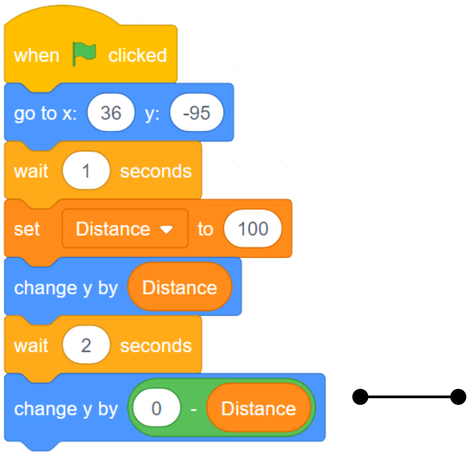
● For all sprites ○ For this sprite only

Cancel OK

Figure 3.7

Write a Sprite Script

Deliver the cargo to Gary's home. For the program of the corresponding sprite, see Figure 3.8.



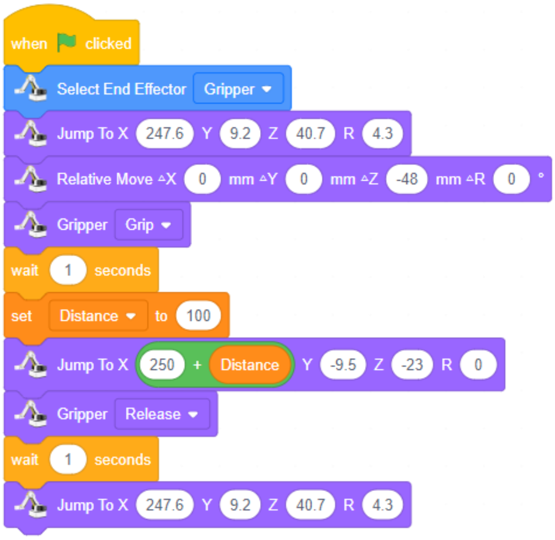
```
when 🏁 clicked
go to x: 36 y: -95
wait 1 seconds
set Distance ▾ to 100
change y by Distance
wait 2 seconds
change y by 0 - Distance
```

Think about why "0-Distance" enables the robot to jump back to the supermarket?

Figure 3.8

There is a similarity between the program to deliver the cargo to Gary's home and the one to deliver the cargo to Daisy's and Gary's homes. Write the program for delivery to Daisy's and Gary's homes.

Deliver the cargo to Gary's home and jump back to the supermarket. For the program of the corresponding sprite in this case, see Figure 3.9.



```
when 🏁 clicked
Select End Effector Gripper ▾
Jump To X 247.6 Y 9.2 Z 40.7 R 4.3
Relative Move ΔX 0 mm ΔY 0 mm ΔZ -48 mm ΔR 0 °
Gripper Grip ▾
wait 1 seconds
set Distance ▾ to 100
Jump To X 250 + Distance Y -9.5 Z -23 R 0
Gripper Release ▾
wait 1 seconds
Jump To X 247.6 Y 9.2 Z 40.7 R 4.3
```

Figure 3.9

There is a similarity between the program to handle the cargo to Gary's home and the one to handle the cargo to Daisy's and Ella's homes. Boys and girls, please write the program for the later delivery yourselves.

Innovation Park

To modify the value of a variable, we can not only "set (Distance) to 0", but also use "increase (Distance) by (1)". In the task of home delivery, can we implement the function of "increase (Distance) by (1)" or both "set (Distance) to 0" and "increase (Distance) by (1)"? Please consider this question. And perform and complete this task yourselves.

Self-Assessment Room

Content	Result
I've understood data types	☆☆☆☆☆
I've known the definition of the variable	☆☆☆☆☆
I've learned how to create a variable	☆☆☆☆☆
I've learned how to initialize a variable and to modify its value	☆☆☆☆☆
I've completed the task of "home delivery service"	☆☆☆☆☆

3.2. Program Data (2)

- Store Opening

Why is this place so bustling with a sea of people?
My supermarket is launching a lucky draw for store opening ceremony. Only if you press the draw button, the chosen prize appears in the screen. Come try your luck. Maybe, you will win a grand prize!

John wants you to think about two questions. How does the draw system randomly offer awards? How does the draw system store so many prizes?

1. List

A variable can store a single value. But if you desire to store a series of values, the variable may fall short of your expectation. For example, there are 20 prizes. Supposing you intend to store the names of 20 prizes, the program will have to use 20 variables. This is a complicated job. But we can rely on a list to solve this problem.

(1) Define a List

A list is a container storing multiple variables. You can store and get the value of each variable in the container.

To create a list, we first name it, and then access each variable in the list based on the variable position.

Next, let's create and use a list.

(2) Create a list and add data

Click Make a List in the Variables module, see Figure 3.10. Fill in the list name. Here, we name the list "Prize List", see Figure 3.11.

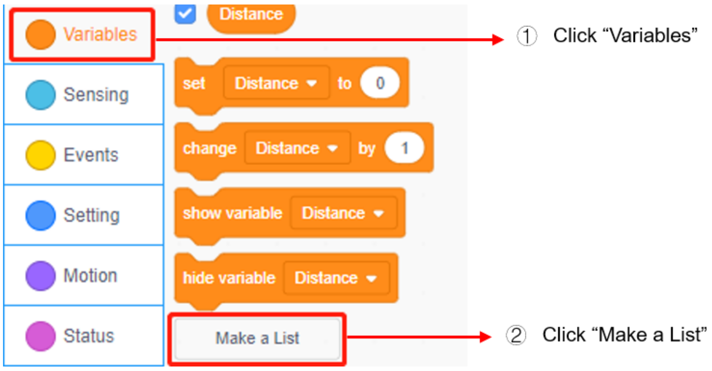


Figure 3.10

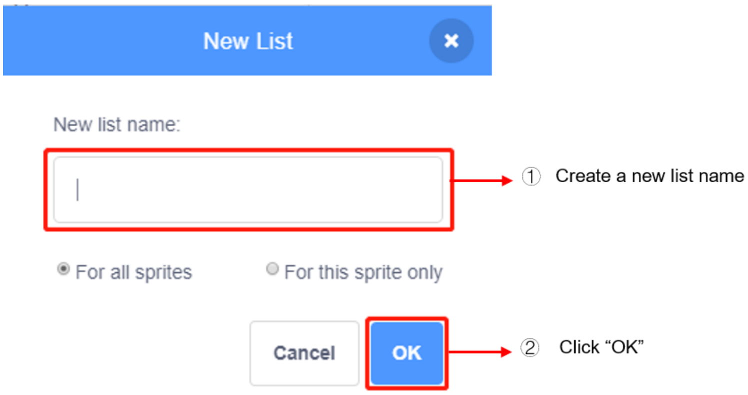


Figure 3.11

After we create a list, the list is initially empty. So its length is 0. We can click the "+" sign in Figure 3.12 to add a new variable, and change the size of this interface by dragging the equality sign with a mouse.

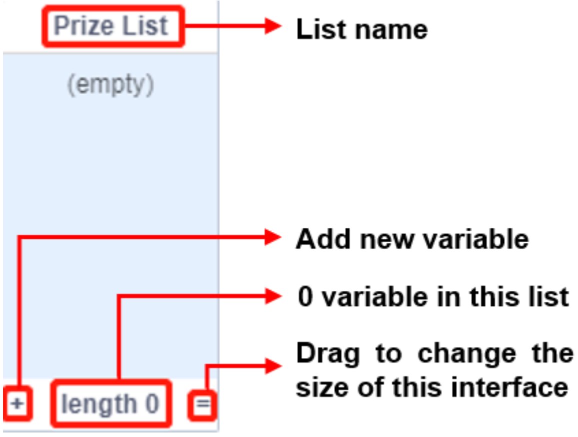
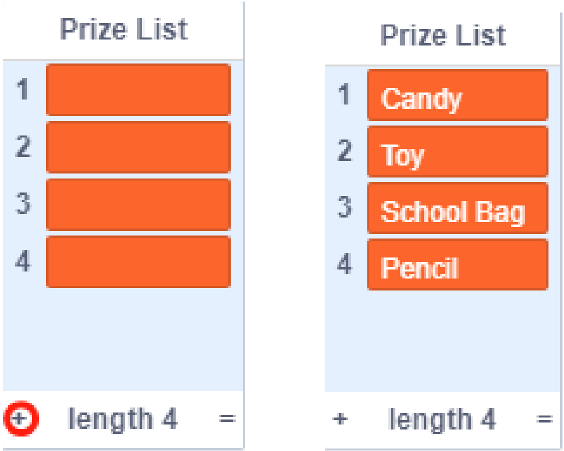


Figure 3.12

If the Prize List in the list contains four different prizes, we can create four variables by clicking the "+" sign in the lower left corner four times. Each variable is an element of the list. In the editing brackets, enter the names of the prizes, namely, candy, toy, school bag, and pencil, see Figure 3.13.



Prize List

1
2
3
4

length 4

Prize List

1 Candy
2 Toy
3 School Bag
4 Pencil

length 4

Figure 3.13

Let's do it

✧ Create Prize List to store the names of the prizes.

Research Laboratory

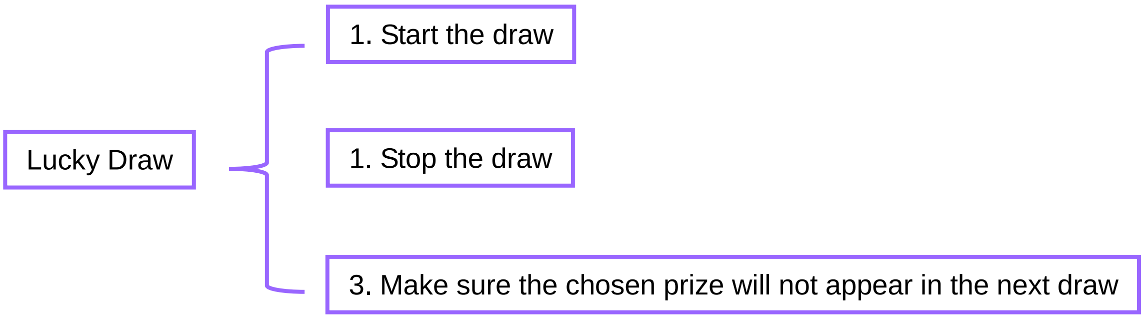
1. Task Release

Create a Lucky Draw System: John's supermarket launches a special offer for opening. Please press the luck button, and you may draw a lucky prize.

2. Task Analysis

The draw system provides the Start sprite and the Stop sprite. Start functions to start the draw. After you click Start, the prize name will quickly roll on the stage. Stop functions to stop the draw. After you click Stop, the name of the chosen prize will appear on the stage. The chosen prize will not appear in the next draw.

The draw system falls functionally into three tasks, see Figure 3.14.



Lucky Draw

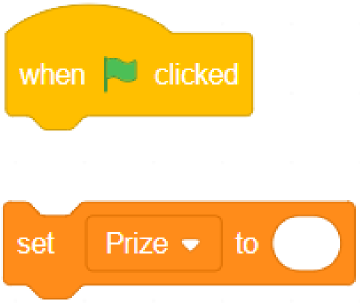
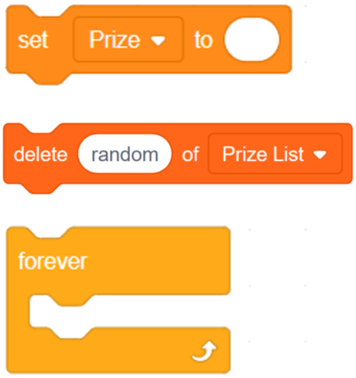
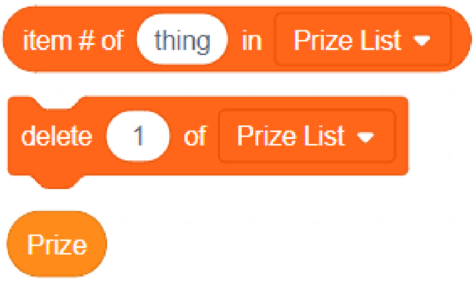
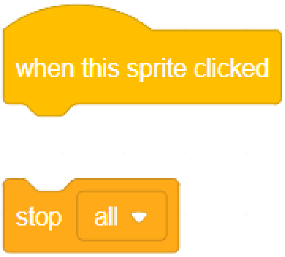
1. Start the draw

1. Stop the draw

3. Make sure the chosen prize will not appear in the next draw

Figure 3.14

3. Script Plan

Sprite	Task Description	Block
 Start	1. Initialize the value of a variable	 when ⚑ clicked set Prize ▾ to ( )
	2. Start the draw	 set Prize ▾ to ( ) delete (random) of Prize List ▾ forever
	3. Delete the chosen prize	 item # of (thing) in Prize List ▾ delete (1) of Prize List ▾ Prize
 Stop	Stop draw	 when this sprite clicked stop all ▾

Processing Workshop

Add a Sprite and Backdrop

Step 1: Add a sprite. Click Upload Sprite in the Sprite List, find the files “Start” and “Stop”, upload the files to the Sprite List, and delete the default sprite "Sprite 1", see Figure 3.15.

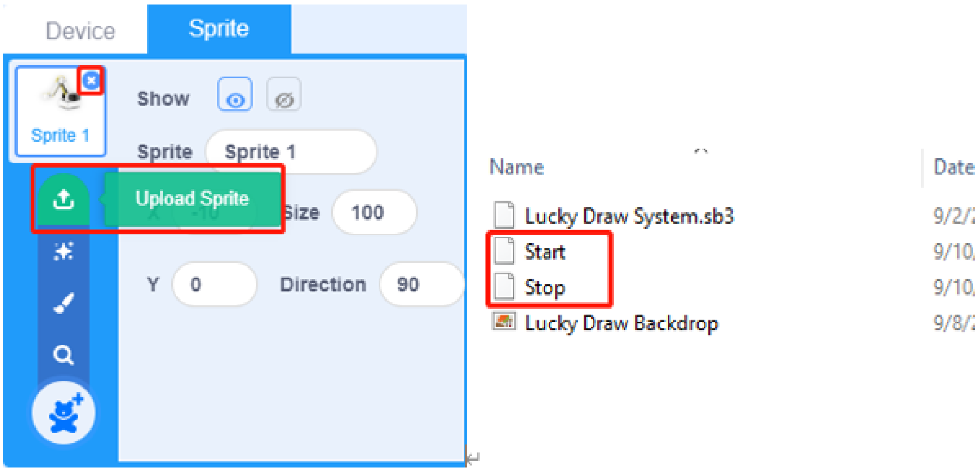


Figure 3.15

Add a Sprite and Backdrop

Step 2: Add a backdrop. Click Upload Backdrop in the Sprite List, find the file "Lucky Draw Dropback.png", and upload the backdrop "Lucky Draw Dropback.png" to the stage, see Figure 3.16.

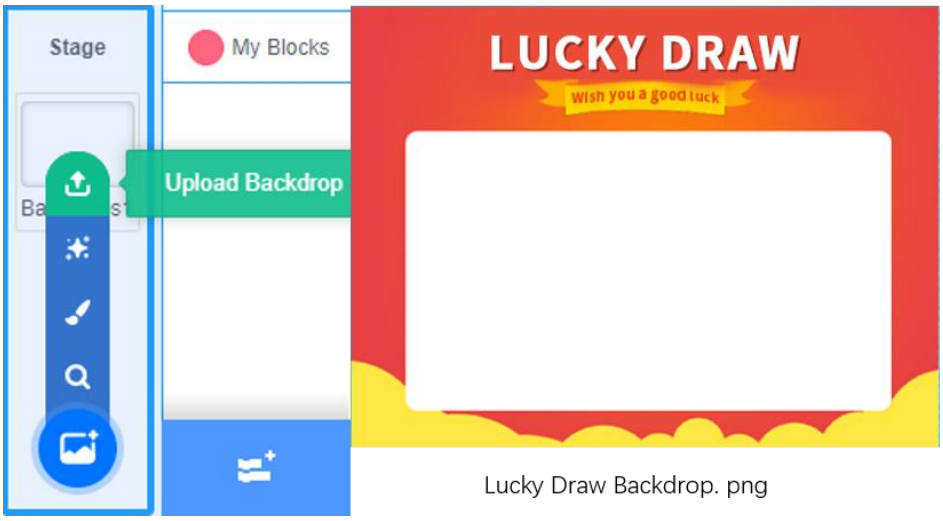
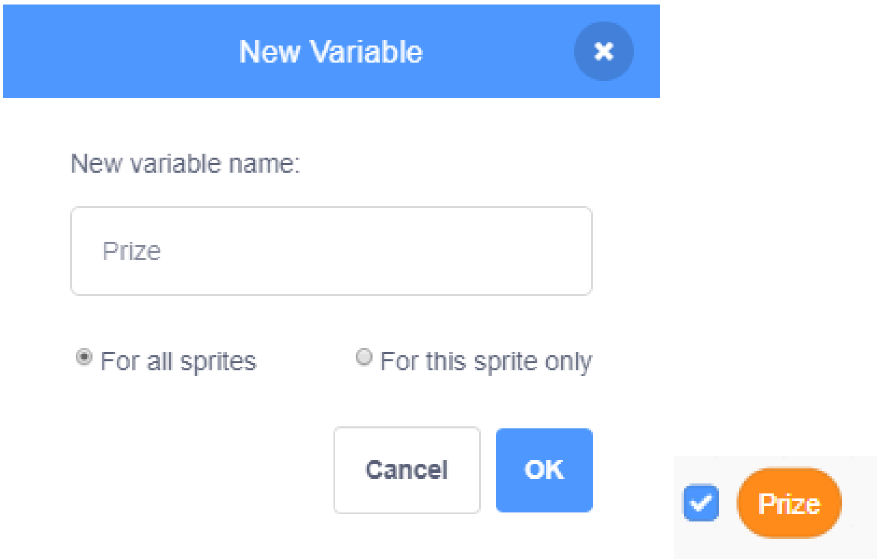


Figure 3.16

Create a Variable and List

Click Make a Variable in the Variables module, and fill in the variable name "Prize", see Figure 3.17.



New Variable

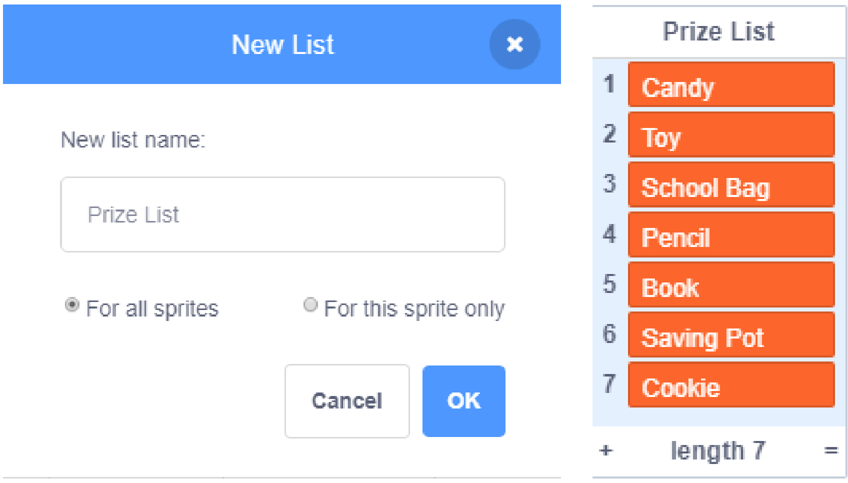
New variable name: Prize

● For all sprites ○ For this sprite only

Figure 3.17

Create a Variable and List

Step 2: Create Prize List. Click Make a List in the Variables module, fill in the variable name "Prize", and edit the elements of the list "Prize List", see Figure 3.18.



New List

New list name: Prize List

● For all sprites ○ For this sprite only

Prize List

1	Candy
2	Toy
3	School Bag
4	Pencil
5	Book
6	Saving Pot
7	Cookie

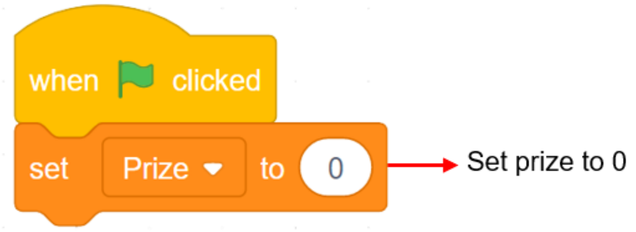
Figure 3.18

Write a Sprite Script

Step 1: Edit the Start program of the sprite.

The program of the Start sprite falls into two parts: initialize the value of a variable and implement the start draw function. The two parts should be programmed separately.

When the green flag is clicked, we can set the variable prize to 0, see Figure 3.19.



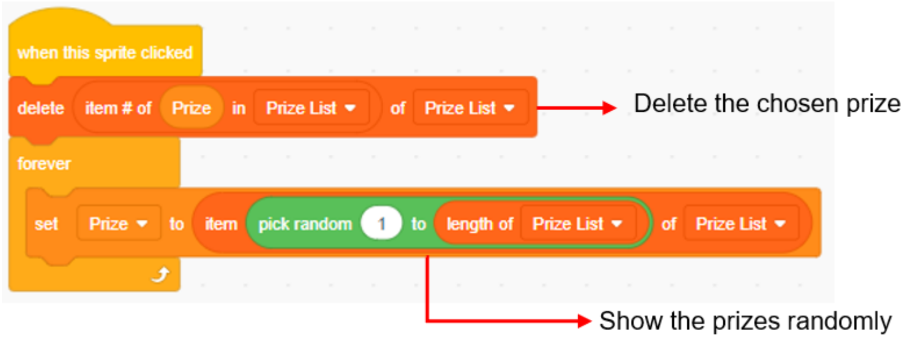
```
when ⚑ clicked
set Prize to 0
```

Set prize to 0

Figure 3.19

When the sprite Start is clicked, we can start the draw. Delete the chosen prize, and allow the prizes to appear randomly, see Figure 3.20.

Write a Sprite Script



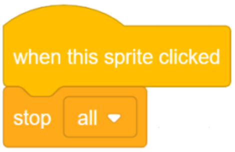
```
when this sprite clicked
delete item # of Prize in Prize List of Prize List
forever
    set Prize to item pick random 1 to length of Prize List of Prize List
```

Delete the chosen prize

Show the prizes randomly

Figure 3.20. Starting the draw

Step 2: Write the program of the sprite Stop. When this sprite Stop is clicked, we can stop the draw, and all scripts, see Figure 3.21.



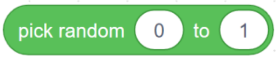
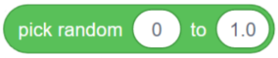
```
when this sprite clicked
stop all
```

Figure 3.21

Let's read it

To allow the draw system to show the prizes randomly, we use the "pick random (0) to (10)" block. This block randomly generates a number every time. Of the numbers generated, the minimum value is 0, and the maximum value is 10. These two numbers form the range of randomly picked values. Table 3.2 gives the use case of the random number block.

Table 3.2

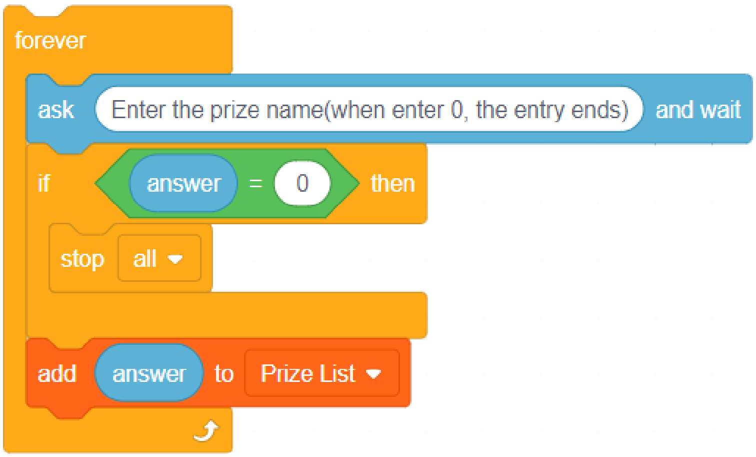
Sample	Output Result
 pick random 0 to 1	{0, 1}
 pick random 0 to 1.0	{0, 0.1, 0.15, 0.2, 0.268, 0.3521…1.0}

Note: "pick random (0) to (1)" differs from "pick random (0) to (1.0)". Although the maximum value 1 is equal to 1.0, 1 is the integer type of data, 1.0 the floating-point type of data. Thus, the random numbers you obtain from both are accordingly the integer type of data and the floating-point type of data.

Innovation Park

The draw system in the preceding case offers seven types of prizes. Generally, however, we will arrange multiple draw links in the evening party, with different quantities and names of prizes. Four types of prizes may be available to the draw link at the opening of the evening party. Two types of prizes may be available to the draw link in the middle of the evening party. One type of prize may be available to the draw link at the end of the evening party. Then, how does the program learn about the quantities and names of prizes at each link?

We can add a link to ask the user for the prize name. We take a special value (e.g., 0) as the end of the list, and include the prize name answered each time in the list, see Figure 3.22.



```
forever
    ask (Enter the prize name(when enter 0, the entry ends)) and wait
    if <(answer) = (0)> then
        stop [all ▾]
    add (answer) to [Prize List ▾]
```

Figure 3.22

Self-assessment Room

Content	Result
I've understood the definition of the list	☆☆☆☆☆
I've learned how to create a list	☆☆☆☆☆
I've known about the list-related blocks	☆☆☆☆☆
I've completed the task of a draw system	☆☆☆☆☆

3.3. Data Operation (1)

- Holiday Activity (1)

It's summer vacation. Why are there so many lovely kids in the supermarket?
We are having a holiday activity. The theme for the first week is math. Kelly, come answer questions, and let's see who will finish them faster.

John wants you to think about a question. If we want to use a computer to help us with math problems, what shall we do?

3.3.1. Arithmetic Operators

The computer program supports four basic arithmetic operations: addition (+), subtraction (-), multiplication (*) and division (/), see Figure 3.23.

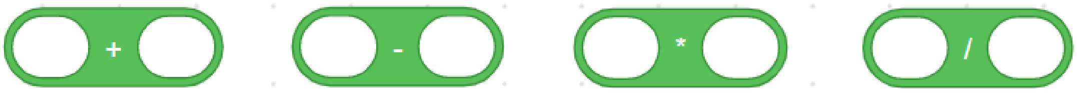


Figure 3.23

We can use "say (Hello!)" block to show the results of the four operations, see Figure 3.24.

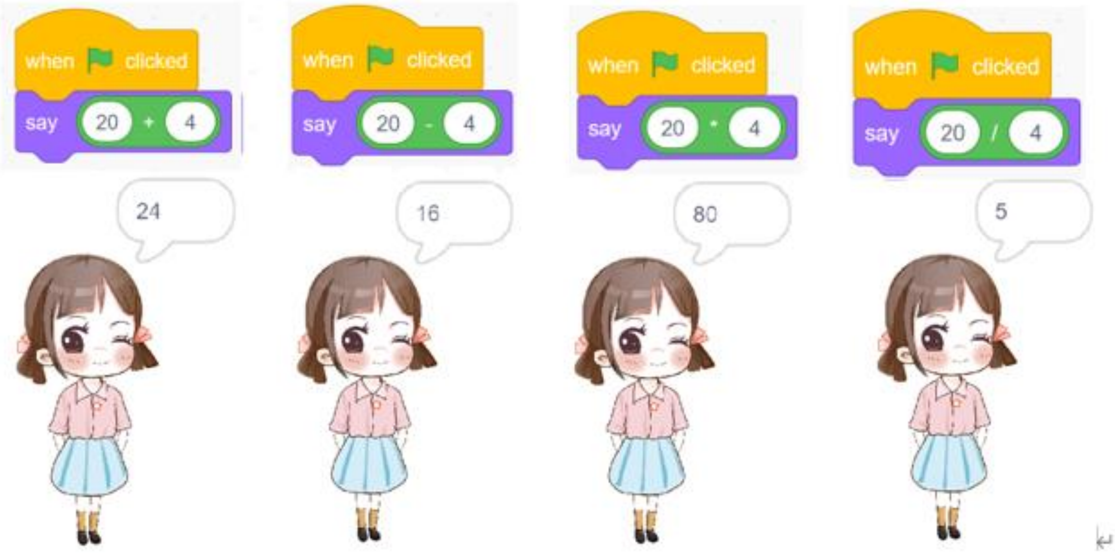


Figure 3.24

Let's do it

- ✧ Try using arithmetic operators according to Figure 3.24.
- ✧ The host starts to raise the first round of questions, asking students to write programs. Let's look at who answers the questions most quickly.

78*5=   999/28=   13567+65327=   7782589-2132413=

Let's think about it

- ✧ The host starts to raise the second round of questions, asking students to calculate a remainder. How do we program the remainder calculation?

Arithmetic operators in DobotBlock also include remainder operations. For the block, see Figure 3.25.

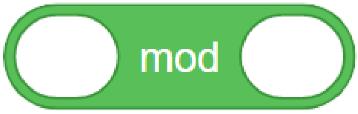
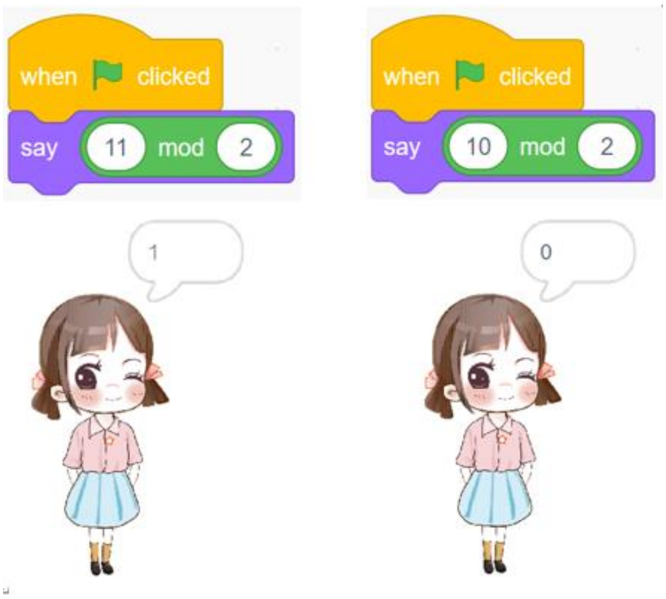


Figure 3.25

Through a remainder operation we can get a remainder after the division between two numbers. The remainder of an exact division expression is 0. For example, the reminder of 11 divided by 2 is 1, and the remainder of 10 divided by 2 is 0, see Figure 3.26.



```
when ⚑ clicked
say (11) mod (2)
```

```
when ⚑ clicked
say (10) mod (2)
```

Figure 3.26

Let's do it

- Program the remainder calculation according to Figure 3.26.
- The host formally starts to raise the second round of questions, asking students to calculate the remainders of the following formulas. Let's look at who answers these questions most quickly. Calculate the remainder of 568 divided by 15, that of 236 divided by 6, that of 333 divided by 12, and that of 789327 divided by 23

3.3.2. Comparison Operators

Let's think about it

- The third round of questions raised by the host is to decide whether the result of an inequality is true. If yes, the result is true; otherwise, it is false. Then, how do we program this condition?

In math, we often use the inequality operation signs like greater than sign (>), less than sign (<) and equality sign (=) to compare numbers. In DobotBlock, we often use comparison operators like greater than sign (>), less than sign (<) and equality sign (=) to compare the relations between numbers, between variables, between expressions or between characters. Comparison operators are in the Operators module, see Figure 3.27.

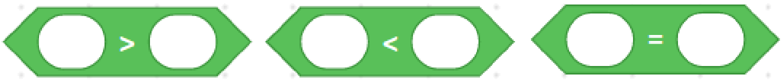


Figure 3.27

Let's read it

The block of comparison operators is hexagonal. The hexagonal block in DebotBlock is also called Boolean expression. The result returned by Boolean expression only contains two items: true and false. The returned result is called Boolean result.

We can use the "say (Hello!)" block to show the comparison result of 3 and 9 by a comparison operator, see Figure 3.28.

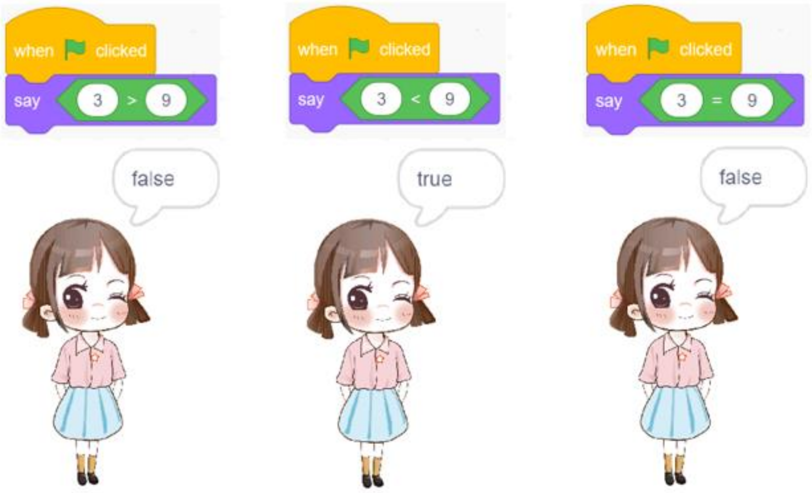


Figure 3.28

Let's do it

✧ Program the comparison operation according to Figure 3.28.

3.3.3. Logical Operators

Let's think about it

✧ Only when your answers to two inequalities are both true can you qualify for the next round of contest. How do you program this condition?

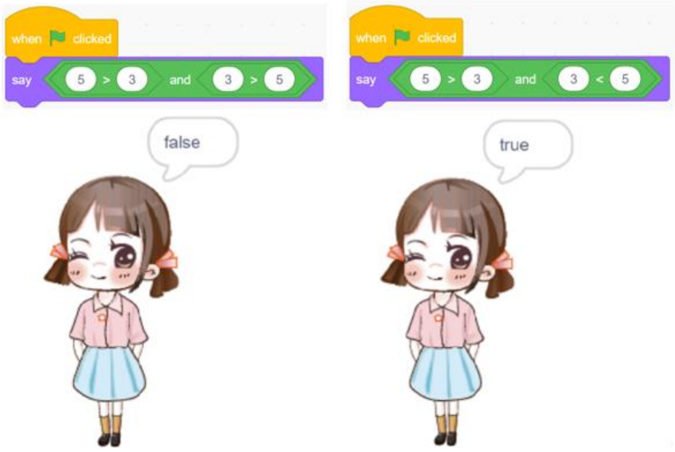
In programming, we often use logical operators, "and", "or" and "not" to connect two or multiple Boolean expressions.

For the three logical operators and their meanings, see Figure 3.3.

Table 3.3

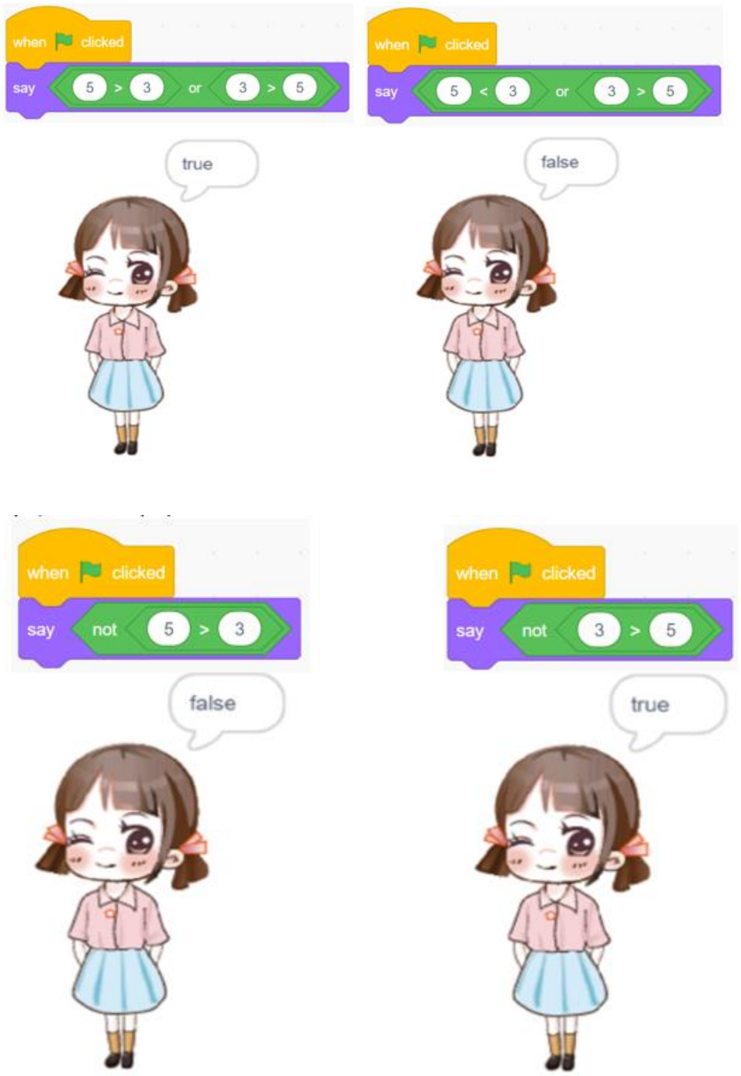
Operator	Meaning
and	Only when two Boolean expressions are both true are their results true.
or	If only one Boolean expression is true, the result will be true.
not	When the Boolean expression is false, the result will be true.

As we know, the expressions "3 < 5" and "5 > 3" are true, whereas "5 < 3" and "3 > 5" are false. The two ones are connected with logical operators. For the program and its running result, see Figure 3.29.



```
when ⚑ clicked
say <(5) > (3) and (3) > (5)>
```

```
when ⚑ clicked
say <(5) > (3) and (3) < (5)>
```



```
when ⚑ clicked
say 5 > 3 or 3 > 5
```

true

```
when ⚑ clicked
say 5 < 3 or 3 > 5
```

false

```
when ⚑ clicked
say not 5 > 3
```

false

```
when ⚑ clicked
say not 3 > 5
```

true

Figure 3.29

Let's do it

✧ Only when your answers to two inequalities are both true can you qualify for the next round of contest. Based on the program in Figure 3.29, decide which logical operator you should choose to achieve the above purpose.

In math, we often compare numbers with greater than or equality sign (≥), less than or equality sign (≤). In DobotBlock, however, there are not these two signs. In this case, how do we implement the functions of such two signs?

For this, we can use logical operators. For example, to express the range of x ≥ 7, we can use "or" or "not", see Figure 3.30.

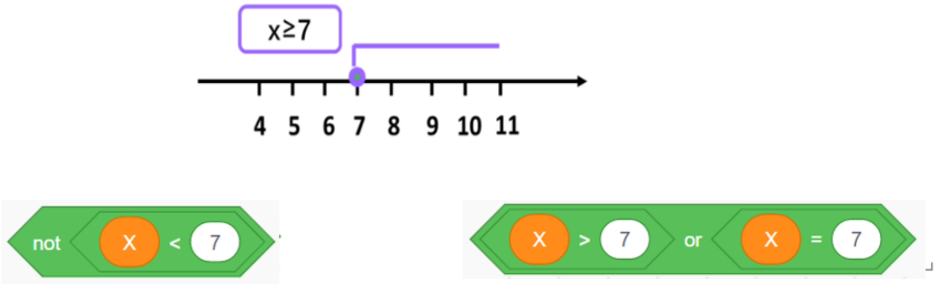


Figure 3.30

Let's do it

- Try using logical operators to denote x ≤ 7 and 5 ≤ x ≤ 7.

Let's read it

To perform more mathematic operations, DobotBlock offers rounding off, random numbers, absolute values and among other mathematical functions. For example, from the "absolute value ()" block in the Operators module, we can draw down a menu, and find mathematical functions like "sqrt", "trigonometric functions", "log" and "exponent", see Figure 3.31.

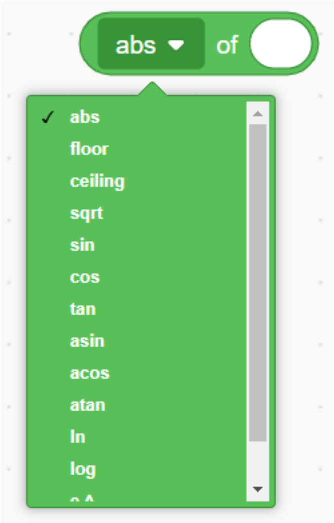


Figure 3.31

Research Laboratory

1. Task Release

Make a Simple Calculator: We can design a simple calculator with DobotBlock to perform the basic operations of addition, subtraction, multiplication and division.

2. Task Analysis

The simple calculator can perform three tasks, see Figure 3.32.

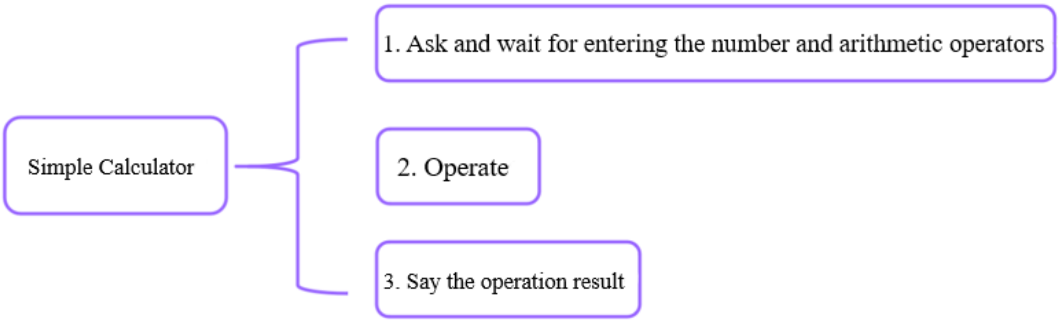
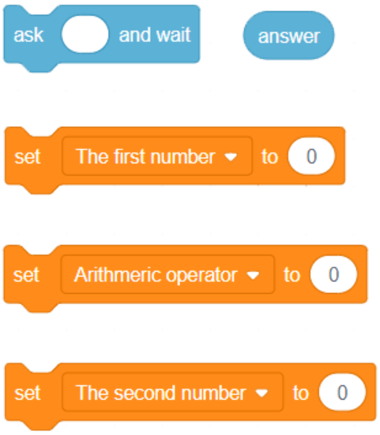
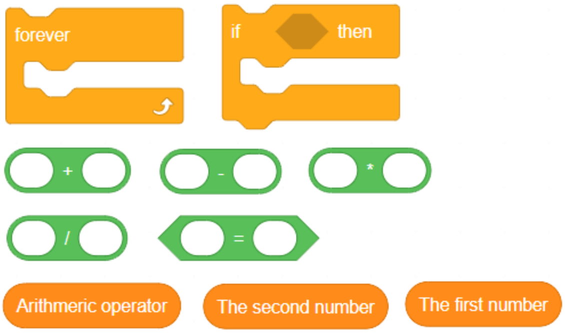
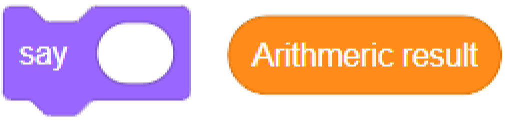

Simple Calculator
1. Ask and wait for entering the number and arithmetic operators
2. Operate
3. Say the operation result

Figure 3.32

3. Script Plan

Sprite	Task Description	Block
	1. Ask and wait for entering the number and mathematical operators	 ask ( ) and wait answer set The first number ▾ to 0 set Arithmeric operator ▾ to 0 set The second number ▾ to 0

Sprite	Task Description	Block
	2. Operate	 forever if then + - * / = Arithmetic operator The second number The first number
	3. Say the operation result	 say Arithmetic result

Processing Workshop

Add a Sprite

In the Sprite List, add the sprite "Kelly (1)", and delete the default sprite "Sprite 1", see Figure 3.33.

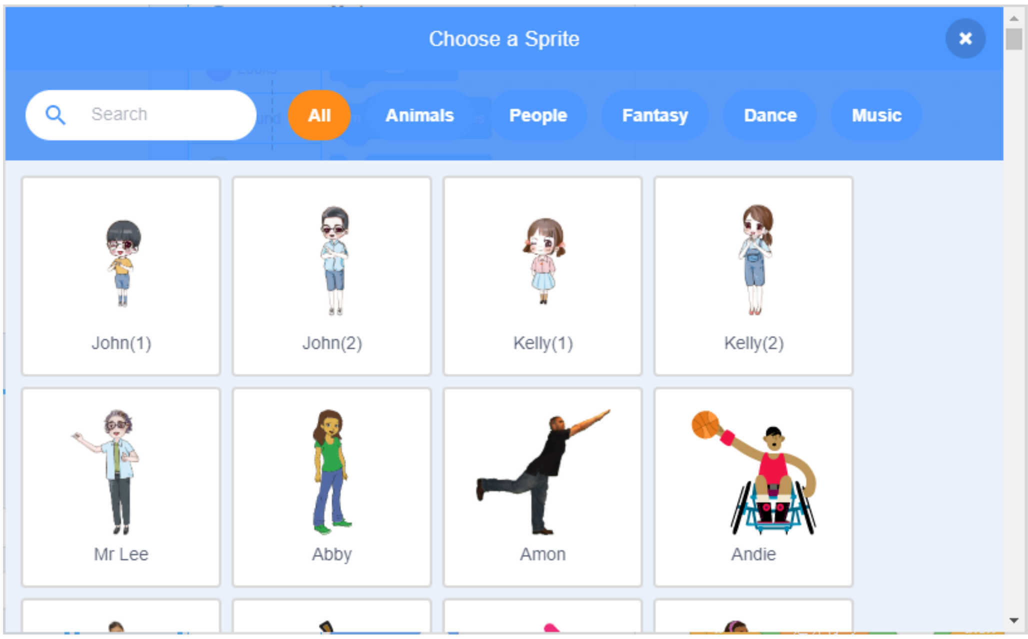
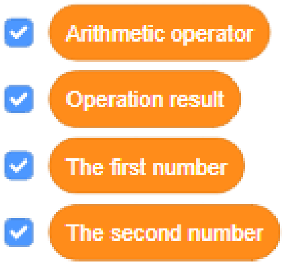


Figure 3.33

Create a Variable

In the Variables module, click Make a Variable and we can create four variables, and name them The first number, The second number, Arithmetic operator and Operation result, see Figure 3.34.



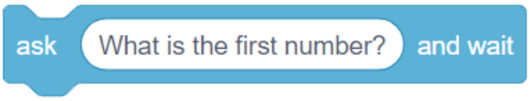
☑ Arithmetic operator
☑ Operation result
☑ The first number
☑ The second number

Figure 3.34

Write a Sprite Script

Step 1: Ask and wait for entering the number and mathematical operators.

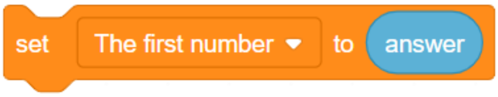
Find the "ask (What is your name?) and wait" block in the Sensing module, and enter the inquiry content "What is the first number?" in such module, see Figure 3.35.



```
ask (What is the first number?) and wait
```

Figure 3.35

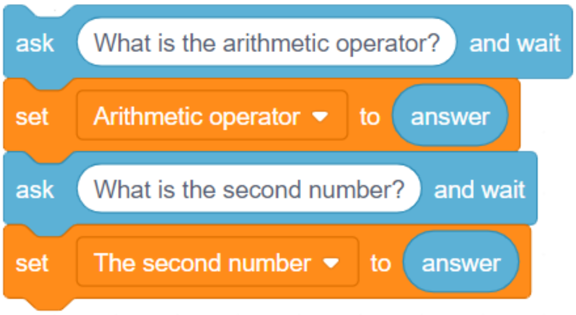
Set the value of the variable "The first number" to answer, see Figure 3.36.



```
set [The first number ▾] to (answer)
```

Figure 3.36

Similarly, ask the arithmetic operator and the second number, see Figure 3.37.

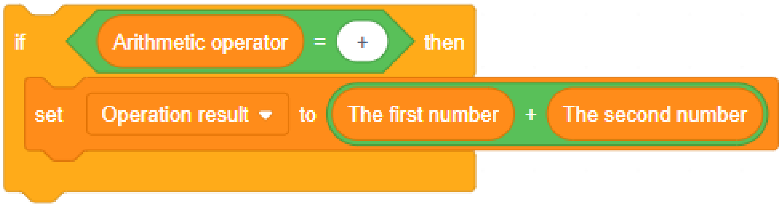


```
ask (What is the arithmetic operator?) and wait
set [Arithmetic operator ▾] to (answer)
ask (What is the second number?) and wait
set [The second number ▾] to (answer)
```

Figure 3.37

Step 2: Operate.

Decide operators, and then perform related operations. For example, decide whether the variable "Arithmetic operator" is "+". If yes, perform the addition operation on the first number and the second number, see Figure 3.38.

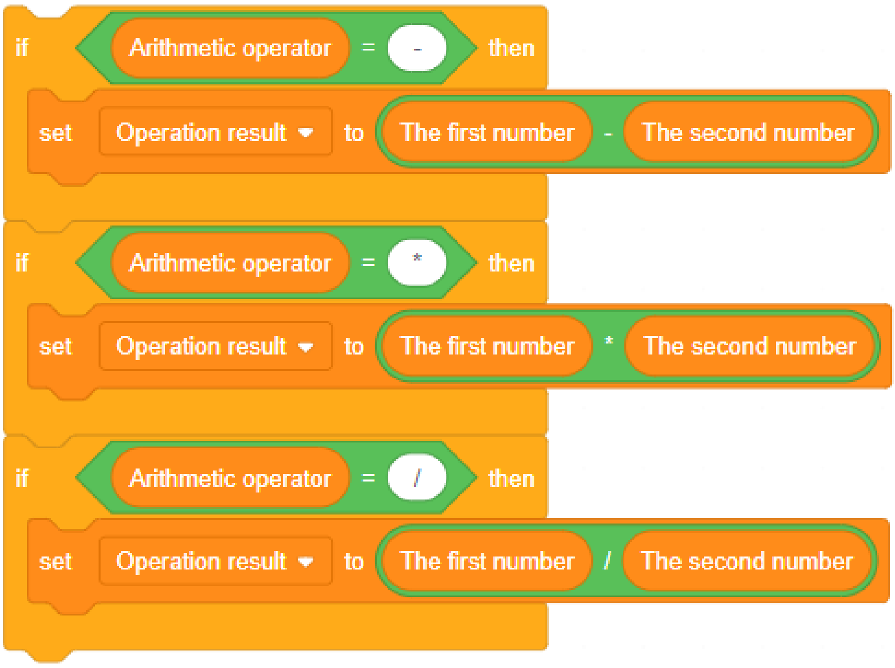


```
if <(Arithmetic operator) = (+)> then
    set [Operation result ▾] to ((The first number) + (The second number))
```

Figure 3.38

Write a Sprite Script

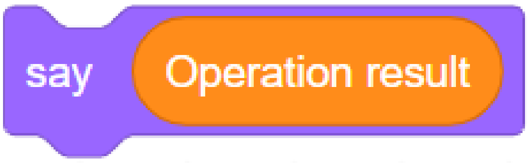
Similarly, if we decide "Arithmetic operator" as "-", "*" or "/", we can perform corresponding operations on the first number and the second number, see Figure 3.39.



```
if Arithmetic operator = - then
    set Operation result to The first number - The second number
if Arithmetic operator = * then
    set Operation result to The first number * The second number
if Arithmetic operator = / then
    set Operation result to The first number / The second number
```

Figure 3.39

Step 3: Say the operation result. Find the "say (Hello!)" block in the Looks module, and allow the sprite to say the operation result, see Figure 3.40.

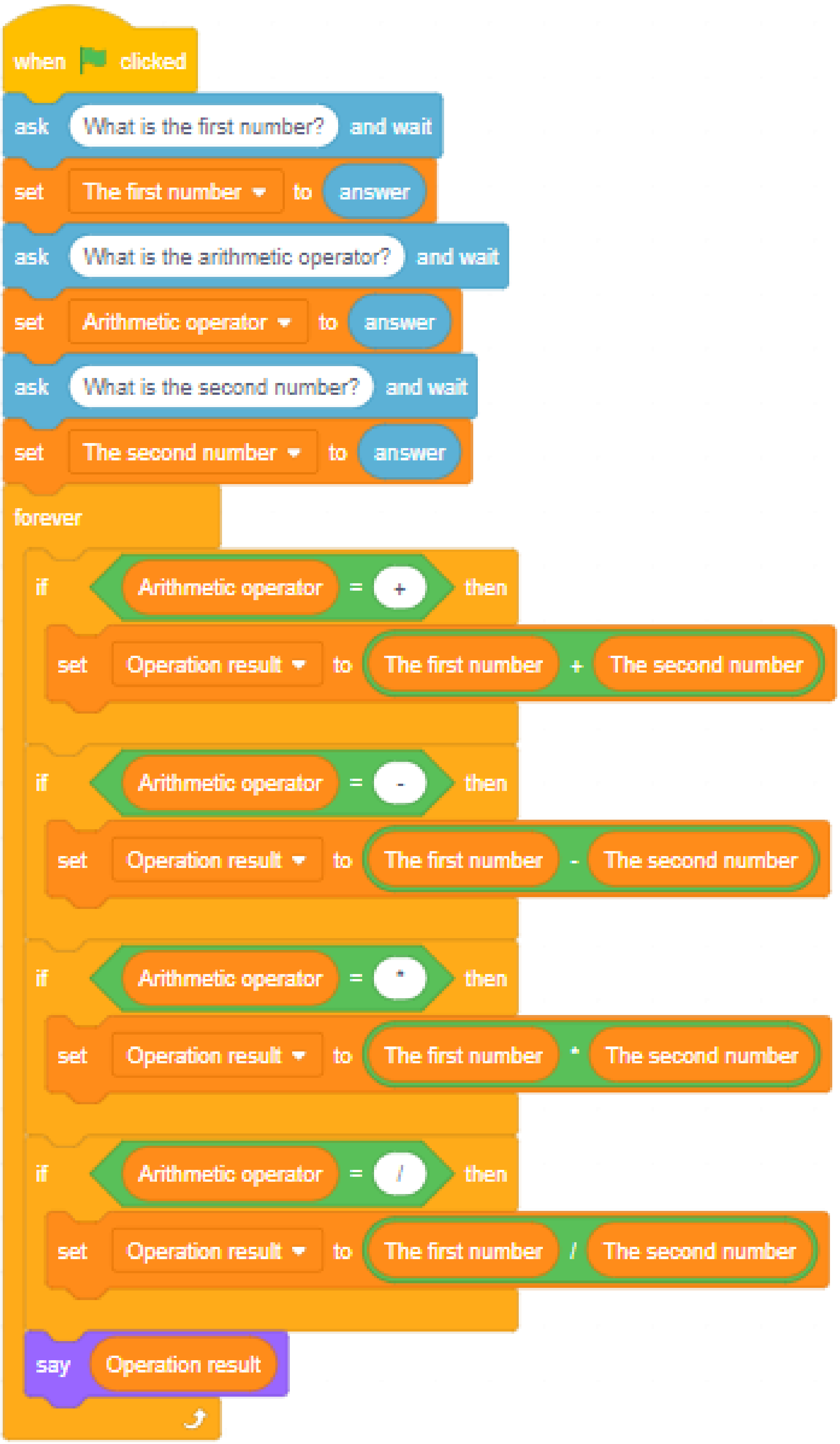


```
say Operation result
```

Figure 3.40

Step 4: Integrate the programs. Repeat the operation and say its result. For the program of the simple calculator, see Figure 3.41.

Write a Sprite Script



```
when 🏴 clicked
ask (What is the first number?) and wait
set [The first number ▾] to (answer)
ask (What is the arithmetic operator?) and wait
set [Arithmetic operator ▾] to (answer)
ask (What is the second number?) and wait
set [The second number ▾] to (answer)
forever
    if <(Arithmetic operator) = (+)> then
        set [Operation result ▾] to ((The first number) + (The second number))
    if <(Arithmetic operator) = (-)> then
        set [Operation result ▾] to ((The first number) - (The second number))
    if <(Arithmetic operator) = (*)> then
        set [Operation result ▾] to ((The first number) * (The second number))
    if <(Arithmetic operator) = (/)> then
        set [Operation result ▾] to ((The first number) / (The second number))
    say (Operation result)
```

Figure 3.41

Running Result of the Program

We can click the green flag button to run the program, enter the formula for calculation, and check the calculator for accuracy. For example, when we enter the formula 5*6, the operation result is shown in Figure 3.42.

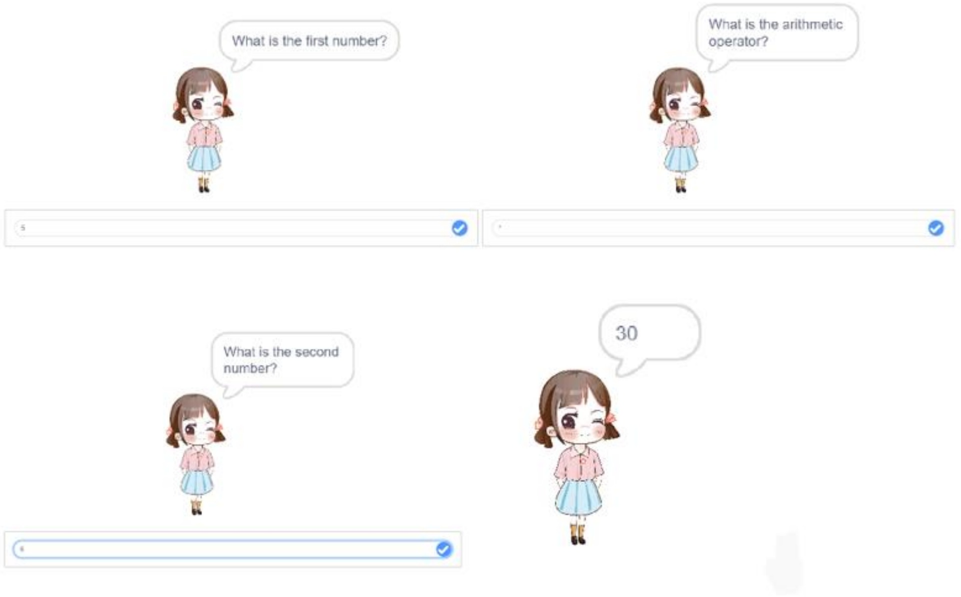

What is the first number?
What is the arithmetic operator?
What is the second number?
30

Figure 3.42

Self-Assessment Room

Content	Result
I've understood arithmetic operators, comparison operators and logical operators.	☆☆☆☆☆
I've completed the tasks of a simple calculator	☆☆☆☆☆

3.4. Data Operation (2)

- Holiday Activity (2)

The holiday activity at the supermarket is warmly welcomed by adults and children in this town. John, when will the second activity start?

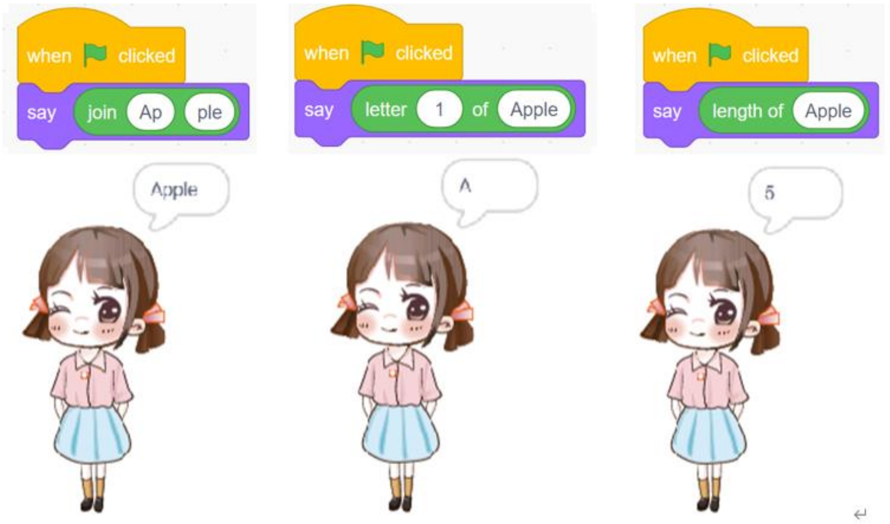
This Friday and the theme will be based on interesting English games. Shuffle the letters of an English word, and guess what the correct word is.

It sounds fun. Can’t wait to play it!

John wants you to think about a question. In the programming, how do we shuffle the letters of an English word to get a new word with shuffled letters? (Characters in the activity are glyph-like units or signs, including letters, numbers, operators and punctuations.)

String Processing

A string is a limited sequence consisting of zero or multiple characters. We look on an English word as a sequence of multiple letters. To shuffle the letters of an English word, we can operate a string. In DobotBlock, characters in a string are stored in order. We can process characters by using the join ○ ○ , letter ○ of ○ , length of ○ and other string processing blocks. For example, we join the string "Apple", take its characters and calculate their numbers, see Figure 3.43.



```
when 🏳 clicked
say join (Ap) (ple)
```

```
when 🏳 clicked
say letter (1) of (Apple)
```

```
when 🏳 clicked
say length of (Apple)
```

Figure 3.43

Let's do it

✧ Process the string "Apple" with a string processing block, and ask Kelly to say the third character and end character of the string.

Research Laboratory

1. Task Release

Create an Interesting English game: Randomly choose a word from the wordlist, shuffle the letters of the word, and guess the correct word according to the output unrecognizable "word". For the game rule, see Figure 3.44.

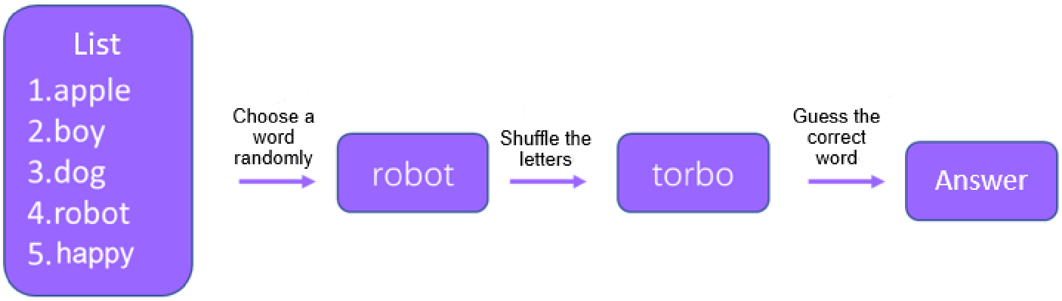


Figure 3.44

2. Task Analysis

According to the game rule, we divide the game into creating a wordlist, choosing a word randomly, shuffling the letters, and guessing the word.

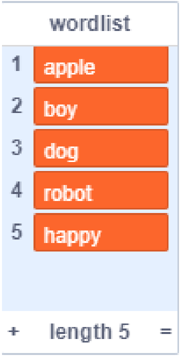
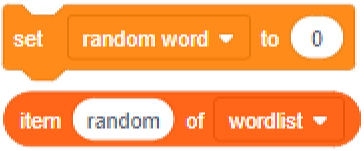
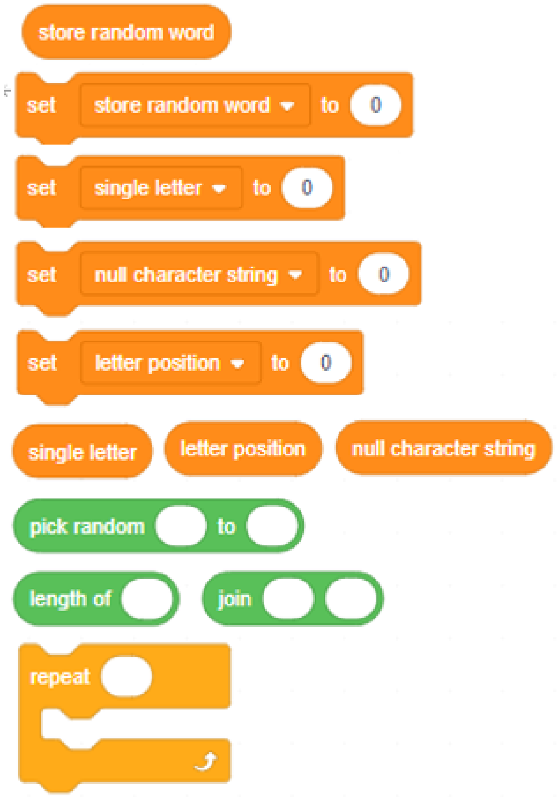
We shuffle the letters through the following steps:

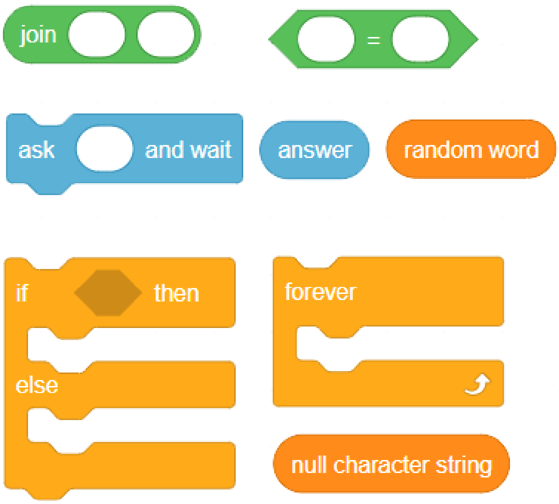
(1) Randomly choose any letter in a word, and put the letter in the first place of the unrecognizable word.

(2) Delete the used letter.

(3) Integrate the randomly selected letters to form an unrecognizable "word".

3. Script Plan

Sprite	Task Description	Block
	1. Create a wordlist.	 wordlist 1 apple 2 boy 3 dog 4 robot 5 happy + length 5 =
	2. Choose a word randomly.	 set random word ▾ to 0 item random of wordlist ▾
	3. Shuffle the letters: Choose any letter from a word, put the letter in a null character string in order according to the random position, and delete the used letter.	 store random word set store random word ▾ to 0 set single letter ▾ to 0 set null character string ▾ to 0 set letter position ▾ to 0 single letter letter position null character string pick random ( ) to ( ) length of ( ) join ( ) ( ) repeat ( )

Sprite	Task Description	Block
	4. Guess the word: Show the shuffled word, ask the answer, and fill in it. If the answer is true the game ends, or you can continue to guess the word.	 join = ask and wait answer random word if then else forever null character string

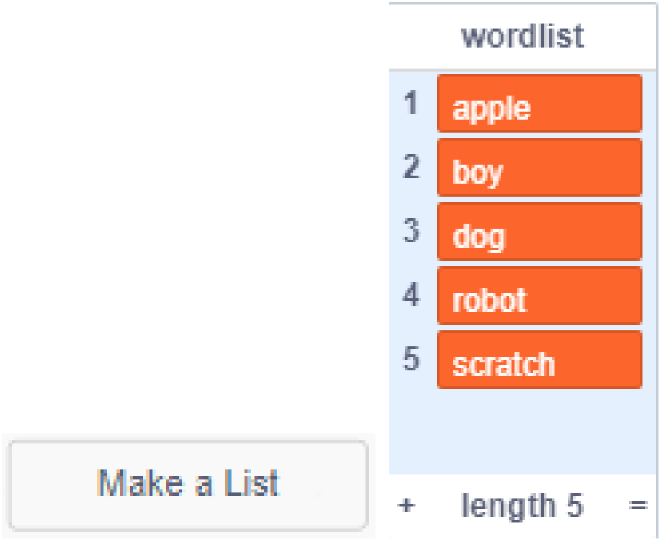
Processing Workshop

Add a Sprite

In the Sprite List, add the sprite "Kelly (1)", and delete the default sprite "Sprite 1".

Create a List

Create a list, name it "wordlist", and add words in the list, see Figure 3.45.



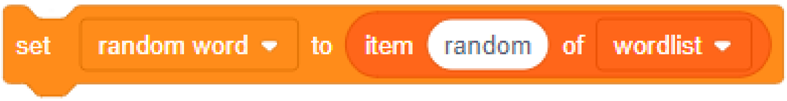
wordlist

1 apple
2 boy
3 dog
4 robot
5 scratch

Figure 3.45

Write a Sprite Script

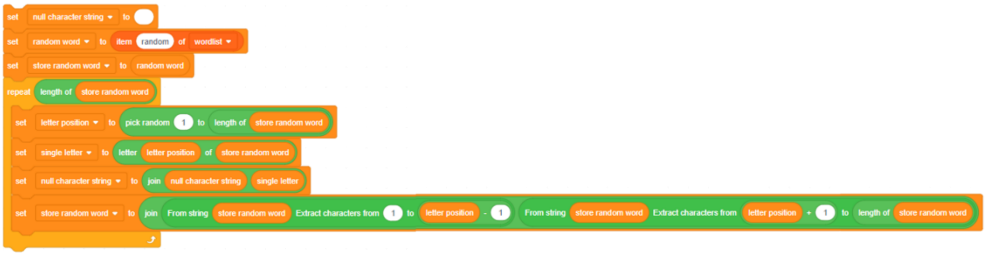
Step 1: Choose a word randomly. Create a variable "random", and store a random word in the wordlist, see Figure 3.46.



```
set random word ▾ to item random of wordlist ▾
```

Figure 3.46

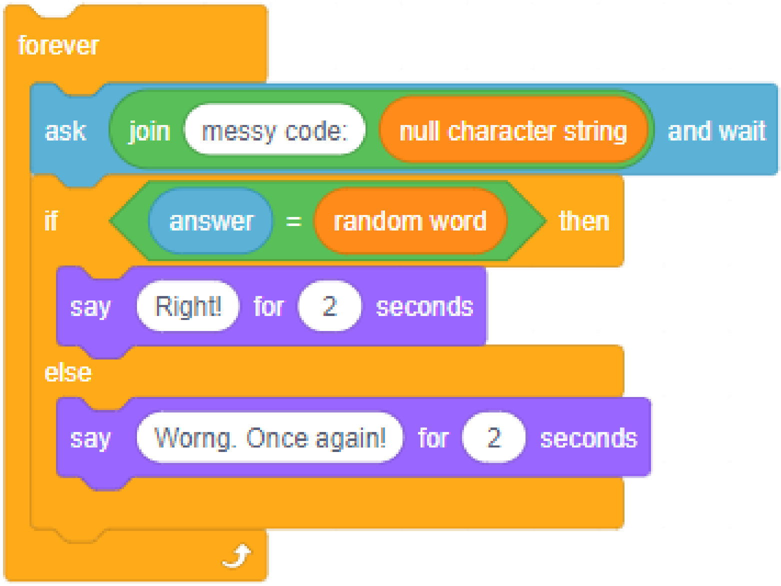
Step 2: Shuffle the letters. Choose any letter from a word, put the letter in a null character string in order according to the random position, and delete the placed letter, see Figure 3.47.



```
set null character string ▾ to ( )
set random word ▾ to item random of wordlist ▾
set store random word ▾ to random word
repeat length of store random word
    set letter position ▾ to pick random 1 to length of store random word
    set single letter ▾ to letter letter position of store random word
    set null character string ▾ to join null character string single letter
    set store random word ▾ to join From string store random word Extract characters from 1 to letter position - 1 From string store random word Extract characters from letter position + 1 to length of store random word
```

Figure 3.47

Step 3: Guess the word. Show the shuffled word, ask the answer, and fill in it. If the answer is true the game ends, or you can continue to guess the word, see Figure 3.48.

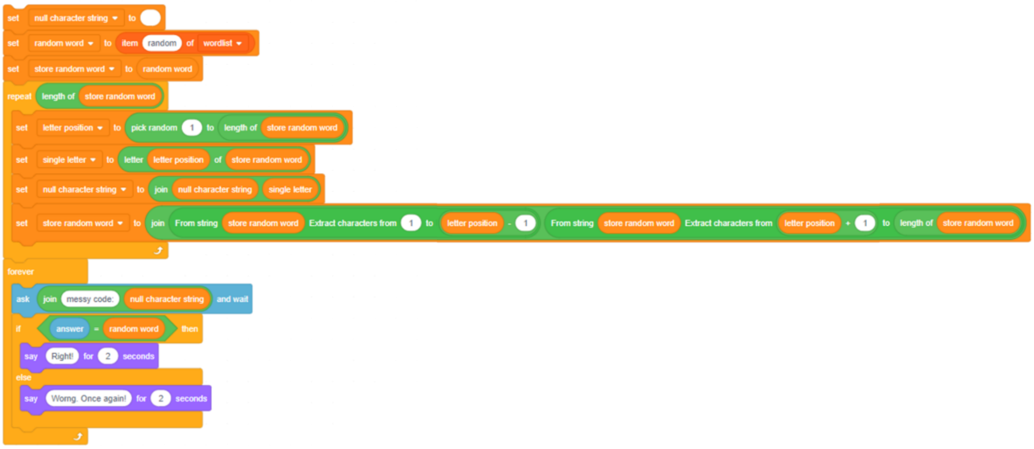


```
forever
    ask join messy code: null character string and wait
    if answer = random word then
        say Right! for 2 seconds
    else
        say Worng. Once again! for 2 seconds
```

Figure 3.48

Write a Sprite Script

Step 4: Integrate the program of the interesting English game, see Figure 3.49.



```
set [null character string ▾] to ( )
set [random word ▾] to (item (random) of [wordlist ▾])
set [store random word ▾] to (random word)
repeat (length of (store random word))
    set [letter position ▾] to (pick random (1) to (length of (store random word)))
    set [single letter ▾] to (letter (letter position) of (store random word))
    set [null character string ▾] to (join (null character string) (single letter))
    set [store random word ▾] to (join (From string (store random word) Extract characters from (1) to ((letter position) - (1))) (From string (store random word) Extract characters from ((letter position) + (1)) to (length of (store random word))))
end
forever
    ask (join (messy code:) (null character string)) and wait
    if <(answer) = (random word)> then
        say (Right!) for (2) seconds
    else
        say (Womg. Once again!) for (2) seconds
    end
end
```

Figure 3.49

Self-Assessment Room

Content	Result
I've known the character and the string	☆☆☆☆☆
I've learned some blocks relating to string processing	☆☆☆☆☆
I've completed the task of an "interesting English game".	☆☆☆☆☆

Chapter 4 Program Structure

-- Expansion of Smart Supermarket

John' supermarket is increasingly popular, especially after he offers the home delivery service. To offer a better shopping experience for his customers, John decides to expand his supermarket into a shopping center. Then he needs to make a lot of decisions during the expansion; for example, what’s to be built first and what’s next? When will the decoration start, and when will his shopping center open again···. All of them can be expressed by program structure. Lets' find the relation between program structure and decisions together in this chapter!

Learning objectives

☼ Learn the basic knowledge of sequence, loop and branch structure.

☼ Program with sequence, loop and branch structure t better use our computer.

Problem-solving skills

Section 1 Sequential Structure and Loop Structure

-- Planning of smart supermarket expansion scheme

Mr. Lee, what are the steps to expand a supermarket into a shopping center?
Everything needs to follow a sequence. Here there are basically the following steps to follow.

Mr. Lee draws a flow chart to help John understand the overall expansion process, see Figure 4.1.

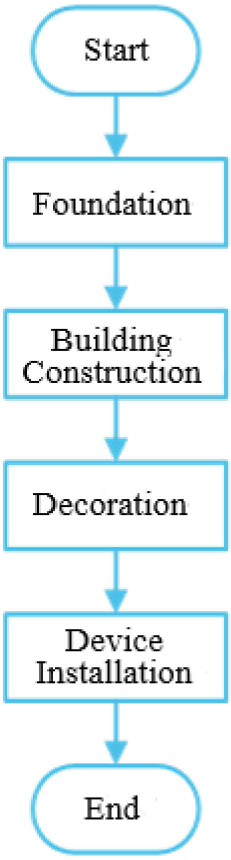


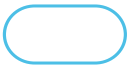
Figure 4.1

1. Flow chart

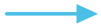
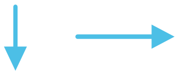
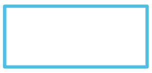
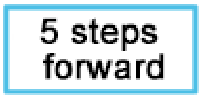
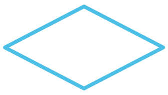
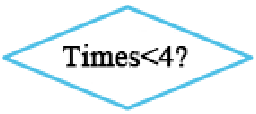
Flow chart is to show the process of a thing in life with specific graphic symbols. Generally, we denote the start or end of a thing with an ellipse, and the specific step with a rectangle, such as foundation treatment, building construction, indoor water and electricity decoration as well as lighting and air conditioning equipment installation in the expansion. Also, we represent the flowline by a straight line with an arrow, to show the direction of the work process. See Table 4.1 for the basic symbols and functions of flow chart.

Table 4.1

Basic Symbols of Flow Chart

Name	Symbols	Function	Examples
Start and end		Start or end the program	

Basic Symbols of Flow Chart

Name	Symbols	Function	Examples
Flowline		Direction of the process	
Processing		Take a step	
Judgment		Judge according to a certain condition	

Remarks: the above are common symbols in flow chart, and we will also learn some others in the later study.

We can also use flow chart during program design to represent the sequence for executing instructions.

2. Sequential structure

Mr. Lee directs John to start from Step 1 and end at the last Step in turn, just like that in Figure 4.2; such a program structure is namely the sequential structure. So in program design, sequential structure is namely the sequence executing instructions from the first one to the last, see Figure 4.2.

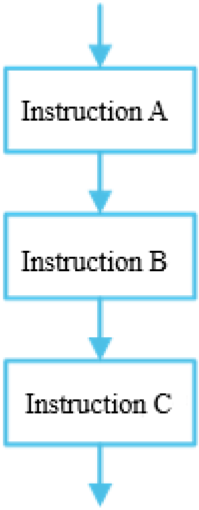


Figure 4.2 Flow Chart of Sequential Structure

Then Mr. Lee and John come to the construction process. Mr. Lee divides the whole process generally into four steps, and each of the steps can be divided into a lot of miniterms. For example, the building construction can further be divided into wall masonry, installation of doors, windows and lintels, concreting for stairs and floorslabs and roofing..., see Figure 4.3.

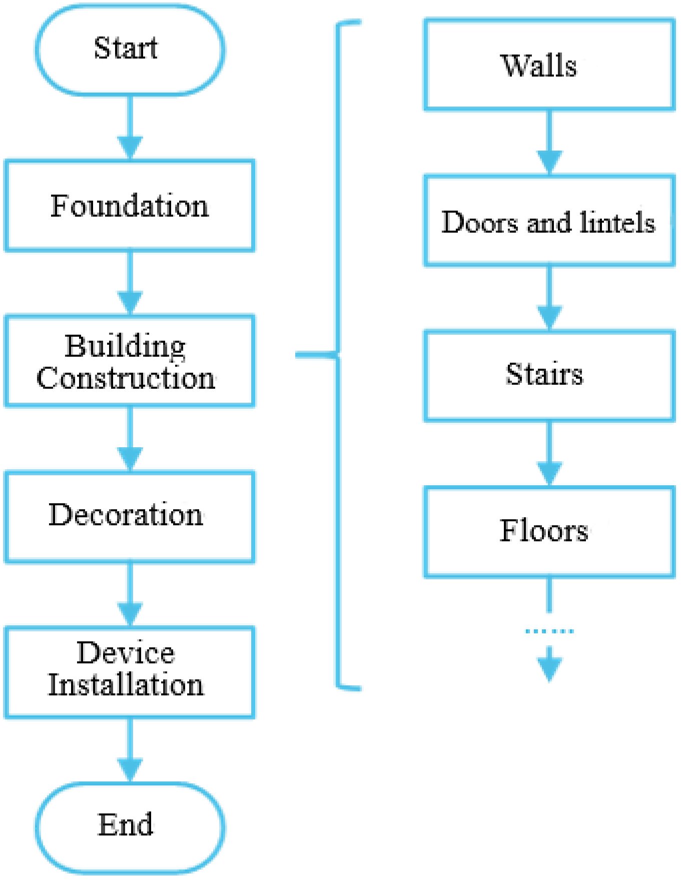


Figure 4.3

To make it easier to understand, Mr. Lee writes a program with DobotBlock for John to move bricks by the robotic arm.

Let's do it

- Run the program "*Graphical Programming and Robots*" \ Chapter 4 \ Section 1 \ Motion of Robots (1).sb3", finish carrying bricks by using blocks. Make robot move to the block position in Area A, open the suction cup to suck the block, and then make the robot move to Area B to place the block.
- See Figure 4.4 for the schematic diagram of the robot and block placement.

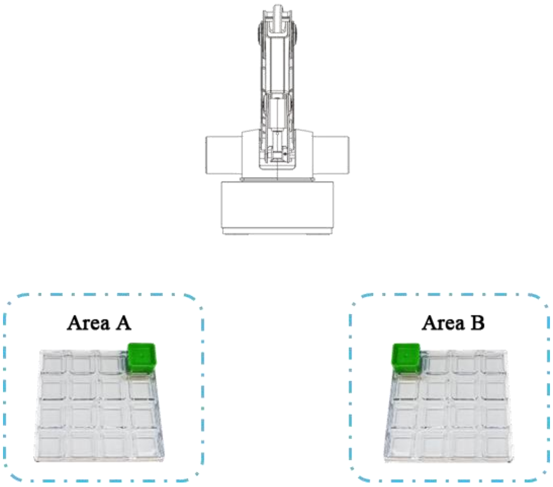


Figure 4.4

3. Loop structure

It is inefficient and takes much time when moving only one brick at a time. Is there any way to carry continuously? Mr. Lee suggests John use the loop structure in program structure, and thus he can repeat a function in the program.

In DobotBlock, blocks that can realize loop are "repeat", "repeat (10)" and "repeat until ()", see Figure 4.5.

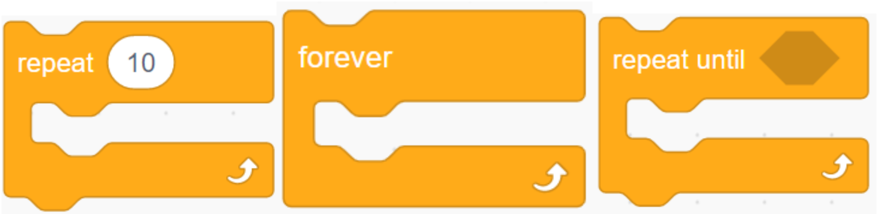


Figure 4.5

(1) "repeat"

It is an instruction to repeat countlessly, namely infinite loop. When using "repeat", the specific block to be repeated is in the groove of the repeat instruction, see Figure 4.6.

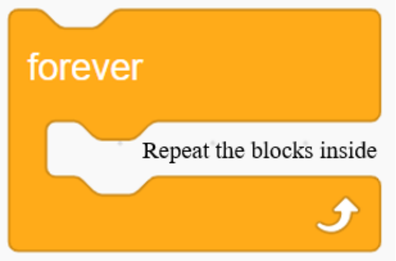


Figure 4.6

Clicking "repeat" during programming, we will keep repeating the block inside until clicking Stop on the Stage, see Figure 4.7 for the flow chart of repeat.

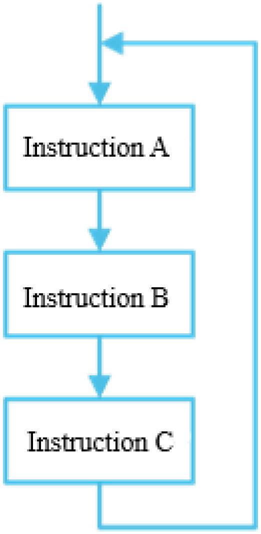


Figure 4.7

Let's do it

✧ Run the program "*Graphical Programming and Robots*" \ Chapter 4\ Section 1 \ Motion of Robots (2).sb3", make robot repeat the motion below: move from the initial position to the first point and then to the second point, and finally return to the initial position.

(2) "repeat (10)"

It is an instruction to repeat for (10) times. The number 10 in the ellipse frame is namely the repeating times 10, and the specific block to be repeated is in the groove of repeat (10), see Figure 4.8.

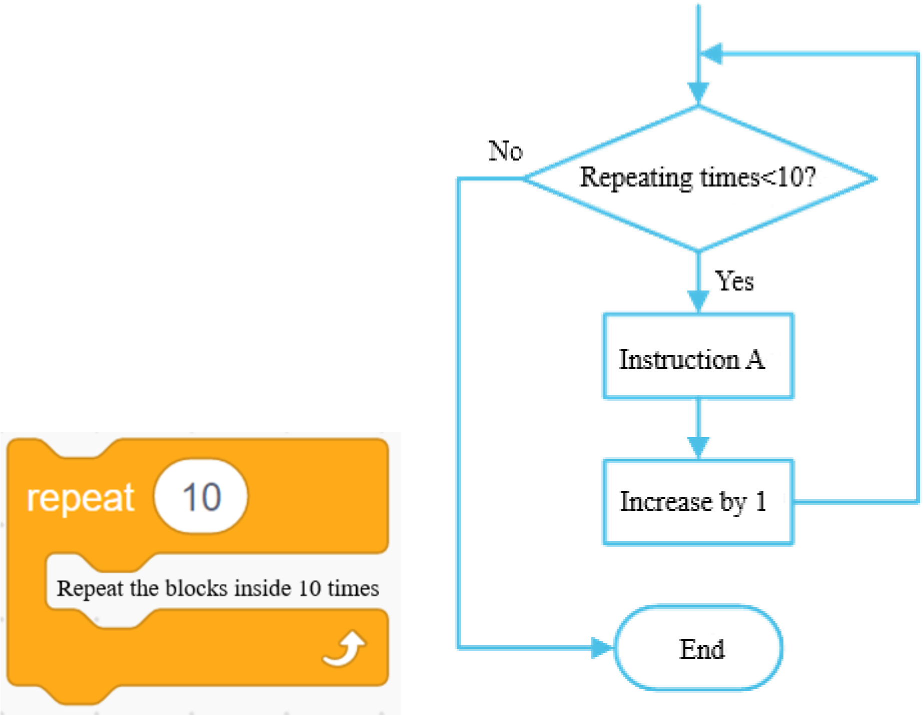


Figure 4.8

Let's do it

✧ Run the program "*Graphical Programming and Robots*" \ Chapter 4 \ Section 1 \ Motion of Robots (3).sb3", make robot repeat the motion below for 5 times: move from the initial position to the first point and then to the second point, and finally return to the initial position.

✧ Use "repeat (10)", but change the repeating times into 5.

(3) "repeat until ()"

When using "repeat until ()", we will repeat the block inside until the repeat condition is met, see Figure 4.9.

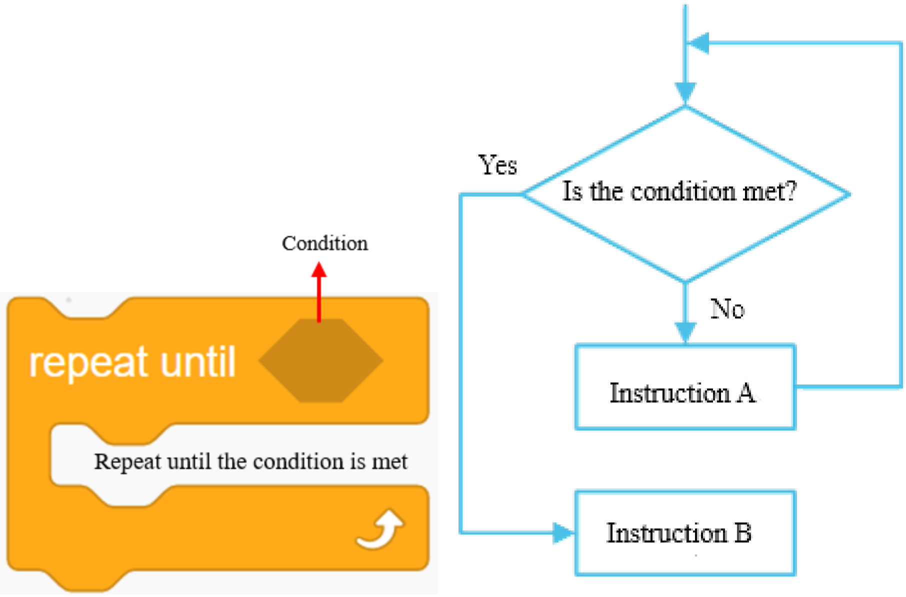
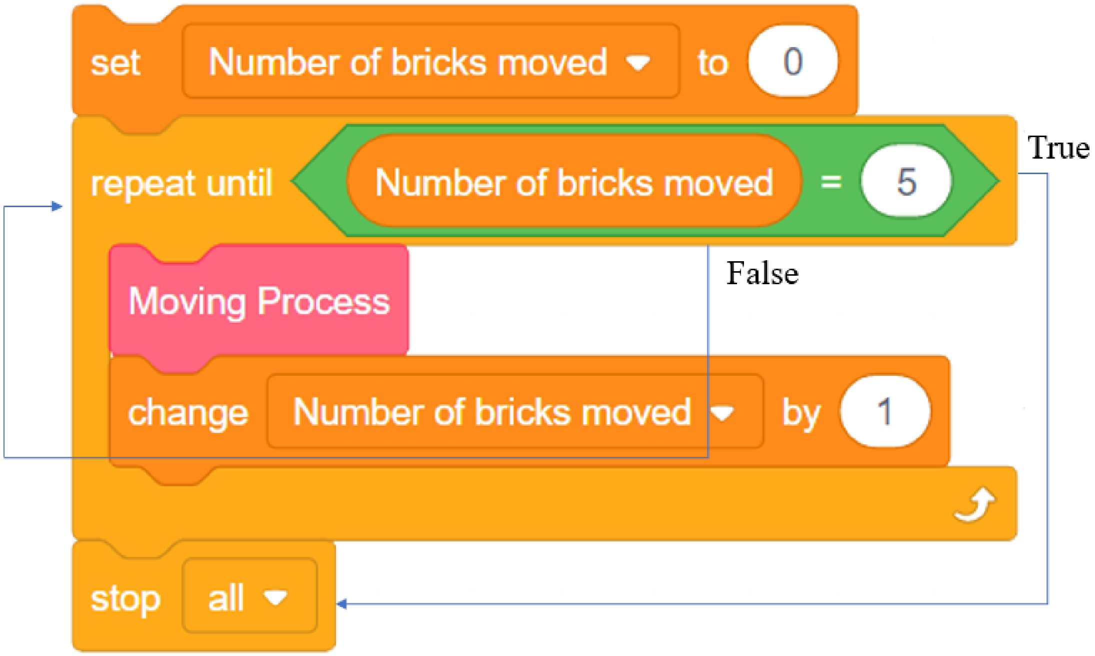


Figure 4.9

When using "repeat until ()", the repeat condition is at the hexagon position of repeat until… (). Both Boolean data and expressions can be the condition. It is to be decided according to the actual condition.

For example, when we need to carry 5 bricks, set the initial value of the number of bricks moved to 0; the number will increase by 1 each time after we carry 1 brick, until the number is 5, then exit the repeat. See Figure 4.10 for the moving process and flow chart.



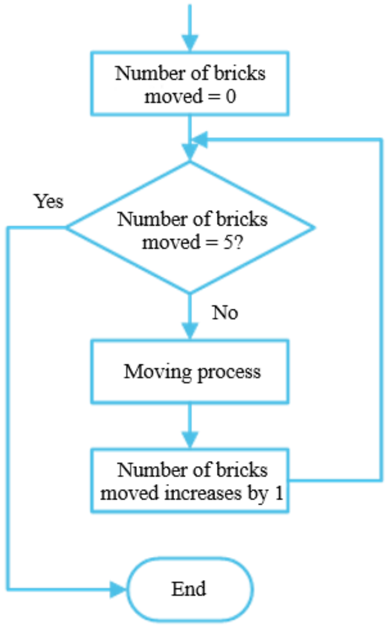

Number of bricks moved = 0
Number of bricks moved = 5?
Yes
No
Moving process
Number of bricks moved increases by 1
End

Figure 4.10

Let's do it

✧ Run the program "*Graphical Programming and Robots*" \ Chapter 4 \ Section 1 \ Moving 5 bricks.sb3", make robot repeat to carry 5 bricks.

Research Office

1. Task release

Carry a list of blocks: carry a list of blocks (4 blocks) in Area A to Area B and place blocks in Area B into a list. See Figure 4.11 for the schematic diagram of the robot and block placement.

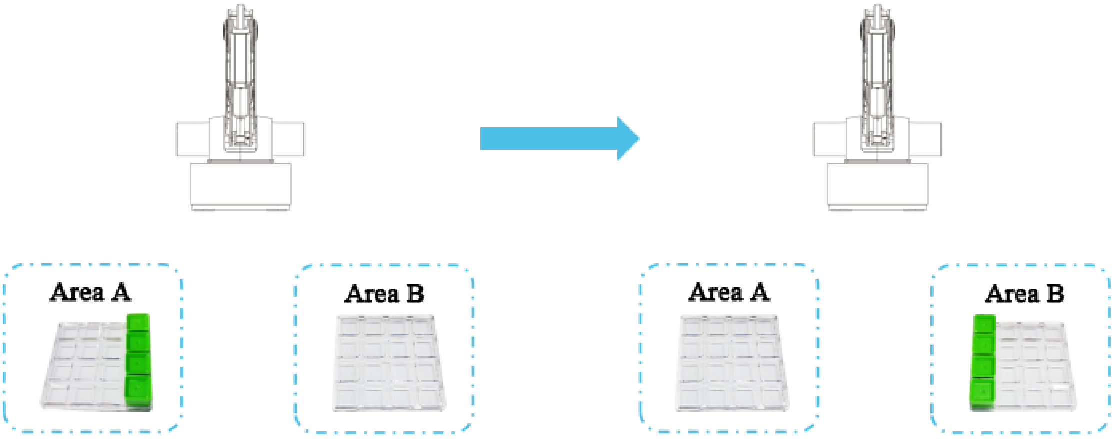


Figure 4.11

2. Task analysis

We carry them from back to front in the task, see Figure 4.12. Similarly, we place the blocks in Area B from back to front.

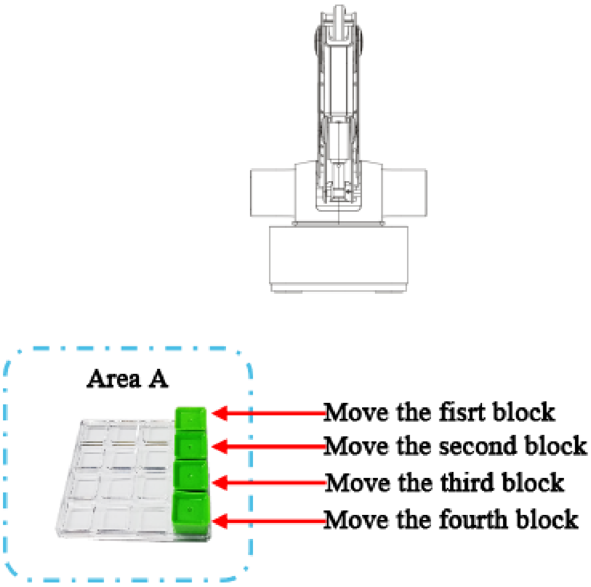
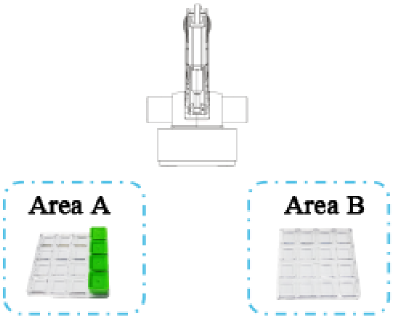
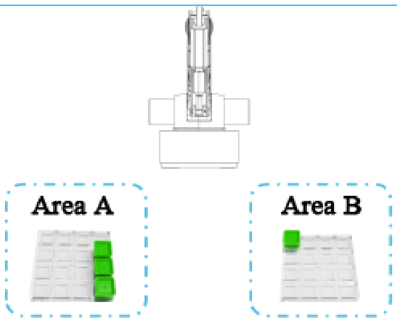
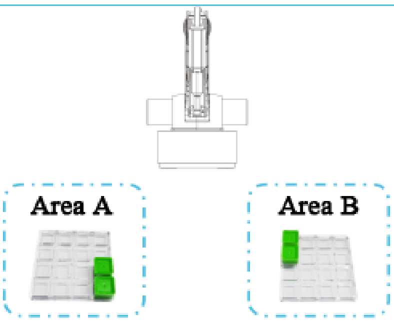
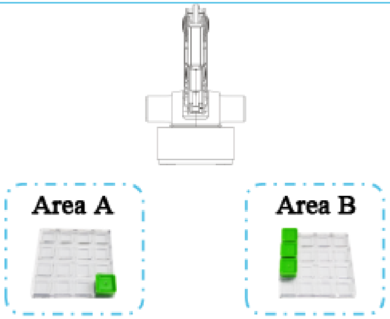
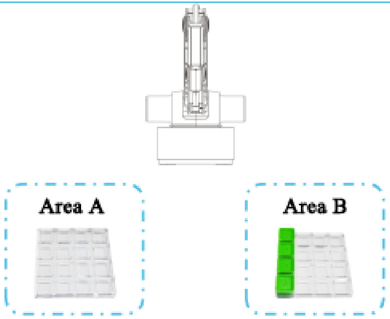


Figure 4.12

When carrying blocks by robot, the number of blocks in Area A and Area B will change, see Table 4.2.

Table 4.2

Before carrying	
After carrying the first block	

After carrying the second block	
After carrying the third block	
After carrying the fourth block	

See Figure 4.13 for the change to the coordination of blocks when carrying them from back to front.

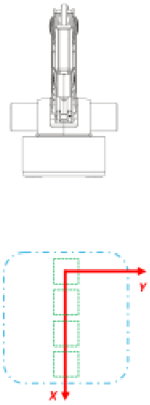


Figure 4.13

All blocks are placed neatly in a list, so the coordination of Y is unchanged, but the coordination of X increases. It is because that both the length and width of the grid for placing blocks are 20 mm in the block box, so the coordination of X will increase by 20 each time after carrying a block.

Read the flow chart of "carrying a list of blocks" according to the task analysis, see Figure 4.14.

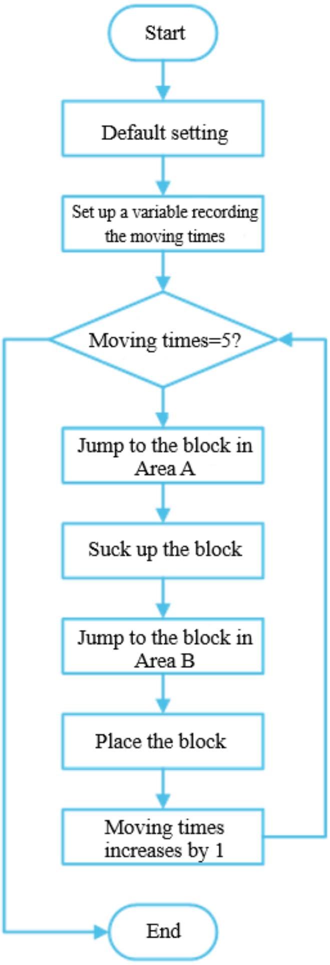
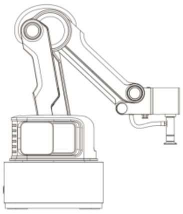
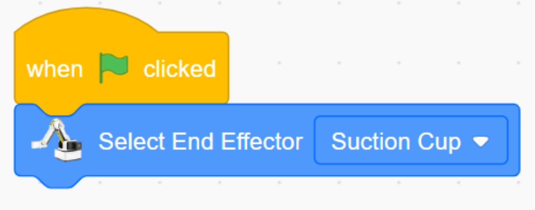
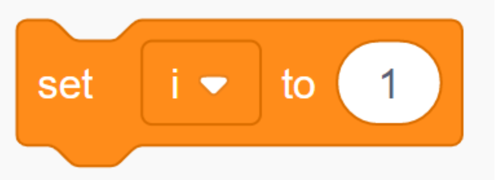
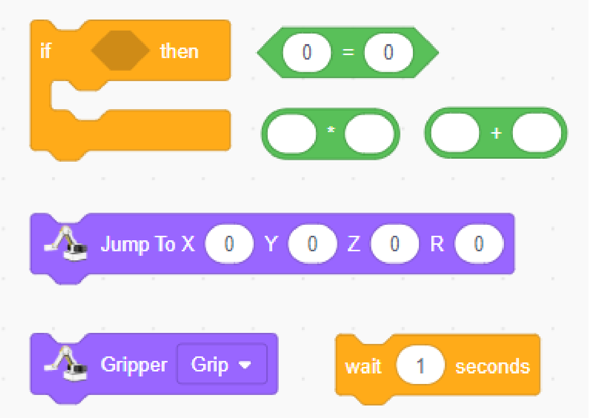

Start
Default setting
Set up a variable recording the moving times
Moving times=5?
Jump to the block in Area A
Suck up the block
Jump to the block in Area B
Place the block
Moving times increases by 1
End

Figure 4.14

3. Script planning

Device	Task description	Block
	1. Initial settings	 when 🏁 clicked Select End Effector Suction Cup ▾

Device	Task description	Block
	2. Value of initial variable "i"	 set i to 1
	3. The robot carries a list of blocks in Area A to Area B	 if then 0 = 0 * + Jump To X 0 Y 0 Z 0 R 0 Gripper Grip wait 1 seconds

Processing zone

Add and connect device

Step 1. Add the device "Magician Lite" in device list at the lower left of DobotBlock interface and delete the default device "Magician".

Step 2. Connect robot and computer with a Type-C cable, and click Connect key in the control area of DobotBlock to connect device.

Create variable

Click "Variable" module and "Set up a variable", fill new variable "i" in the pop-up interface. We can use variable "i" to record the times for carrying blocks, see Figure 4.15.

Figure 4.15

Write device script

Step 1. Initial setting. Suck the block with a suction cup when carrying it with robot, see Figure 4.16.

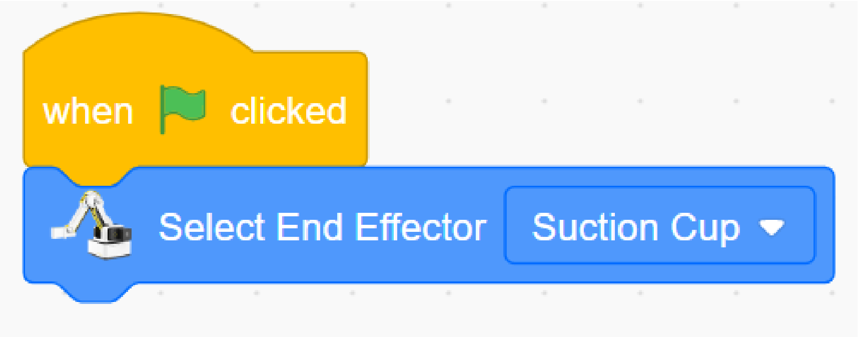


Figure 4.16

Step 2. The initial value of variable "i" is 1, see Figure 4.17.

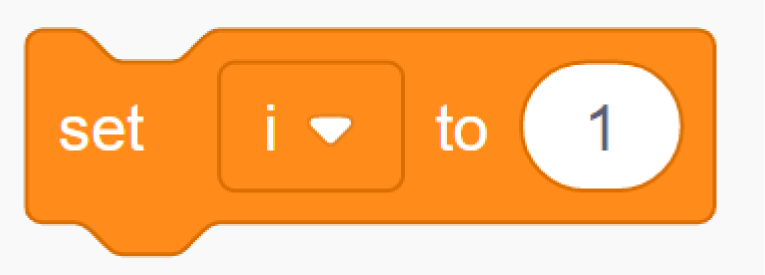


Figure 4.17

Step 3. Set up judgment conditions. When we use robot to carry a list of blocks (4 blocks), when i=1, the robot will carry the first block; when i=2, it will carry the second block, when i=3, it will carry the third block, and when i=4, it will carry the fourth block. So, when i=5, it will stop repeat. When using "repeat until ()", we need to set the program of setting judgment conditions, see Figure 4.18.

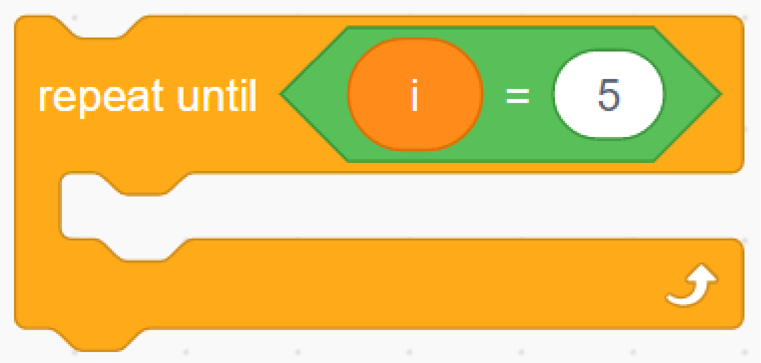
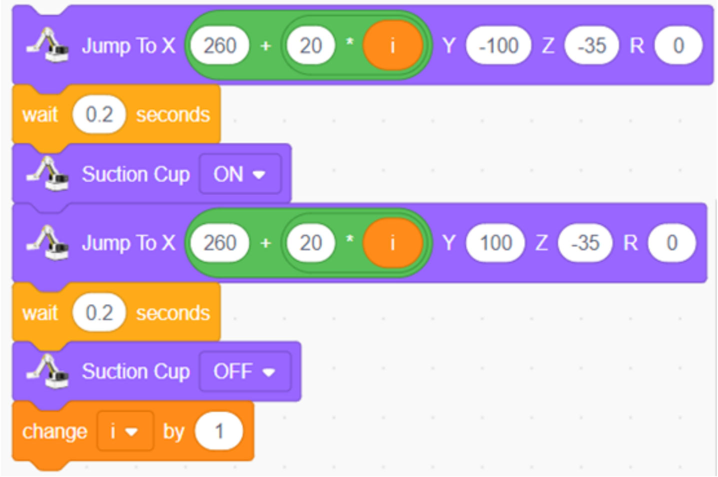


Figure 4.18

Write device script

Step 4. The robot will carry blocks one by one from back to front. It will move to Area A to suck the block, and then move to Area B to place it. The value of variable "i" will increase by 1 each time after carrying a block, see Figure 4.19 for the programming.

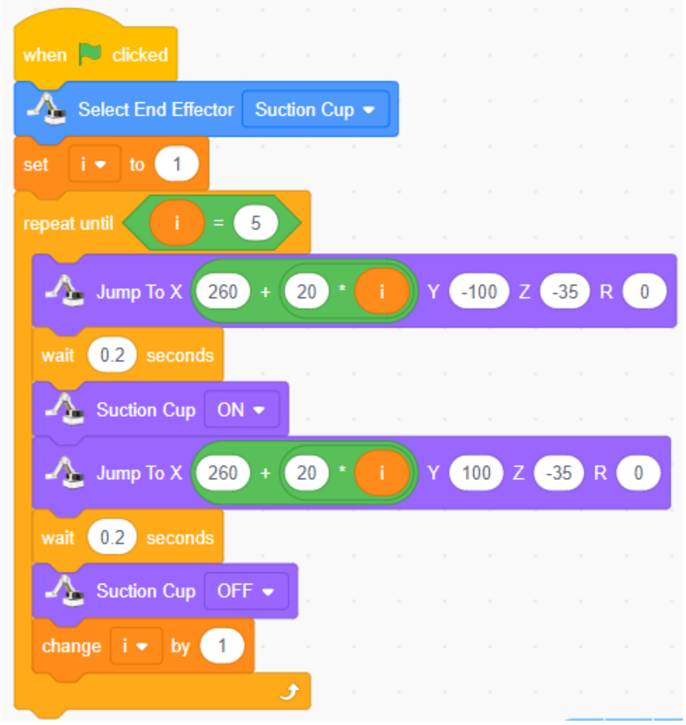


```
Jump To X (260 + (20 * i)) Y -100 Z -35 R 0
wait 0.2 seconds
Suction Cup ON
Jump To X (260 + (20 * i)) Y 100 Z -35 R 0
wait 0.2 seconds
Suction Cup OFF
change i by 1
```

Figure 4.19

Lets' think about it: what does 20 mean in the expression 260+20*i for coordination of X?

Step 5. Integrate program. The robot carries a list of blocks from Area A to Area B, see Figure 4.20 for the programming.



```
when ⚑ clicked
Select End Effector Suction Cup
set i to 1
repeat until (i = 5)
    Jump To X (260 + (20 * i)) Y -100 Z -35 R 0
    wait 0.2 seconds
    Suction Cup ON
    Jump To X (260 + (20 * i)) Y 100 Z -35 R 0
    wait 0.2 seconds
    Suction Cup OFF
    change i by 1
```

Figure 4.20

Innovation park

Carrying two lists of blocks: carry two lists of blocks (4/list) in Area A to Area B and also place the blocks in Area B into two lists. See Figure 4.21 for the schematic diagram of robot and block placement.

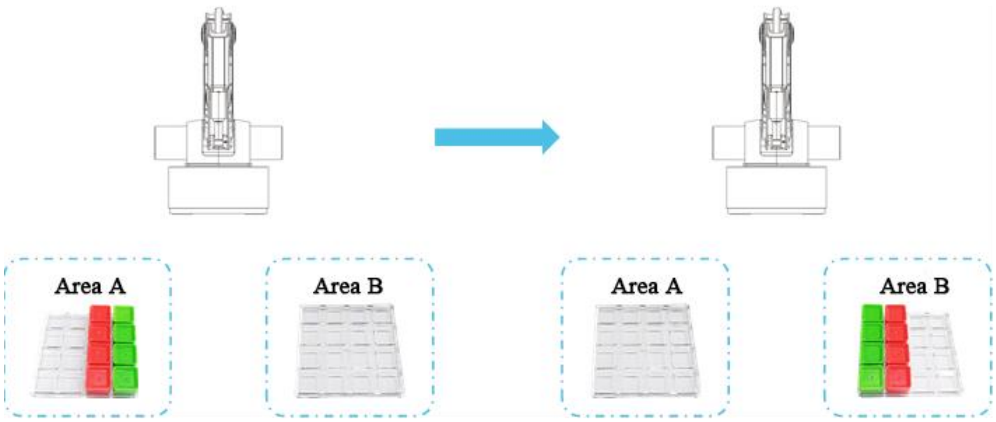


Figure 4.21

Self-Assessment Room

Content	Result
I've learned what's the sequential structure and process.	☆☆☆☆☆
I've learned what's the loop structure and process.	☆☆☆☆☆
I’ve finished "carrying a list of block".	☆☆☆☆☆

Section 2 Branch Structure (1)

-- Solution for tile classification

Mr. Lee, the expansion goes very well. I bought floor tiles of 80 cm X 80 cm and wall tiles of 30 cm X 60 cm for indoor decoration. Look, here are them, but how to distinguish them easily?

We need to judge first and then choose corresponding steps based on our judgment. Here we can solve it by using the branch structure in program design.

1. Single branch structure

There is only one judgment condition in single branch structure, and only when the condition is met, the program will execute corresponding instructions. See Figure 4.22 for the flow chart of single branch structure.

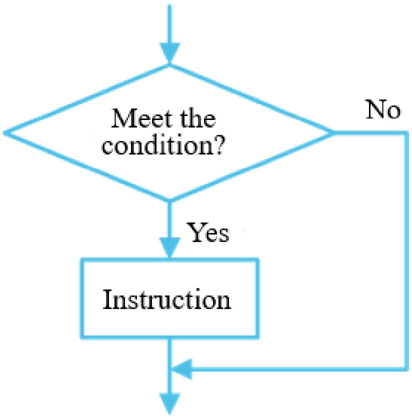


Figure 4.22

Taking the problem encountered by John as the example, if John knows whether the side length of the tile is 80 cm, then he can judge if it is a floor tile, see Figure 4.23.

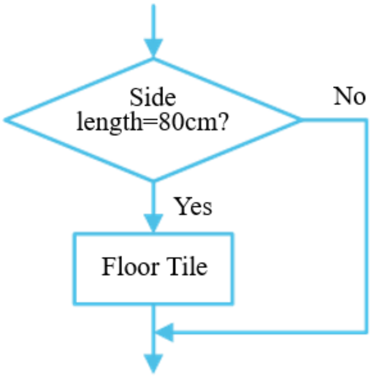


Figure 4.23

In DobotBlock, "if..., then..." is another block corresponding to the single branch structure, see Figure 4.24.

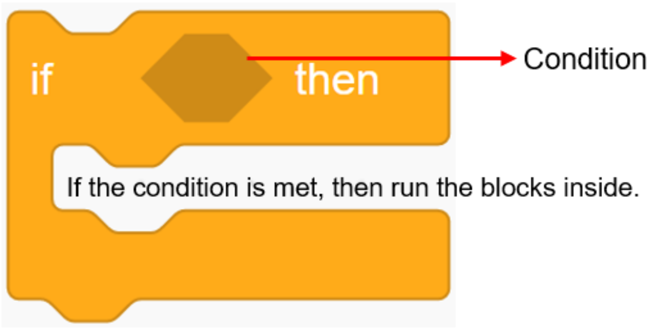
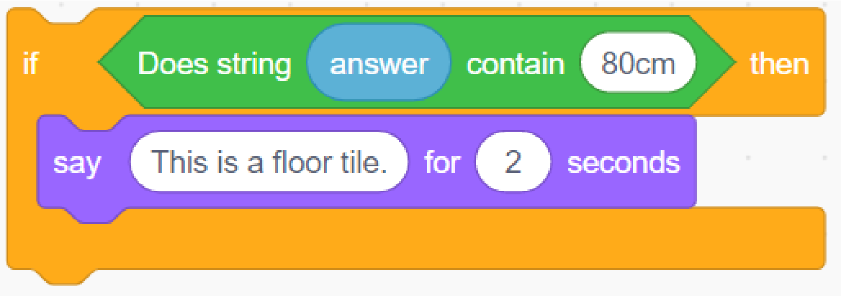


Figure 4.24

In "if..., then..." block, the judgment condition is at the hexagon position; it can include an or several instruction (s). If the condition is met, then run all blocks inside. While if not, then run blocks behind "if..., then...".

For example, when judging if it is a floor tile, if the side length is 80 cm, then the program can judge that it is the floor tile; see Figure 4.25 for some programs for judgment of floor tiles.



```
if <Does string (answer) contain (80cm)> then
    say (This is a floor tile.) for (2) seconds
```

Figure 4.25

Let's do it

✧ Run "*Graphical Programming and Robots*" \ Chapter 4 \ Section 2\ Judgment of Floor Tiles.sb3", enter the side length of a floor tile to check if it is floor tile.

2. Dual branch structure

Is there any way to distinguish the floor tile from wall tile by a judgment?
Sure, have a try about the dual branch structure.

There is only a judgment condition in the dual branch structure to come into two different branches. We can choose any of the branches based on judgment and execute corresponding instructions. As shown in Figure 4.26, if the condition is met, then execute instruction A; while if not, then execute instruction B.

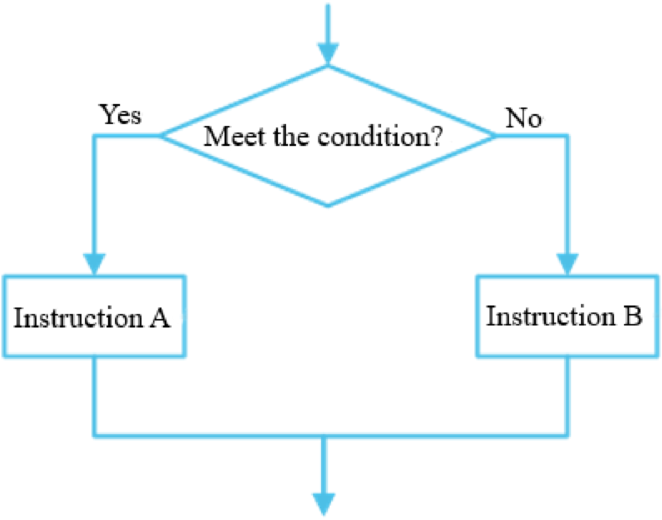


Figure 4.26

In DobotBlock, "if..., then...else..." is another block corresponding to the dual branch structure, see Figure 4.27 for its programming format.

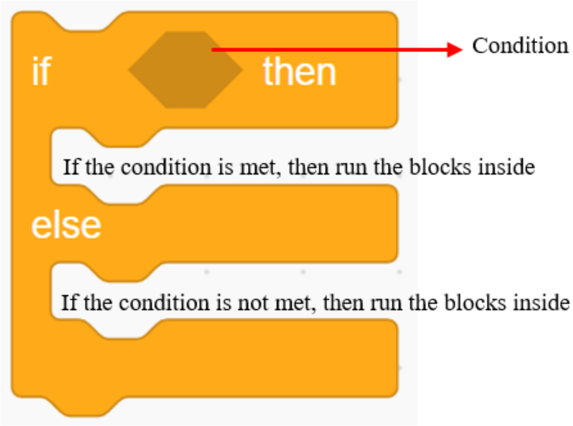
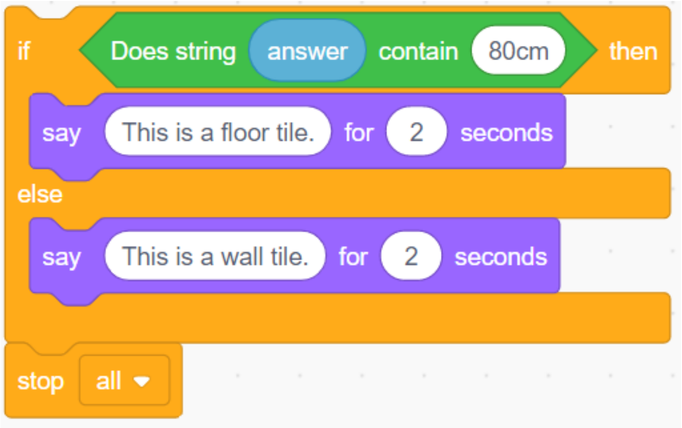

if
then
Condition
If the condition is met, then run the blocks inside
else
If the condition is not met, then run the blocks inside

Figure 4.27

In "if..., then...else..." block, the judgment condition is at the hexagon position. If the condition is met, then run all blocks inside the first groove. While if not, run blocks in the groove below the "else".

Mr. Lee suggests John to solve his new question in the way in Figure 4.28.



```
if <Does string (answer) contain (80cm)> then
    say (This is a floor tile.) for (2) seconds
else
    say (This is a wall tile.) for (2) seconds
stop [all ▾]
```

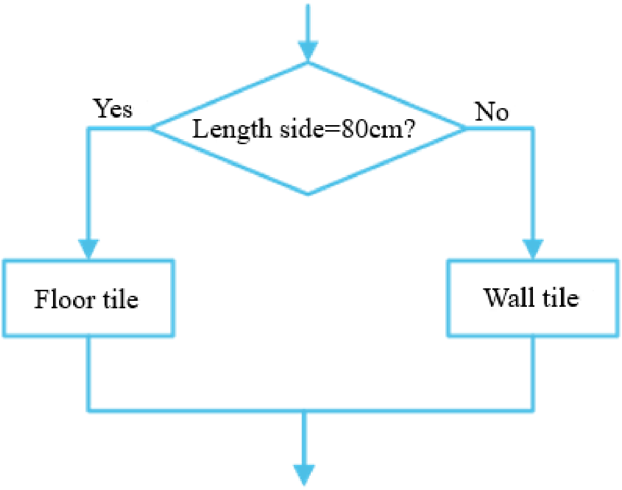


Figure 4.28

Research Office

1. Task release

Ticket checking by ticket inspectors (I): the ticket inspector will ask for height before the student takes the bus. If the height is < or = 1.2 m, then the student can take the bus for free. If > 1.2 m, the ticket inspector will ask the student to buy a ticket.

2. Task analysis

Fill out the flow chart "Ticket Checking by Ticket Inspectors (I)" in Figure 4.29 according to task analysis above.

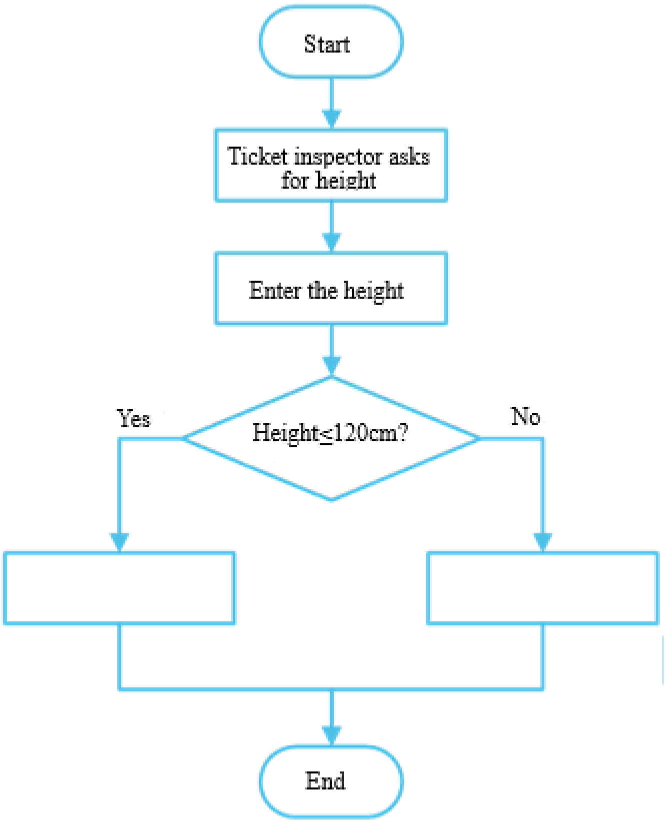
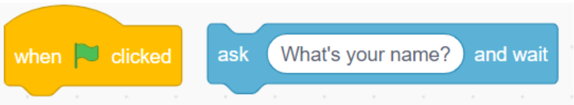
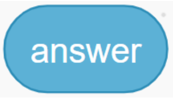
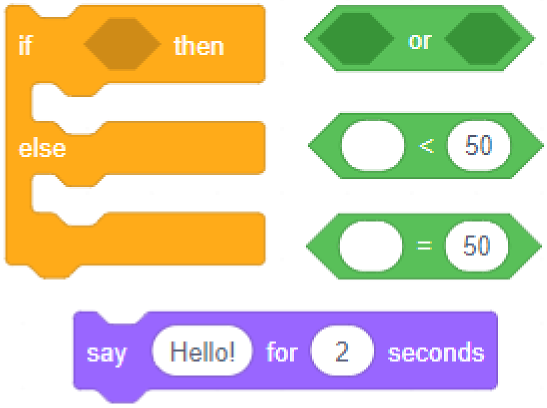


Figure 4.29

3. Script planning

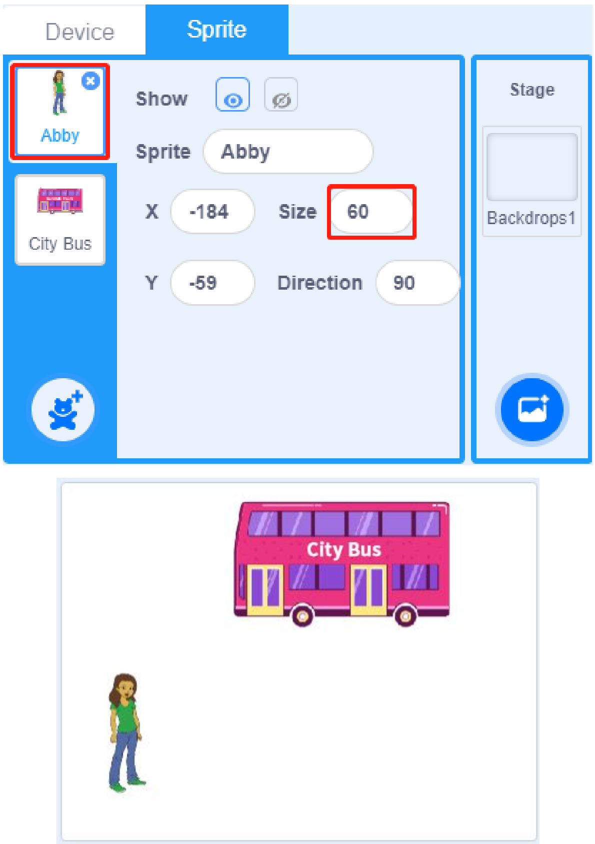
Sprite	Task description	Block
	1. Ticket inspector asks for height.	
	2. Enter the height and set the entered value as the answer.	
	3. Judge if the student needs to buy a ticket according to the answer.	 if then else or < 50 = 50 say Hello! for 2 seconds

Processing zone

Add sprites

Step 1. Add sprite "City Bus". Click Sprite, the key Add Sprite on the interface, and then click sprite "City Bus" on the pop-up interface, delete the default sprite "Sprite 1".

Step 2. Add sprite "Abby" and set its size as 60. Also, adjust the position of "Abby" on Stage, see Figure 4.30.

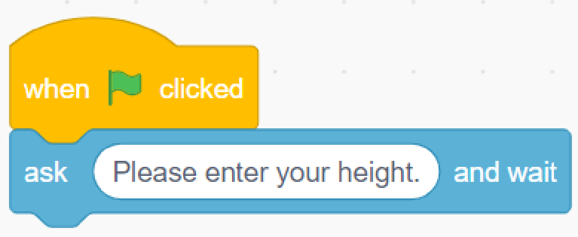


Show
Sprite: Abby
X: -184 Size: 60
Y: -59 Direction: 90

Figure 4.30

Write sprite script

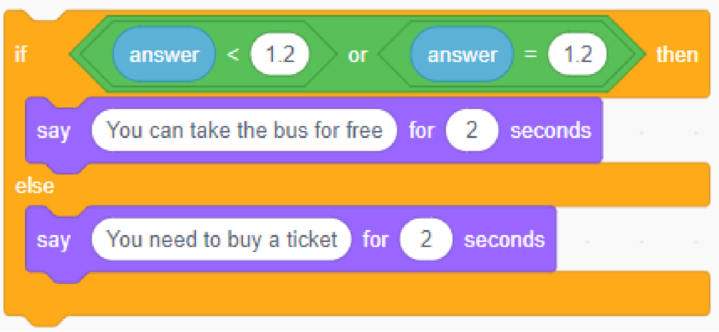
Step 1. Ticket inspector asks for height. Program to ask for student's height in the Coding Area of sprite "Abby", see Figure 4.31.



```
when ⚑ clicked
ask (Please enter your height.) and wait
```

Figure 4.31

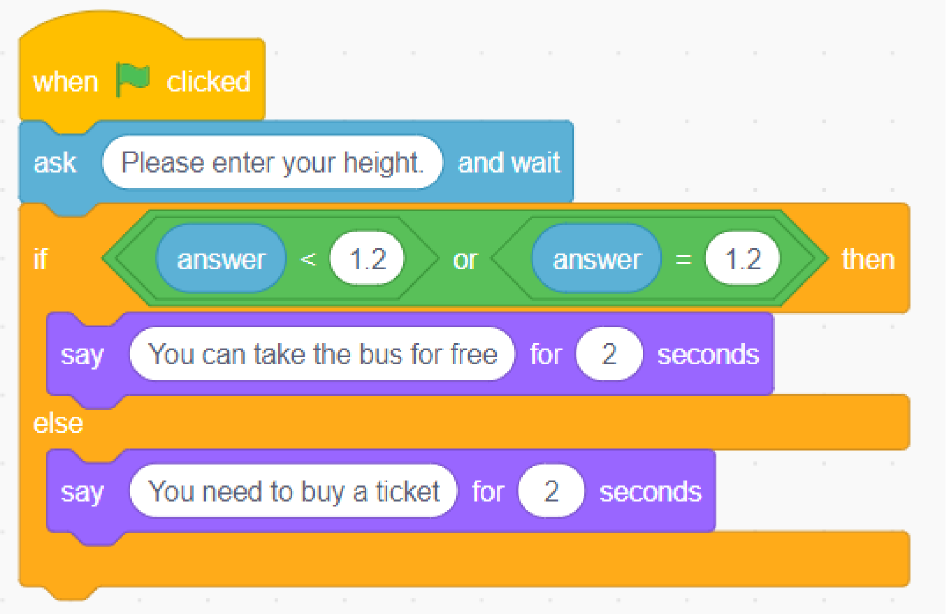
Step 2. Judge if the student needs to buy a ticket based on the answer, see Figure 4.32.



```
if <<(answer) < (1.2)> or <(answer) = (1.2)>> then
    say (You can take the bus for free) for (2) seconds
else
    say (You need to buy a ticket) for (2) seconds
```

Figure 4.32

Step 3. Integrate program. See Figure 4.33 for the program for ticket checking by ticket inspectors (I).



```
when ⚑ clicked
ask (Please enter your height.) and wait
if <<(answer) < (1.2)> or <(answer) = (1.2)>> then
    say (You can take the bus for free) for (2) seconds
else
    say (You need to buy a ticket) for (2) seconds
```

Figure 4.33

Innovation park

Modify the program "Ticket checking by ticket inspectors (I)", try to substitute "if..., then...else..." with "if..., then…" to finish the task "Ticket checking by ticket inspectors (I)".

Self-Assessment Room

Content	Result
Now I know the branch structure.	☆☆☆☆☆
I know the single and dual branch structure.	☆☆☆☆☆
I finished the task "Ticket checking by ticket inspectors (I)".	☆☆☆☆☆

Section 3 Branch Structure (2)

-- Scheme for final acceptance

The shopping center is almost completed, but how to know it is completed?

If the acceptance condition is met at each link in contrast to the construction sequence, then the project is completed!

1. Nested branch structure

In program design, nested branch structure is a branch of the branch structure; it includes a or many branch structure (s). See Figure 4.34 for the flow chart of the nested branch structure. If meeting the condition 1, then judge the condition 2 is met or not, or execute instruction 1. If meeting the condition 2, then keep on judging the next, or execute instruction 2, and so on. If meeting the judgment condition, then keep on judging the next, or execute the corresponding instructions.

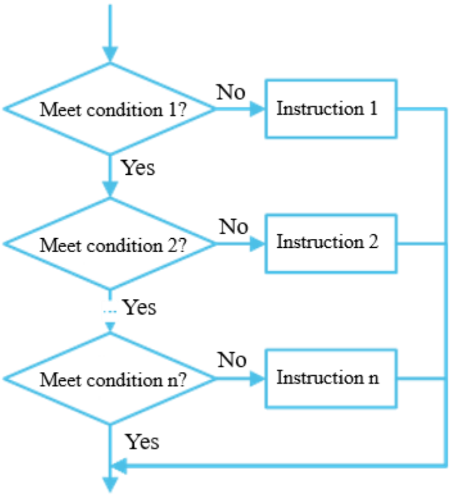


Figure 4.34

Now John goes to the construction site for acceptance of its shopping center. According to the construction process, the overall expansion can be thought ended only when all of the conditions for foundation, building construction, decoration and device installation are met, see Figure 4.35.

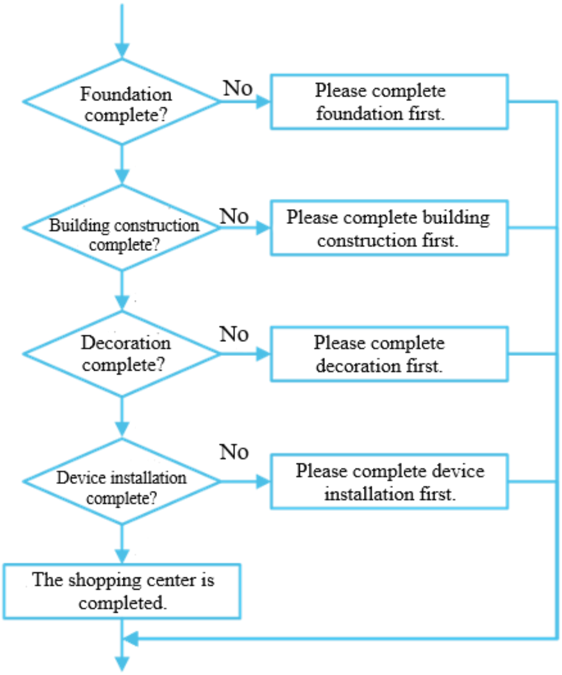

Foundation complete?
No
Please complete foundation first.
Building construction complete?
No
Please complete building construction first.
Decoration complete?
No
Please complete decoration first.
Device installation complete?
No
Please complete device installation first.
The shopping center is completed.

Figure 4.35

2. Building of nested branch structure

In DobotBlock, we can nest "if..., then…" with "if..., then...else..." mutually to come into a multi-branch structure with at least two paths. See Figure 4.36 for one of the way to build nested branch structure.

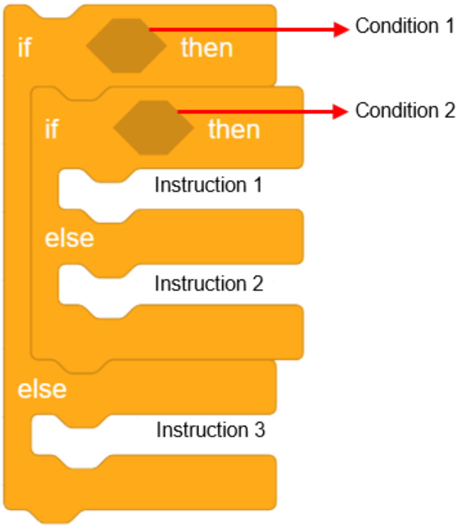
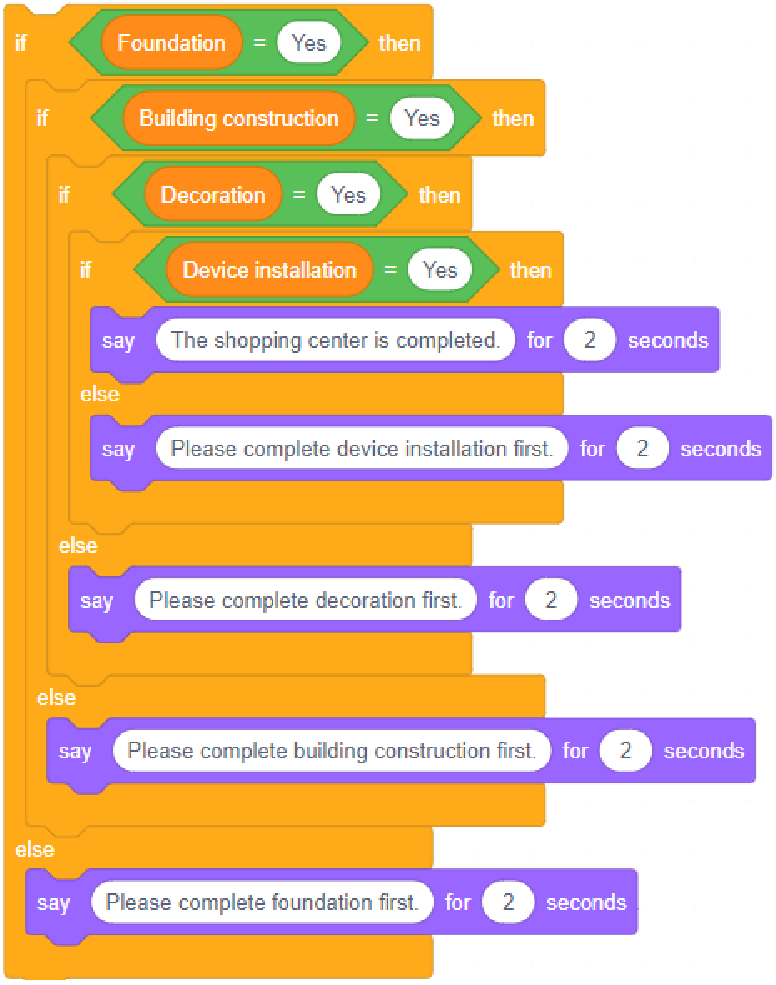


Figure 4.36

See Figure 4.37 for part of the processes accepted by John on construction site.



```
if <(Foundation) = Yes> then
    if <(Building construction) = Yes> then
        if <(Decoration) = Yes> then
            if <(Device installation) = Yes> then
                say (The shopping center is completed.) for (2) seconds
            else
                say (Please complete device installation first.) for (2) seconds
        else
            say (Please complete decoration first.) for (2) seconds
    else
        say (Please complete building construction first.) for (2) seconds
else
    say (Please complete foundation first.) for (2) seconds
```

Figure 4.37

Let's think about it

✧ Is there any other way for block connection in nested branch structure?

Let's do it

✧ Program performing acceptance on construction site by other ways of block connection.

Research Office

1. Task release

Ticket checking by ticket inspectors (2): the ticket inspector will ask for height before the student takes the bus. If the height is < or = 1.2m, then the student can take the bus for free. If > 1.2m and < or = 1.5m, then ticket inspector will ask the student to buy a half-price ticket. If > 1.5m, then ticket inspector will ask the student to buy a full-price ticket.

2. Task analysis

The video about ticket checking by ticket inspectors shows asking for height, entering the height, judging the height and deciding to buy ticket or not. Fill out the flow chart of "Ticket checking by ticket inspectors (2)", see Figure 4.38.

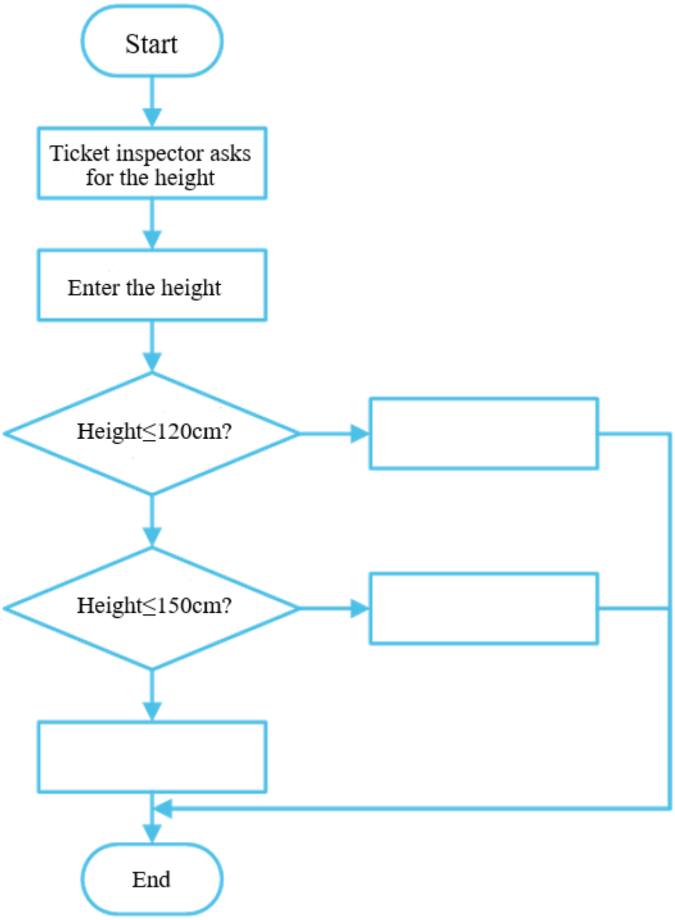
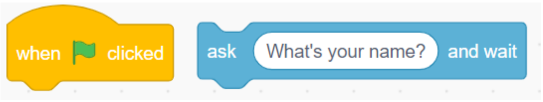
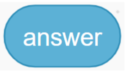
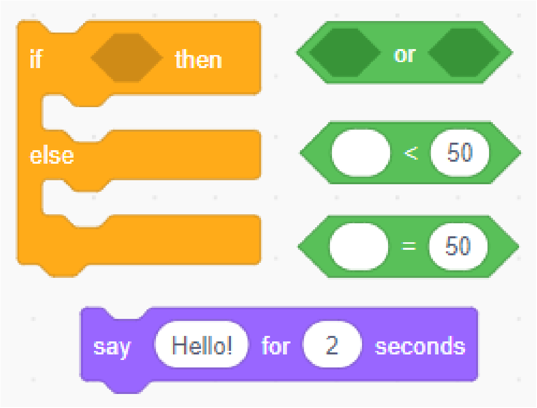


Figure 4.38

3. Script planning

Sprite	Task description	Block
	1. Ticket inspector asks for height.	
	2. Enter the height and set the entered value as the answer.	
	3. Judge if the student needs to buy a ticket based on the answer.	 if then else or < 50 = 50 say Hello! for 2 seconds

Processing zone

Add sprites

(1) Add sprite "City Bus", delete "Sprite 1” in Sprite.

(2) Add and set sprite "Abby". Add sprite "Abby", set its size as 60; adjust the sprite position on Stage (refer to "Ticket checking by ticket inspectors (1)" taught on the second class).

Write sprite script

Step 1. Ticket inspector asks for height. Program to ask for student's height in the Coding Area of sprite "Abby", see Figure 4.39.

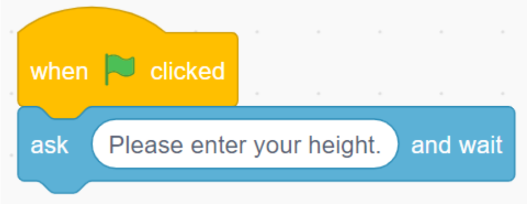
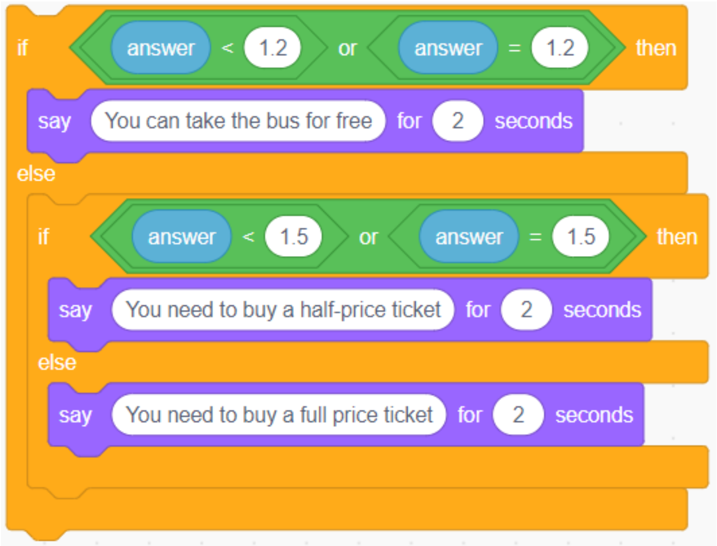


Figure 4.39

Write sprite script

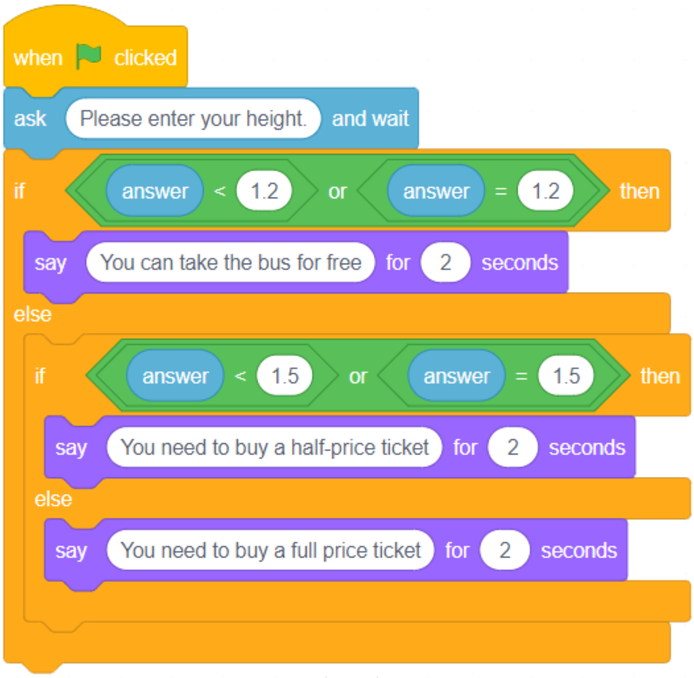
Step 2. Judge if the student needs to buy a ticket based on the answer, see Figure 4.40.



```
if <<(answer) < (1.2)> or <(answer) = (1.2)>> then
    say (You can take the bus for free) for (2) seconds
else
    if <<(answer) < (1.5)> or <(answer) = (1.5)>> then
        say (You need to buy a half-price ticket) for (2) seconds
    else
        say (You need to buy a full price ticket) for (2) seconds
    end
end
```

Figure 4.40

Step 3. Integrate program. See Figure 4.41 for the program for ticket checking by ticket inspectors (2).



```
when ⚑ clicked
ask (Please enter your height.) and wait
if <<(answer) < (1.2)> or <(answer) = (1.2)>> then
    say (You can take the bus for free) for (2) seconds
else
    if <<(answer) < (1.5)> or <(answer) = (1.5)>> then
        say (You need to buy a half-price ticket) for (2) seconds
    else
        say (You need to buy a full price ticket) for (2) seconds
    end
end
```

Figure 4.41

Innovation Park

Modify the program "Ticket checking by ticket inspectors (2)", try to use "if..., then…" and "if..., then...else..." to finish the task "Ticket checking by ticket inspectors (2)".

Self-Assessment Room

Content	Result
I know the nested branch structure.	☆☆☆☆☆
I've learned building of nested branch structure.	☆☆☆☆☆
I've finished the task "Ticket checking by ticket inspectors (2)".	☆☆☆☆☆

Chapter 5 Message and Sensing

-- A wonderful circus show

John's new shopping center was completed as scheduled, and then John invited the famous circus to do a grand opening show. It was a wonderful show; there were Drums Conga by monkey and magic show elephant changing color, and the most fascinating one was Dog Maze. Everyone was fascinated by the animals, but John was curious that how did they communicate with each other and with the magician on stage?

Mr. Lee told John that in our daily life, humans can communicate with each other or with animals by language, body actions and even facial expression. We can regard the specific contents communicated by above ways as a message, and after receiving the message, the other side will give corresponding response.

Then in program design, how to deliver messages between sprites, between sprite and device and between devices? How does the sprite sense the environment during programming?

In this chapter, let's learn about message instruction and sensing instruction.

Learning objectives

- ☼ Learn how to broadcast and receive messages and sensing instructions as well as other related blocks.
- ☼ Experience to deliver messages and sense environment by programming, to sense the charm of computer programming.

Section 1 Message Instruction

-- Monkey drumming and elephant changing color

Wow, what a wonderful show, Kelly! The monkey drummed and the elephant changed color as they understand the instruction, amazing!
Yes, it was really fascinating! Mr. Lee told me right now that the dialogue between animals and what the magician said to the elephant are actually the message.

1. Message

Message is an independent communication content delivered from the sender to a or more object (s). And the process delivering messages is also called message broadcasting. After broadcasting the message, the receiver will give different responses.

In the above example, broadcasting that “Monkey begins to drum” is namely a message, and all of the people on and off the stage and at front stage and behind the scenes can receive the message, but only the monkey will begin to drum after receiving it. Even then the elephant received the message, it will not perform changing color.

So, what’s the message in program design?

In program design, message is a notice that can be received by all sprites on Stage. After receiving the message, the sprite (or device) will give corresponding responses and take actions.

2. Broadcasting and receiving of message

(1) Broadcasting of message

1. Broadcasting block of message

In DobotBlock, we use "Broadcast message 1" and "Broadcast message 1 and wait" to broadcast messages, see Figure 5.1.

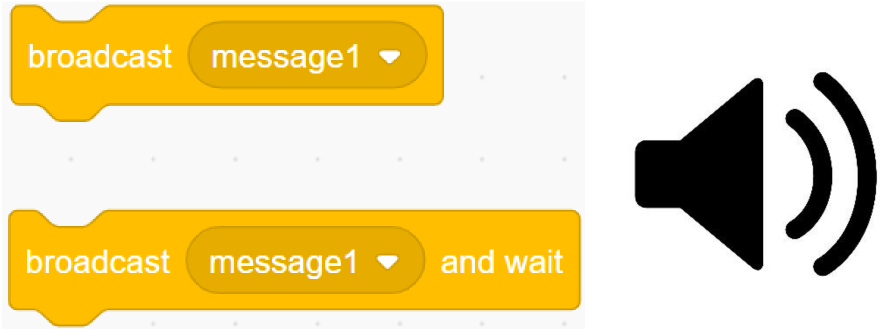
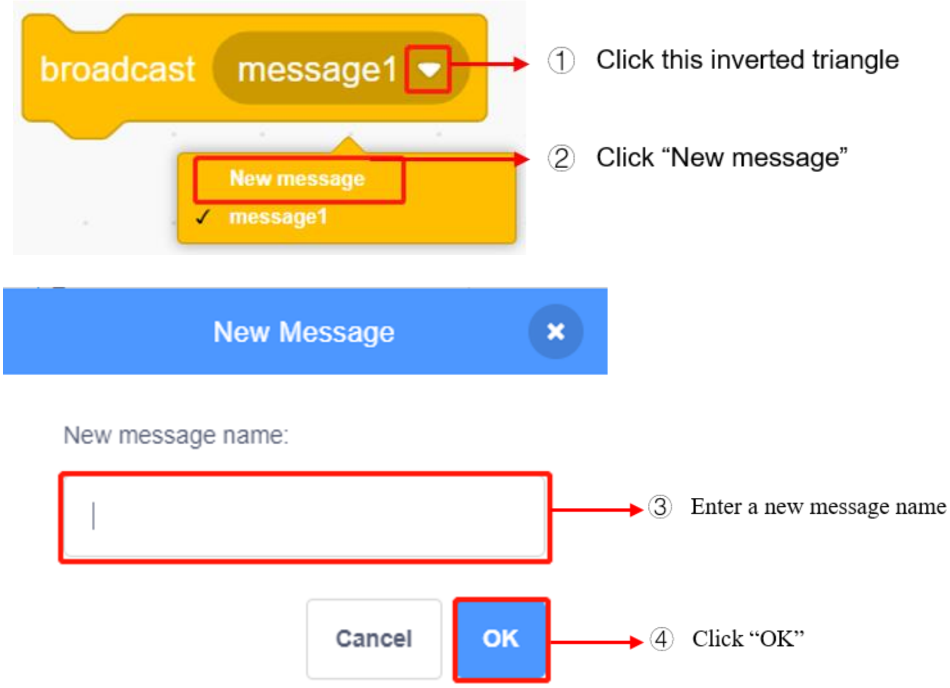


Figure 5.1

2. Create new messages

In DobotBlock, click Events module, drag "Broadcast message 1" or "Broadcast message 1 and wait" to Coding Area, click the inverted triangle symbol on the right of the block and then click "New message" in the pull-down menu. Then enter message name in the pop-up dialog box and click OK to create new message. Create new message with "Broadcast message 1", see Figure 5.2.



① Click this inverted triangle

② Click “New message”

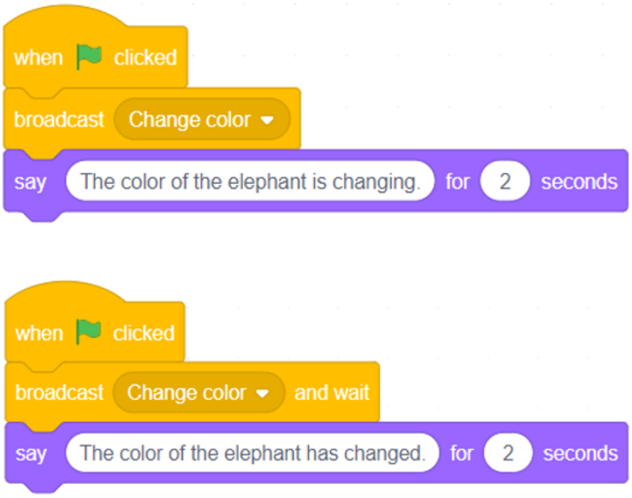
③ Enter a new message name

④ Click “OK”

Figure 5.2

"Broadcast message 1" is very similar to "Broadcast message 1 and wait", but the former will run program next immediately after broadcasting. The latter will, after the broadcasting, run program next until running of all scripts receiving the message.

For instance in the magic show elephant changing color, the program running sequence is different when different message broadcasting blocks are used, see Figure 5.3.



```
when ⚑ clicked
broadcast Change color ▾
say The color of the elephant is changing. for 2 seconds
```

After broadcasting "Change color", the elephant changing color and the saying "The color of the elephant is changing." happen at the same time.

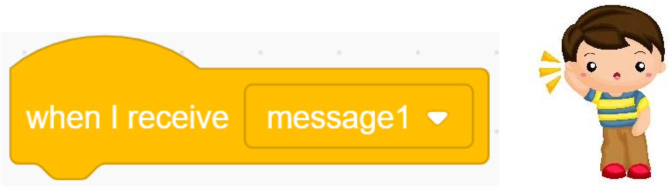
```
when ⚑ clicked
broadcast Change color ▾ and wait
say The color of the elephant has changed. for 2 seconds
```

After broadcasting "Change color", the elephant changes color before "The color of the elephant has changed." is spoke out.

Figure 5.3

(2) Receiving of message

In DobotBlock, we use "When I receive message 1" to receive message, see Figure 5.4; the sprite (or device) that has received the message will give response and take actions as needed.



```
when I receive message1 ▾
```

Figure 5.4

In DobotBlock, click Events module, drag "When I receive message 1" to Coding Area, click the inverted triangle symbol on the right of the block and then receive the message to be received in the pull-down menu, see Figure 5.5.

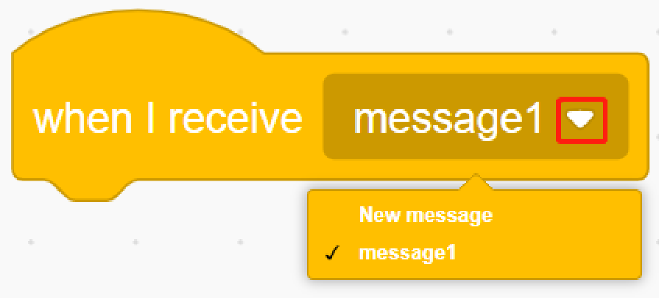


Figure 5.5

Taking the magic show elephant changing color as the example again, after broadcasting "Change color", the sprite elephant will receive the message and then change its color for 10 times, see Figure 5.6.

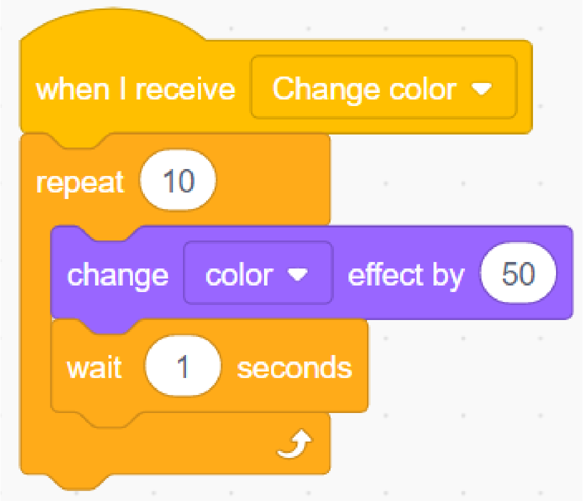

```
when I receive [Change color ▾]
repeat (10)
    change [color ▾] effect by (50)
    wait (1) seconds
```

Figure 5.6

Let's do it

✧ Run the program "Magic Show", observe the change to sprite on the stage.

✧ After running the program, substitute the "Broadcast message 1" with "Broadcast message 1 and wait" and then run the program again.

Research Office

1. Task release

Circus show: monkey makes opening introduction and then drums. After the show, monkey introduces the next show, namely dinosaur's dance.

2. Task analysis

There are three sprites in the circus show: monkey, drums conga and dinosaur. See Figure 5.7 for the action analysis of each of them, and see Figure 5.8 for the process of the circus show.

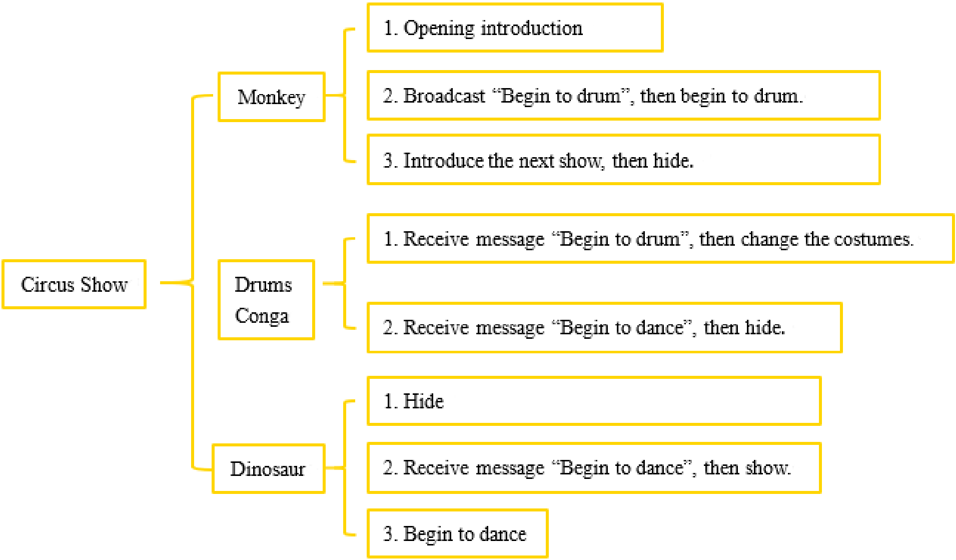

Circus Show
Monkey
1. Opening introduction
2. Broadcast "Begin to drum", then begin to drum.
3. Introduce the next show, then hide.
Drums Conga
1. Receive message "Begin to drum", then change the costumes.
2. Receive message "Begin to dance", then hide.
Dinosaur
1. Hide
2. Receive message "Begin to dance", then show.
3. Begin to dance

Figure 5.7

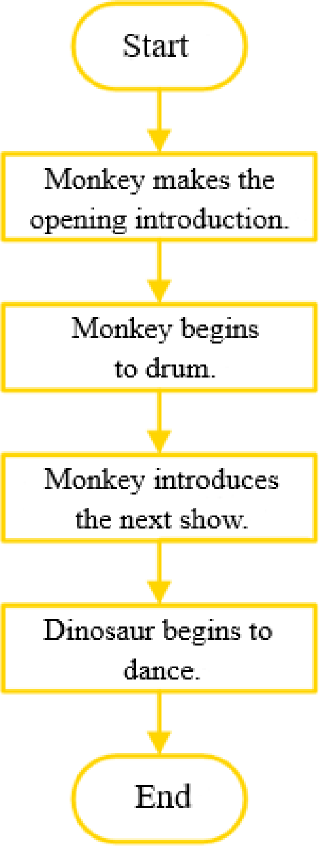
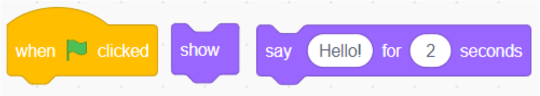
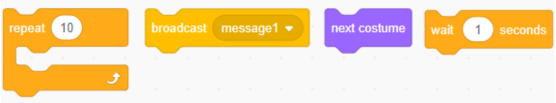
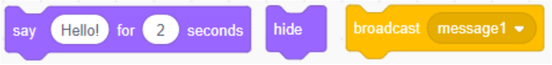
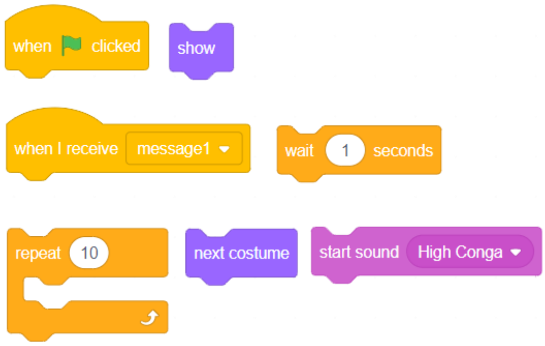
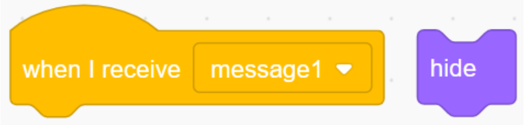
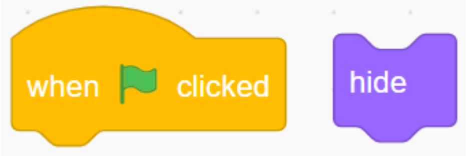
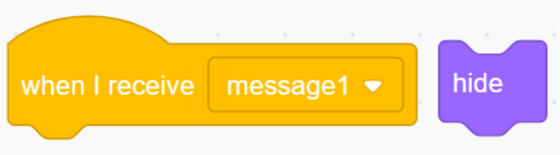
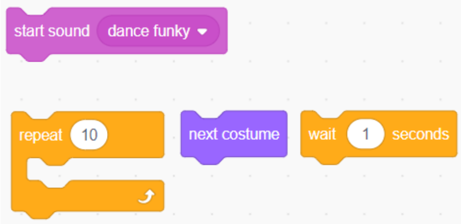

Start
Monkey makes the opening introduction.
Monkey begins to drum.
Monkey introduces the next show.
Dinosaur begins to dance.
End

Figure 5.8

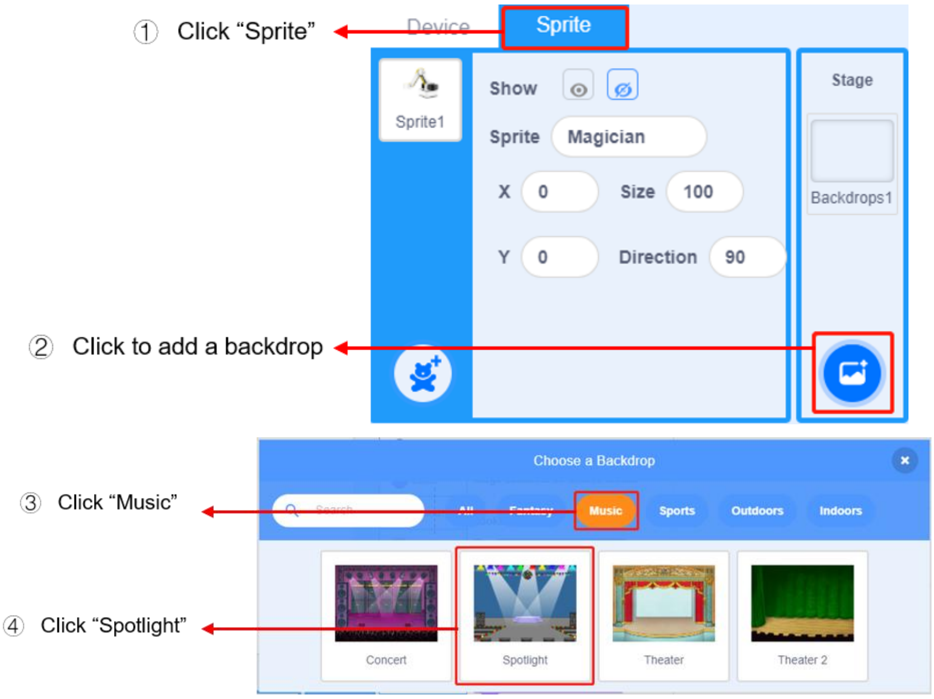
3. Script planning

Sprite	Task description	Block
	1. Monkey makes opening introduction.	 when [flag] clicked show say Hello! for 2 seconds
	2. Broadcast "Begin to drum" and then begin to drum.	 repeat 10 broadcast message1 next costume wait 1 seconds
	3. Introduce the next show and then hide.	 say Hello! for 2 seconds hide broadcast message1
	1. Receive message "Begin to drum" and then change the costumes and start sound.	 when [flag] clicked show when I receive message1 wait 1 seconds repeat 10 next costume start sound High Conga
	2. Receive message "Begin to dance" and then hide.	 when I receive message1 hide
	1. Hide before receiving message "Begin to dance".	 when [flag] clicked hide
	2. Receive message "Begin to dance" and then show.	 when I receive message1 hide
	3. Start sound, and dinosaur begins to dance.	 start sound dance funky repeat 10 next costume wait 1 seconds

Processing zone

Add backdrop and sprite

Step 1. Add stage backdrop, see Figure 5.9 for the steps.



① Click “Sprite”

② Click to add a backdrop

③ Click “Music”

④ Click “Spotlight”

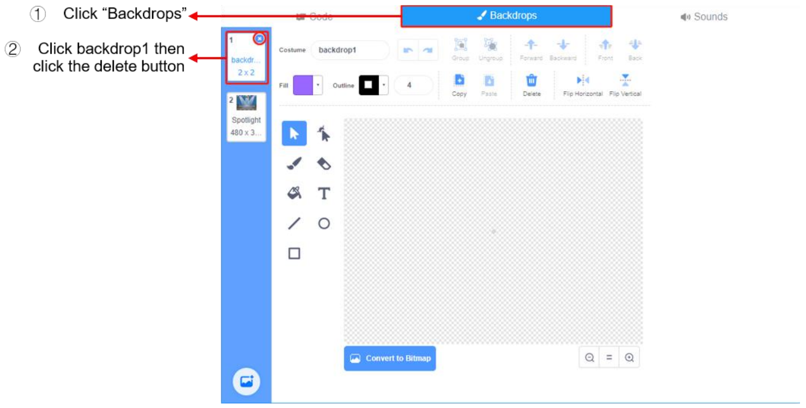
Sprite: Magician

X: 0 Size: 100

Y: 0 Direction: 90

Figure 5.9

Step 2. Delete the default backdrop. Delete backdrop 1 on the page of Backdrops, see Figure 5.10.



① Click “Backdrops”

② Click backdrop1 then click the delete button

Costume: backdrop1

Figure 5.10

Add backdrop and sprite

Step 3: Add sprite. Add sprites "Monkey", "Drums Conga" and "Dinosaur5" in Sprite, delete the default sprite, see Figure 5.11 for the position of sprites on the stage.

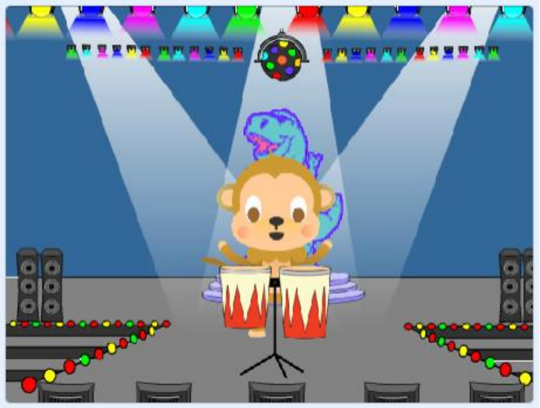
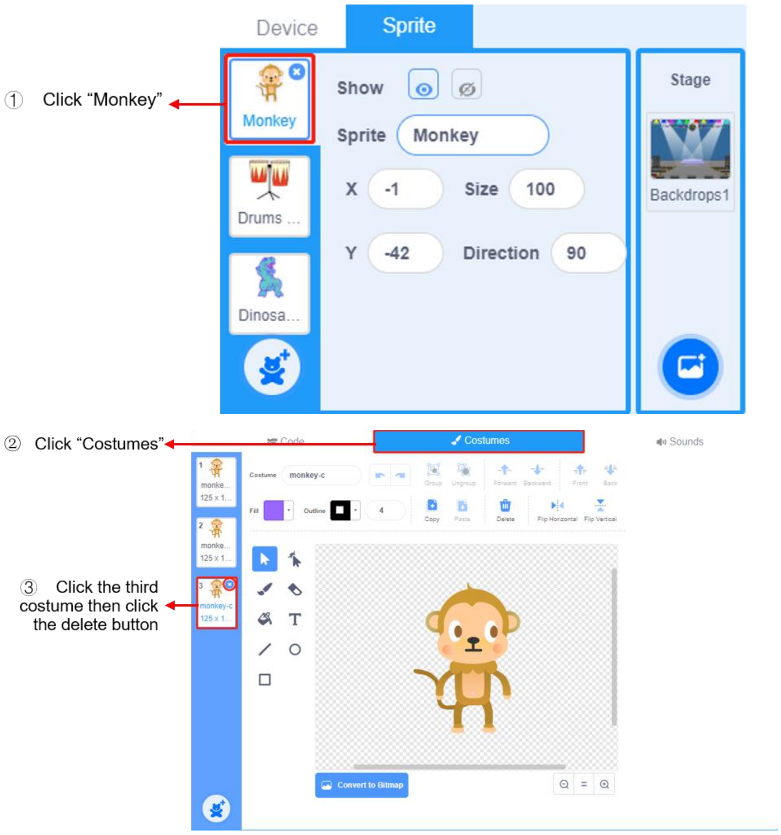


Figure 5.11 Position of Sprites on Stage.

Step 4. Delete costumes. Monkey needs only two costumes during drumming: handing up and waving hand downward; delete the third costume of monkey, see Figure 5.12.



① Click "Monkey"

② Click "Costumes"

③ Click the third costume then click the delete button

Sprite: Monkey

Show

X: -1 Size: 100

Y: -42 Direction: 90

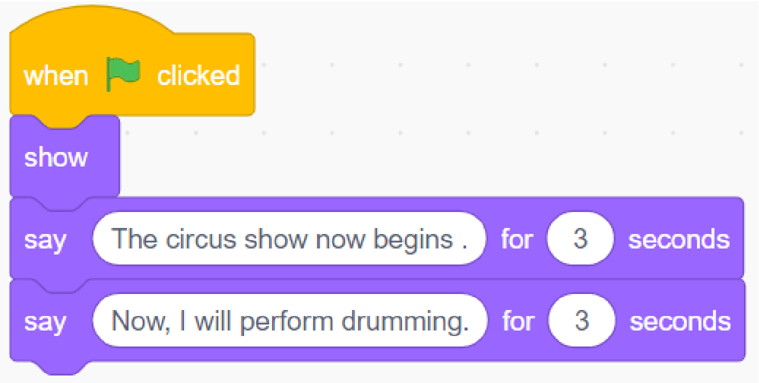
Costume: monkey-c

Figure 5.12

Write sprite script

Step 1. Write the script of monkey.

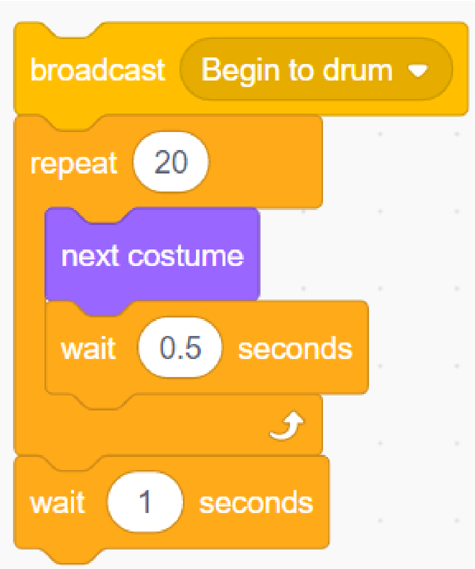
Monkey makes opening introduction before its show, see Figure 5.13.



```
when 🏳 clicked
show
say (The circus show now begins .) for (3) seconds
say (Now, I will perform drumming.) for (3) seconds
```

Figure 5.13 Opening Introduction

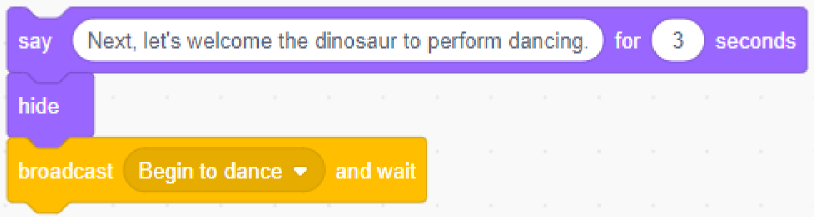
After broadcasting "Begin to drum", then monkey begins to drum, see Figure 5.14.



```
broadcast (Begin to drum ▾)
repeat (20)
    next costume
    wait (0.5) seconds
end
wait (1) seconds
```

Figure 5.14

Introduce the next show, and then hide, see Figure 5.15.

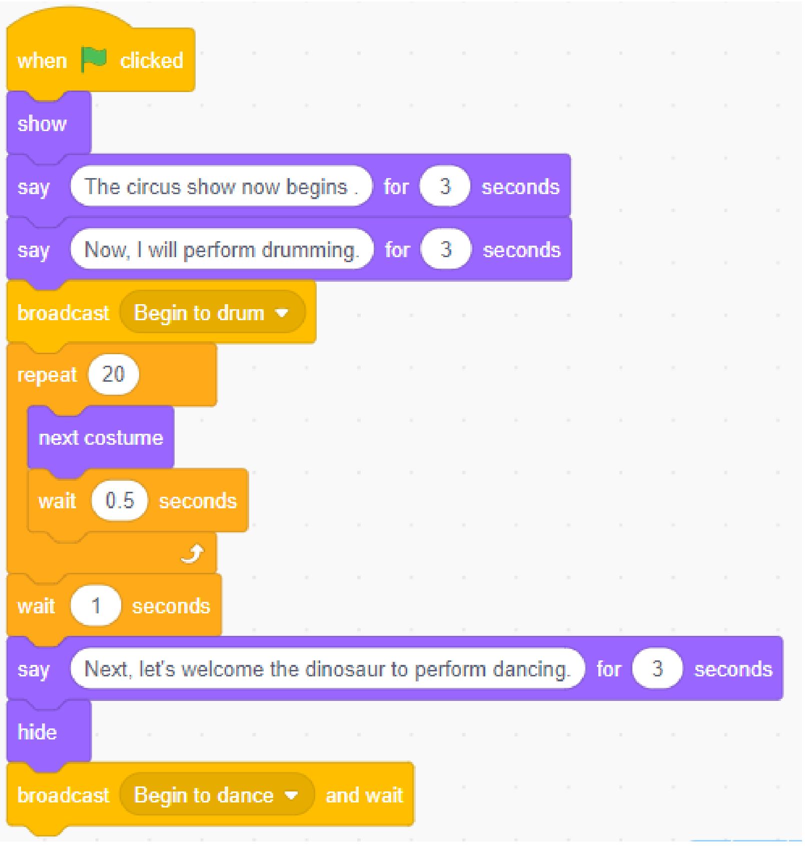


```
say (Next, let's welcome the dinosaur to perform dancing.) for (3) seconds
hide
broadcast (Begin to dance ▾) and wait
```

Figure 5.15

Write sprite script

Integrate code. After the opening introduction by monkey, the program broadcasts, and monkey begins to drum. After the performance, monkey introduces the next show and then hide, see Figure 5.16.

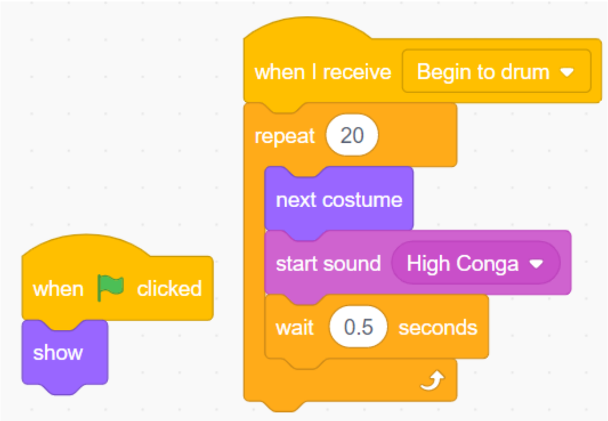


```
when 🏳 clicked
show
say The circus show now begins . for 3 seconds
say Now, I will perform drumming. for 3 seconds
broadcast Begin to drum ▾
repeat 20
    next costume
    wait 0.5 seconds
↺
wait 1 seconds
say Next, let's welcome the dinosaur to perform dancing. for 3 seconds
hide
broadcast Begin to dance ▾ and wait
```

Figure 5.16

Step 2. Write script of Drums Conga.

Receive message "Begin to drum" and then change the costumes and start sound, see Figure 5.17.



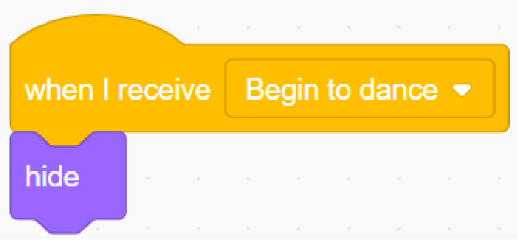
```
when 🏳 clicked
show
```

```
when I receive Begin to drum ▾
repeat 20
    next costume
    start sound High Conga ▾
    wait 0.5 seconds
↺
```

Figure 5.17

Write sprite script

After receiving the message "Begin to dance", the "Drums Conga" then hides, see Figure 5.18.



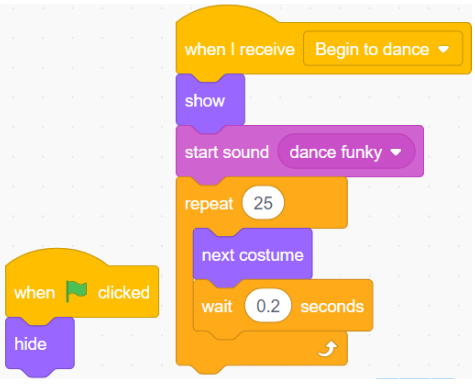
```
when I receive [Begin to dance ▾]
hide
```

Figure 5.18

Step 3. Write script of dinosaur.

Click Sprite and sprite "Dinosaur5", and then begin to write script of dinosaur.

Dinosaur hides before receiving message "Begin to dance". After receiving the message "Begin to dance", Dinosaur then shows and begins to dance, see Figure 5.19.



```
when 🏁 clicked
hide

when I receive [Begin to dance ▾]
show
start sound [dance funky ▾]
repeat (25)
    next costume
    wait (0.2) seconds
```

Figure 5.19

Innovation park

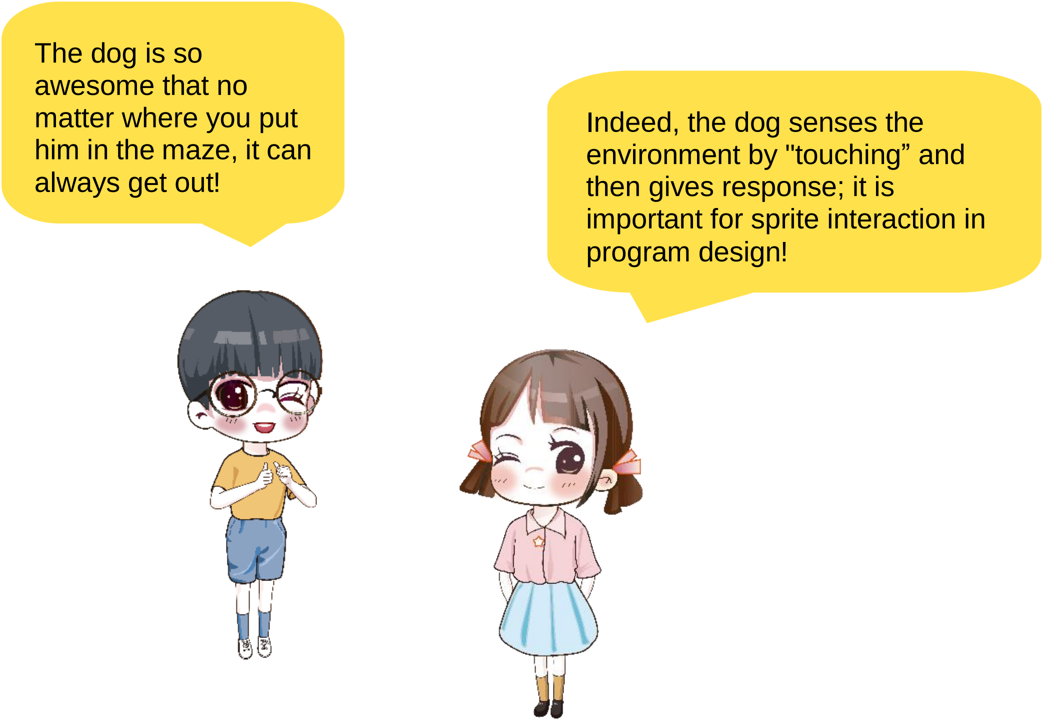
Modify the video of "circus show": there are two shows in the original program. Now we need to add another one "poetry reading"; choose sprite "Starfish" to read poems on its own decision.

Self-Assessment Room

Content	Result
Now I know what’s message.	☆☆☆☆☆
I’ve learned the block "broadcasting of message" and its use.	☆☆☆☆☆
I’ve learned the block "receiving of message" and its use.	☆☆☆☆☆
I’ve finished "circus show".	☆☆☆☆☆

Section 2 Sensing Instruction

-- Maze Escape


The dog is so awesome that no matter where you put him in the maze, it can always get out!
Indeed, the dog senses the environment by "touching” and then gives response; it is important for sprite interaction in program design!

Let's repeat the thrilling and exciting process by programming!

1. Setting Dog!

After start of the performance, we need to set Dog at a corner of the maze. When setting the sprite, we need to move mouse to the position of the sprite, mouse down the left key and drag the sprite to the designated place, and then release the left key of mouse.

We can find "Mouse down?” from Sensing module in software block area, see Figure 5.20.

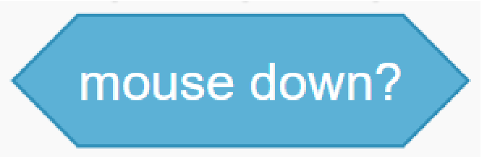
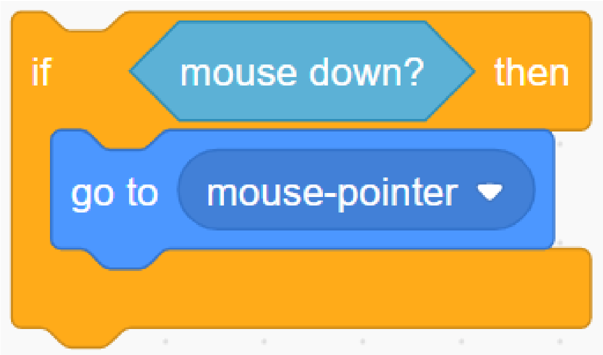


Figure 5.20

Choose the position in maze, mouse down, and then the Dog will be set at the position mouse-down; see Figure 5.21 for some programs.

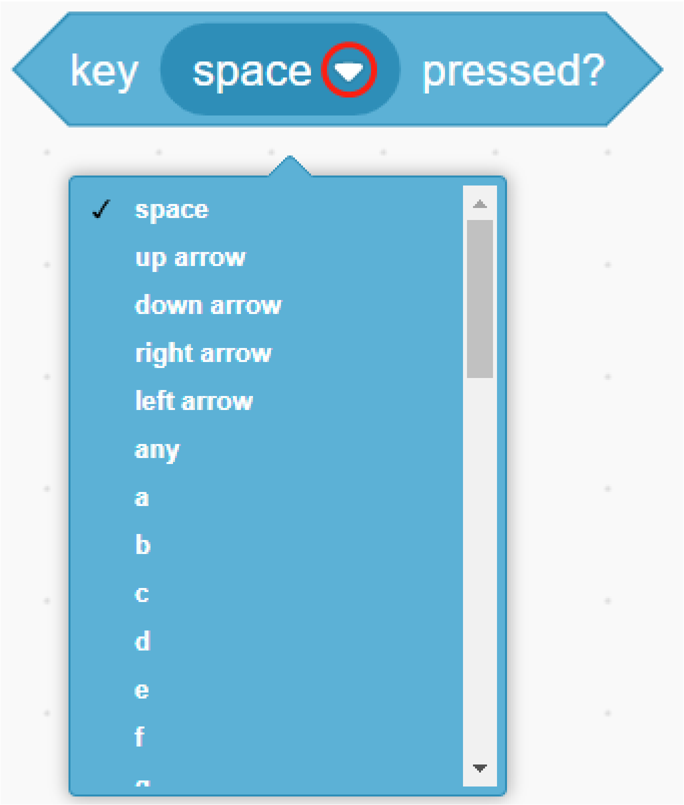


```
if mouse down? then
    go to mouse-pointer
```

Figure 5.21

2. Navigating the maze of Dog

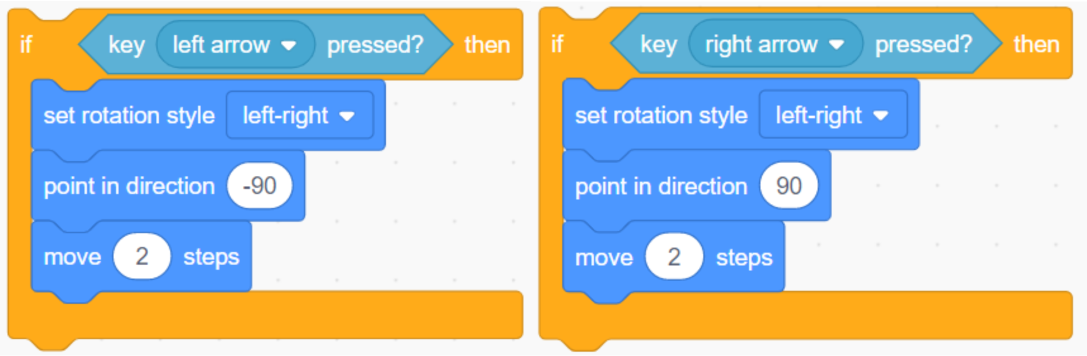
After setting Dog, we can command the Dog to begin to navigate the maze by up arrow, down arrow, left arrow and right arrow. We can find “Key (space) pressed?” from Sensing module in software block area. Click the white inverted triangle symbol, and then we can see several options, see Figure 5.22.



```
key space pressed?
```

Figure 5.22

Part of the program for dog walking is shown in figure 5.23.



```
if key left arrow pressed? then
    set rotation style left-right
    point in direction -90
    move 2 steps
```

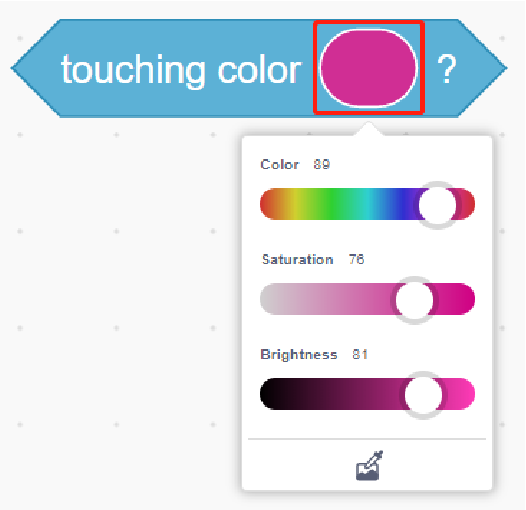
```
if key right arrow pressed? then
    set rotation style left-right
    point in direction 90
    move 2 steps
```

Figure 5.23

3. Avoiding getting through black wall

How to keep Dog from getting through black wall in maze.

We can find "Touching color ()?" block from Sensing module in software block area. Click the elliptical position on right of the block, and then we can choose the color, saturation and brightness, see Figure 5.24.



touching color ?

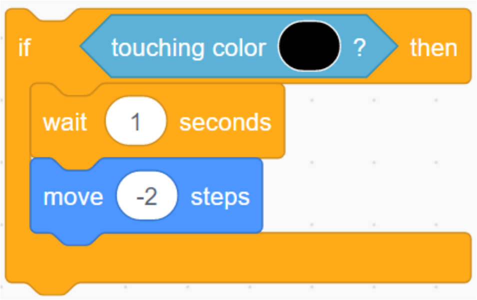
Color 89

Saturation 76

Brightness 81

Figure 5.24

If Dog attempts to get through black wall, he will wait for 1s and then move back for 2 steps, see Figure 5.25.



```
if touching color ? then
    wait 1 seconds
    move -2 steps
```

Figure 5.25

4. Arriving Victory Gate

When controlling Dog to touch the exit, the Dog arrives at the Victory Gate. We can find "Touching (mouse-pointer)?" from the sensing block in software block area, click the white inverted triangle symbol, then we can see two basic options "mouse-pointer" and "edge". When adding new sprite on stage, the block will add the option of new sprite automatically, see Figure 5.26.

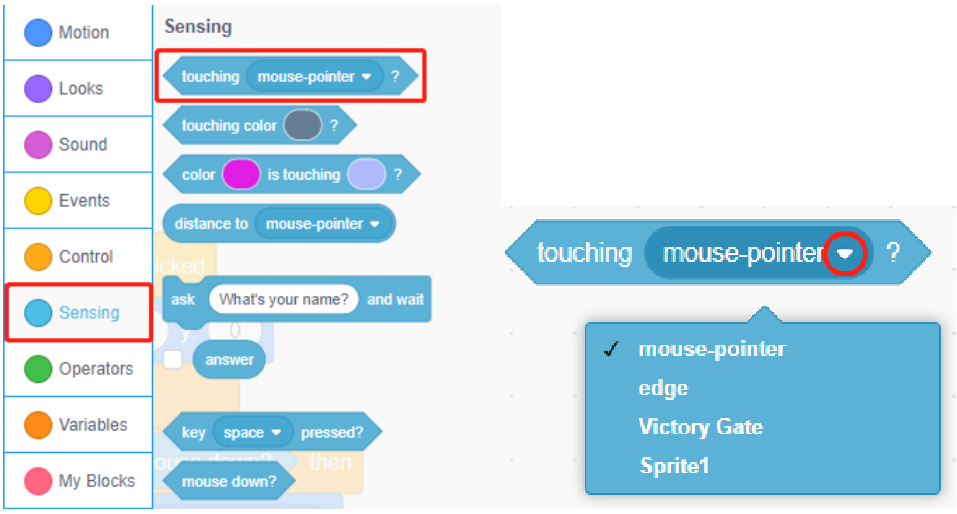
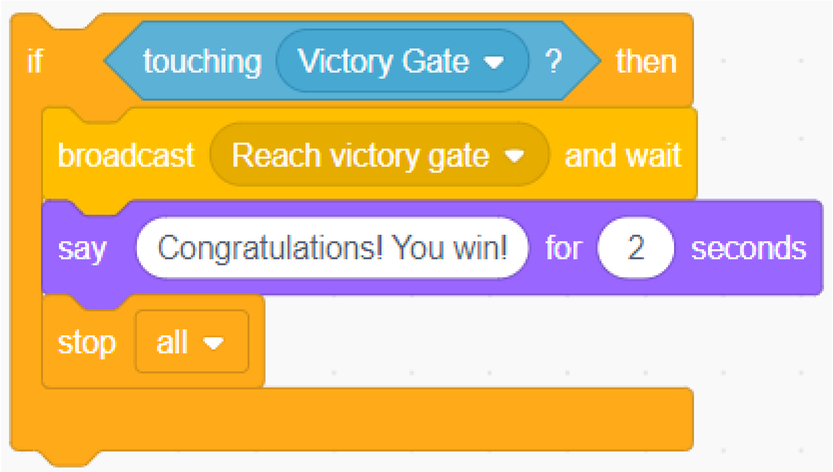


Figure 5.26

In the example above, some program for arriving at Victory Gate is shown in Figure 5.27.



```
if touching Victory Gate ? then
    broadcast Reach victory gate and wait
    say Congratulations! You win! for 2 seconds
    stop all
```

Figure 5.27

Let's do it

✧ Run the program "Dog Maze.sb3".

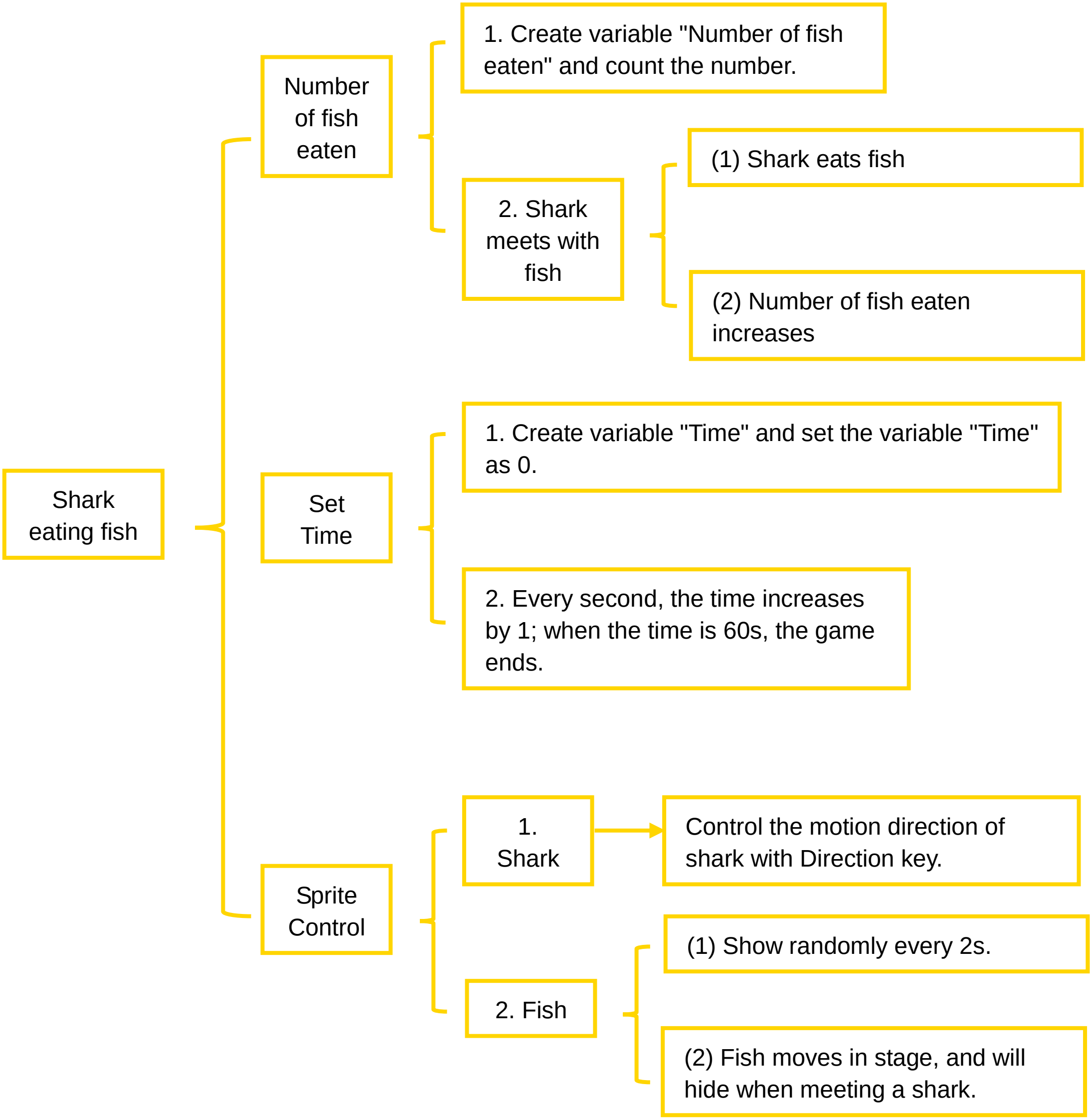
Research Office

1. Task release

Shark eating fish: there are different kinds of fishes in the vast sea, and when a shark meets a fish, the shark will eat the fish. Count the number of fish eaten by shark within given time.

2. Task analysis

We can divide the game into three parts: number of fish eaten, game time and sprite control. See Figure 5.28 for task analysis of shark eating fish.



Shark eating fish

Number of fish eaten

1. Create variable "Number of fish eaten" and count the number.

2. Shark meets with fish

(1) Shark eats fish

(2) Number of fish eaten increases

Set Time

1. Create variable "Time" and set the variable "Time" as 0.

2. Every second, the time increases by 1; when the time is 60s, the game ends.

Sprite Control

1. Shark

Control the motion direction of shark with Direction key.

2. Fish

(1) Show randomly every 2s.

(2) Fish moves in stage, and will hide when meeting a shark.

Figure 5.28

See Figure 5.29 for flow chart of shark eating fish.

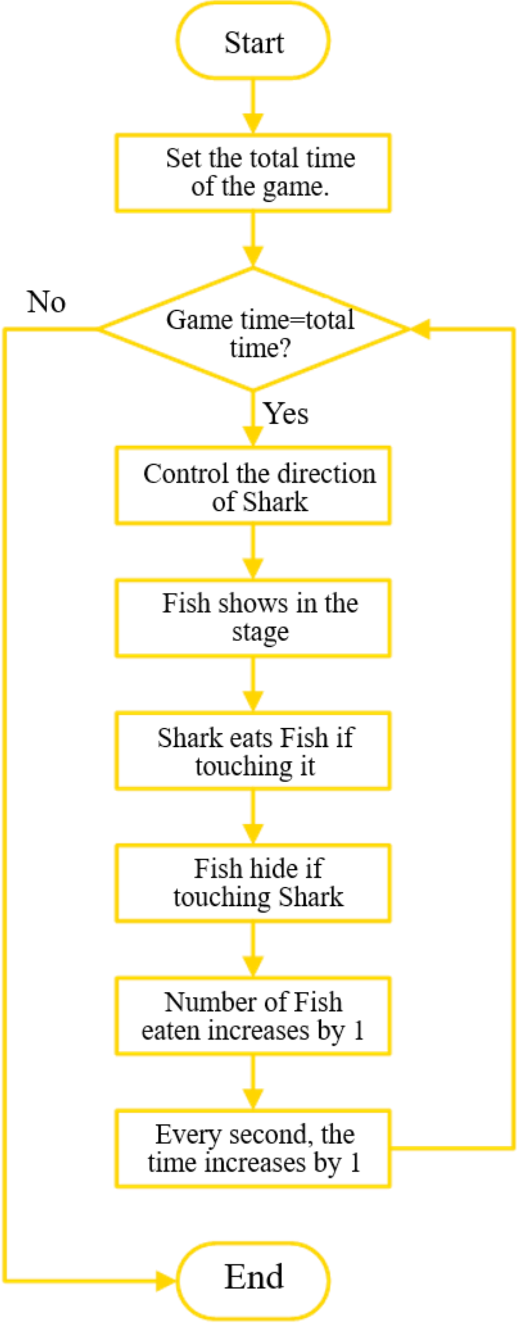
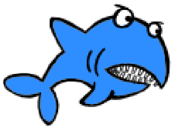
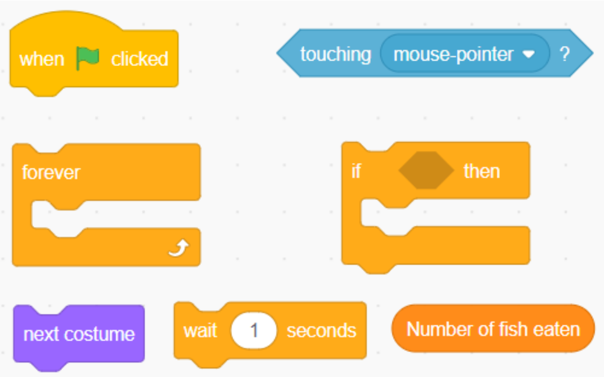
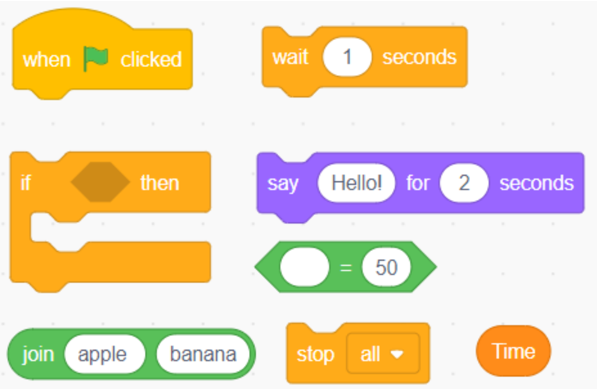
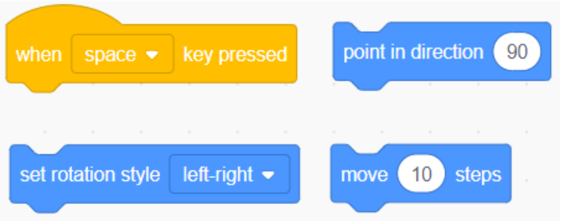
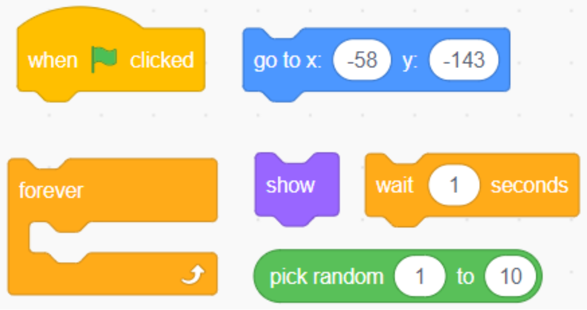
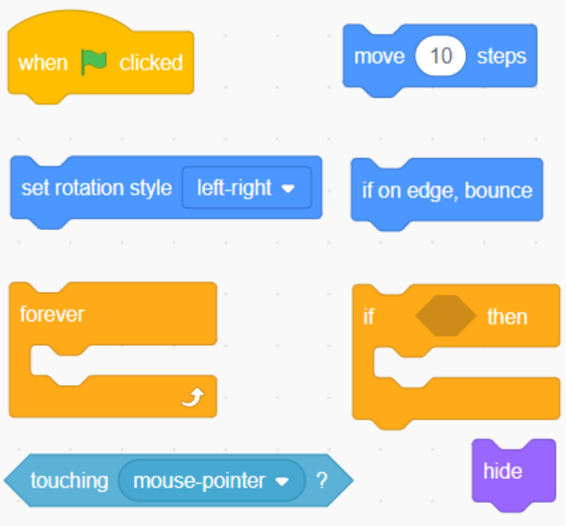

Start
Set the total time of the game.
Game time=total time?
No
Yes
Control the direction of Shark
Fish shows in the stage
Shark eats Fish if touching it
Fish hide if touching Shark
Number of Fish eaten increases by 1
Every second, the time increases by 1
End

Figure 5.29

3. Script planning

Sprite	Task description	Block
	1. Count number of fish eaten.	 when ⚑ clicked touching mouse-pointer ? forever if then next costume wait 1 seconds Number of fish eaten
	2. Set the time	 when ⚑ clicked wait 1 seconds if then say Hello! for 2 seconds = 50 join apple banana stop all Time
	3. Control the motion direction of shark with Direction key.	 when space key pressed point in direction 90 set rotation style left-right move 10 steps
	1. Show at random position of stage every 2s.	 when ⚑ clicked go to x: -58 y: -143 forever show wait 1 seconds pick random 1 to 10

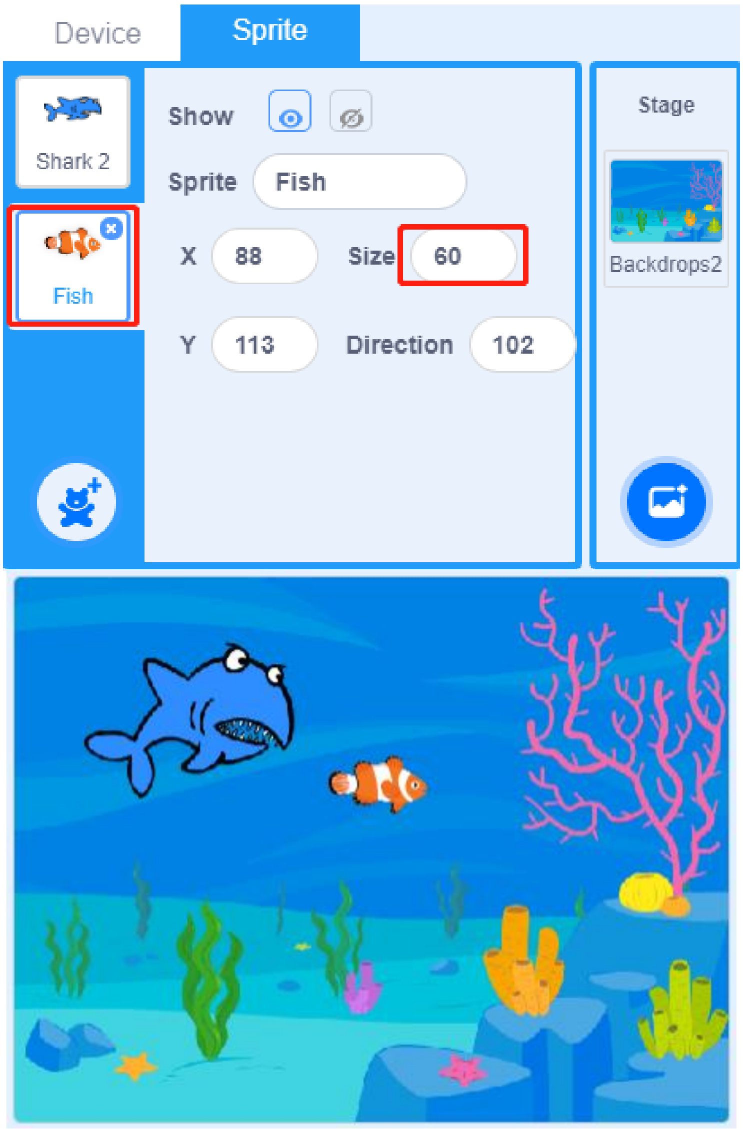
Sprite	Task description	Block
	2. Fish moves around stage. Fish hides when meeting a shark.	 when clicked move 10 steps set rotation style left-right if on edge, bounce forever if then touching mouse-pointer ? hide

Processing zone

Add backdrop and sprite

Step 1. Add backdrop "Underwater 1" and delete the default backdrop.

Step 2: Add sprite. Add sprite "Shark 2" and "Fish" in Sprite, and then delete the default sprite; change the size of sprite "Fish" into 60, see Figure 5.30 for stage and sprites on the stage.



Show

Sprite Fish

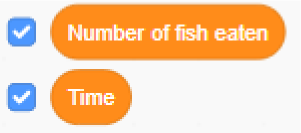
X 88 Size 60

Y 113 Direction 102

Figure 5.30

Create variable

Click "Variable" module and "Set up a variable", enter variable name "Number of fish eaten" and click OK. Create variable "Time" with the same method, see Figure 5.31 for the two variables.



☑ Number of fish eaten

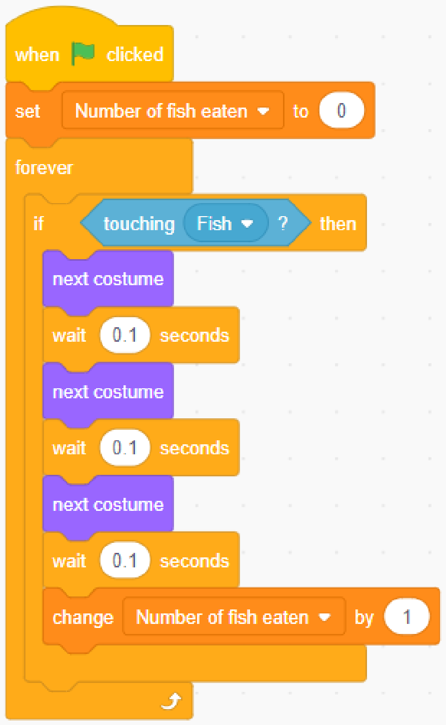
☑ Time

Figure 5.31

Write sprite script

Step 1. Write the script of shark.

Number of fish eaten. Create variable "Number of fish eaten" to count the number of fish eaten by shark. The variable "Number of fish eaten" will increase by 1 each time after shark eats a fish, see Figure 5.32.

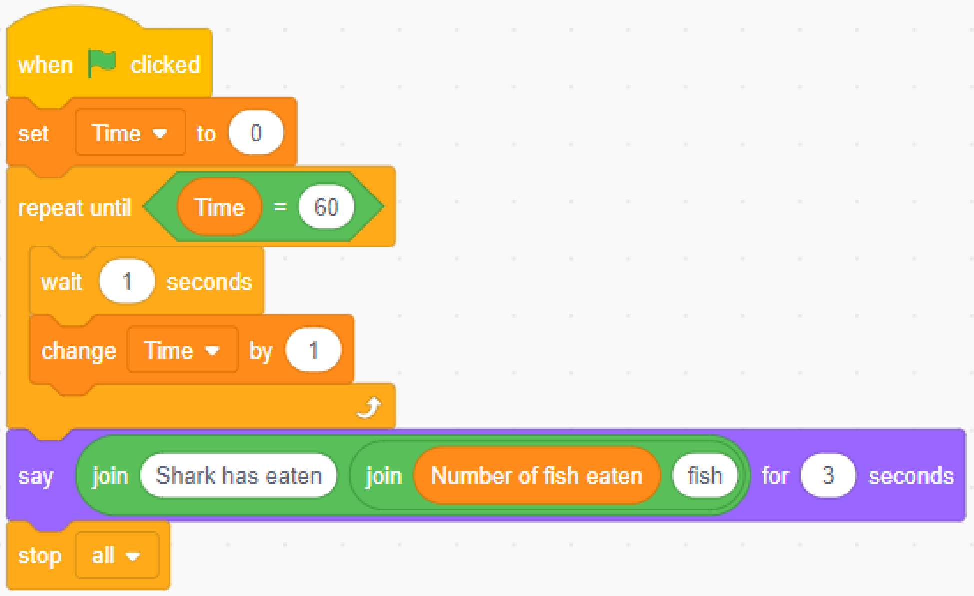


```
when 🏴 clicked
set Number of fish eaten ▾ to 0
forever
    if touching Fish ▾ ? then
        next costume
        wait 0.1 seconds
        next costume
        wait 0.1 seconds
        next costume
        wait 0.1 seconds
        change Number of fish eaten ▾ by 1
```

Figure 5.32

Write sprite script

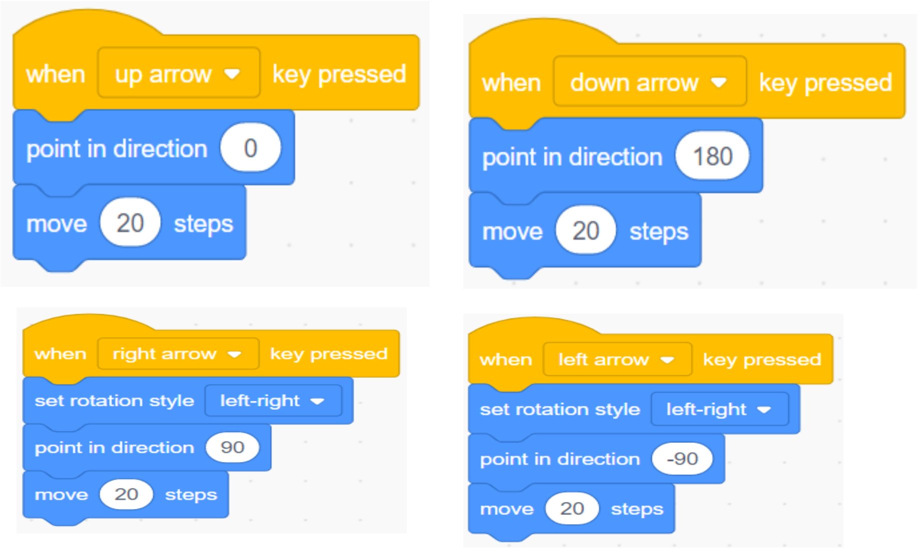
Set the time. Create variable "Time" to count the game time. It is only 60s for each game. When the game time is used out, Shark will say the total number of fish eaten, and then stop all scripts, see Figure 5.33.



```
when ⚑ clicked
set Time ▾ to 0
repeat until Time = 60
    wait 1 seconds
    change Time ▾ by 1
↻
say join Shark has eaten join Number of fish eaten fish for 3 seconds
stop all ▾
```

Figure 5.33

Control the motion direction of shark. Shark can move up and down and left and right. We can control the motion direction of shark with Direction key, see Figure 5.34.



```
when up arrow ▾ key pressed
point in direction 0
move 20 steps
```

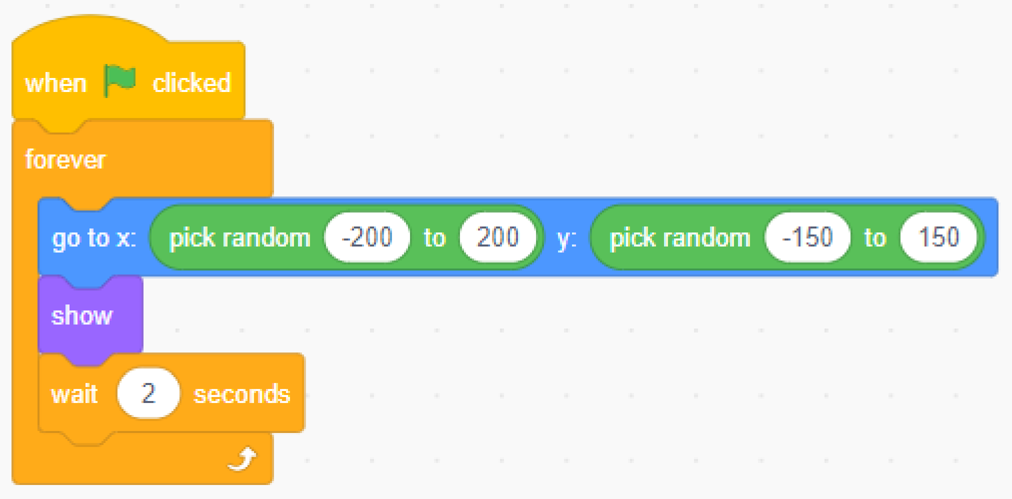
```
when down arrow ▾ key pressed
point in direction 180
move 20 steps
```

```
when right arrow ▾ key pressed
set rotation style left-right ▾
point in direction 90
move 20 steps
```

```
when left arrow ▾ key pressed
set rotation style left-right ▾
point in direction -90
move 20 steps
```

Write sprite script

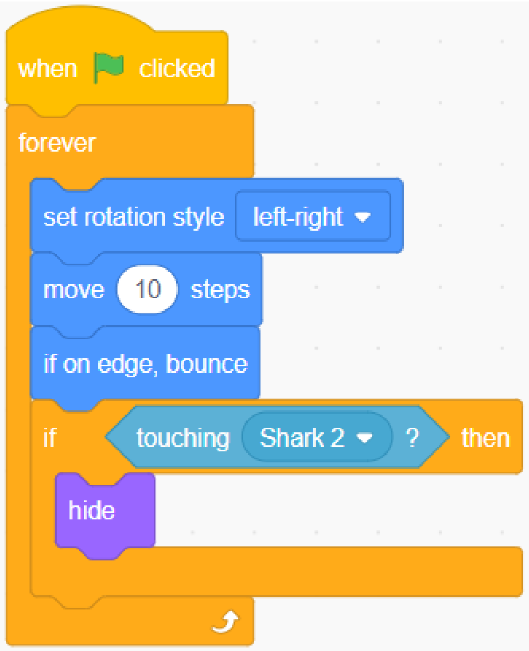
Step 2. Click sprite "Fish", and then begin to write script of fish.
To make fish show on the stage all the time, we set it to show every 2s, see Figure 5.35.



```
when 🏁 clicked
forever
    go to x: (pick random (-200) to (200)) y: (pick random (-150) to (150))
    show
    wait (2) seconds
↻
```

Figure 5.35

Fish moves on stage and hides when meeting with a shark, see Figure 5.36.



```
when 🏁 clicked
forever
    set rotation style [left-right ▾]
    move (10) steps
    if on edge, bounce
    if <touching [Shark 2 ▾] ?> then
        hide
↻
```

Figure 5.36

Innovation Park

Modify the game shark eating fish: add an effect that shark will get large each time after eating a fish; the more shark eats, the larger it will be. Boys and girls, lets' challenge to see who has the largest shark.

Self-Assessment Room

Content	Result
I know the four sensing blocks.	☆☆☆☆☆
I've learned how to use the four sensing blocks.	☆☆☆☆☆
I’ve finished "shark eats fish".	☆☆☆☆☆

Chapter 6
Comprehensive Project

-- A new medical project in the town

With the development of technology, self-service rises, and people change their way of shopping. There are already many self-service systems in John's smart supermarket. In order to provide more convenient medical services to residents around, John wants to add an auto medicine vending system next to his supermarket.

In the field of medicine circulation, the auto medicine vending system is quick, convenient and intelligent, allowing people to buy the medicines they need at any time of the day. So in this chapter, let's comprehensively apply what we’ve learned about programming to build an auto medicine vending system.

Learning objectives

- ☼ Design the interactive interface of an auto medicine vending system to cultivate aesthetic consciousness and design ability
- ☼ Write the program of an auto medicine vending system to improve programming ability
- ☼ Optimize the auto medicine vending system and develop engineering thinking

Auto Medicine Vending System

Kelly, it would be great if there is an auto medicine vending system next to my supermarket. Users nearby could purchase medicines themselves anytime they want through the interactive interface of the system.

That's a great idea! No salesmen would be required for shopping. This will help reduce labor costs and make people's lives better!

Let's discuss it

✧ Boys and girls, have you ever seen an auto vending system? Figure 6.1 shows an auto vending system in our life. Have a look about it, and let’s think what menu interfaces will you need to design on the auto vending system? What interactive functions will you need to realize through the program? How to control the robot to take the corresponding medicines automatically?

Figure 6.1

Research Office

1. Task release

Design and complete an auto medicine vending system with software interaction and robot control.

Requirements: For the software interaction part, users can click a button to enter the pharmacy, choose medicines, and delete those chose by mistake. The program will show the Total Price and Medicine List on Purchase Interface during purchasing, and then come to the End Interface after purchase. There are Start Interface, Purchase Interface and End Interface on the software interaction interface, see Figure 6.2.

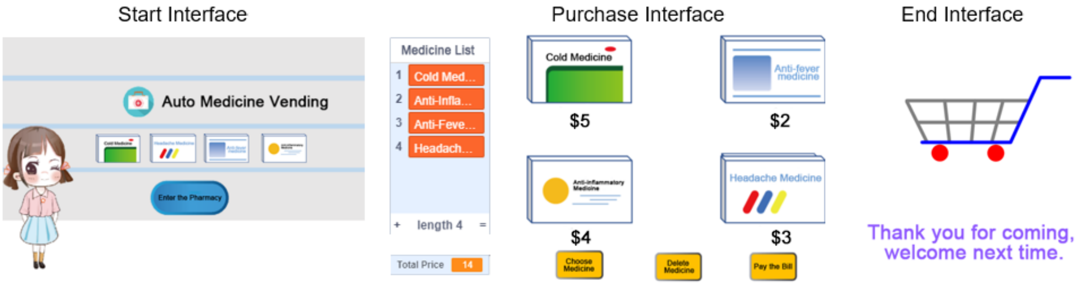


Figure 6.2

As for the robot control part, the robot takes medicine according to the "Medicine List", and then puts them in the Laydown Area. See Figure 6.3 for position of device.

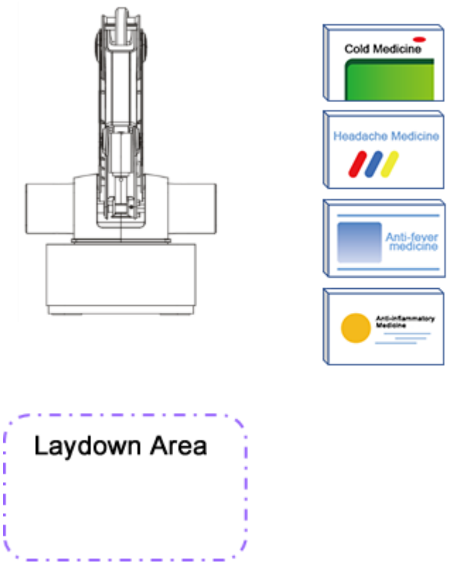


Figure 6.3

2. Task analysis

There are six steps to build an auto medicine vending system, and see Figure 6.4 for the overall task flow.

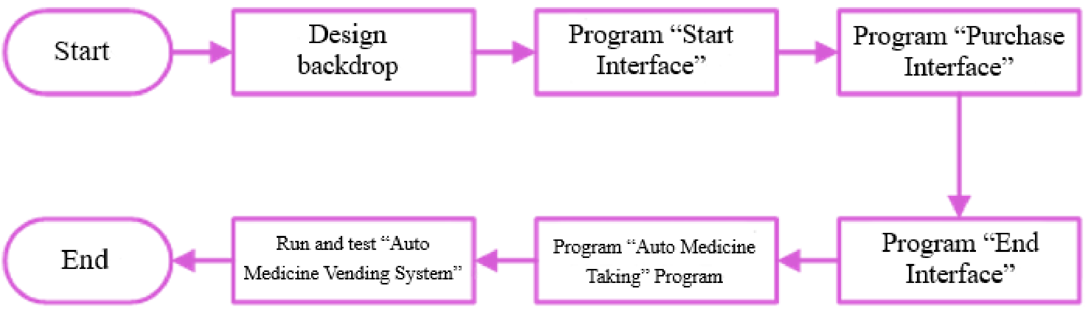

Start
Design backdrop
Program "Start Interface"
Program "Purchase Interface"
Program "End Interface"
Program "Auto Medicine Taking" Program
Run and test "Auto Medicine Vending System"
End

Figure 6.4

Step 1. Design backdrop. Design "Backdrop of Start Interface", "Backdrop of Purchase Interface" and "Backdrop of End Interface" respectively, see Figure 6.5.

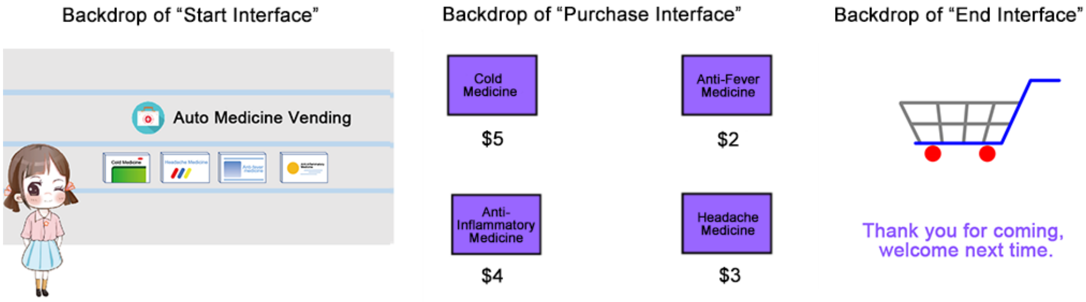

Backdrop of “Start Interface”
Auto Medicine Vending
Backdrop of “Purchase Interface”
Cold Medicine
$5
Anti-Fever Medicine
$2
Anti-Inflammatory Medicine
$4
Headache Medicine
$3
Backdrop of “End Interface”
Thank you for coming, welcome next time.

Figure 6.5

Step 2. Program "Start Interface". Analyze tasks of each sprite in the "Start Interface" and program for each one. See Figure 6.6 for the work flow chart of "Start Interface".

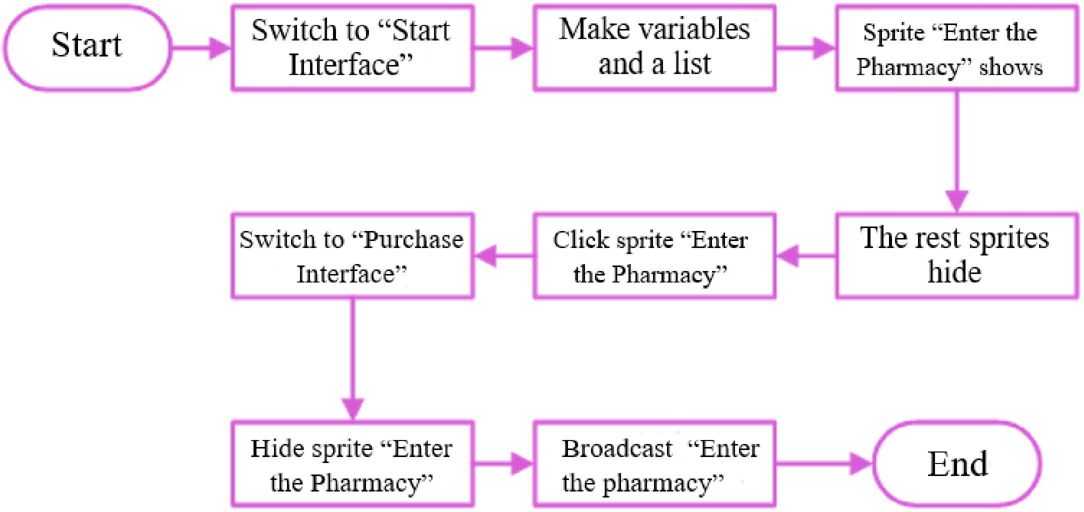
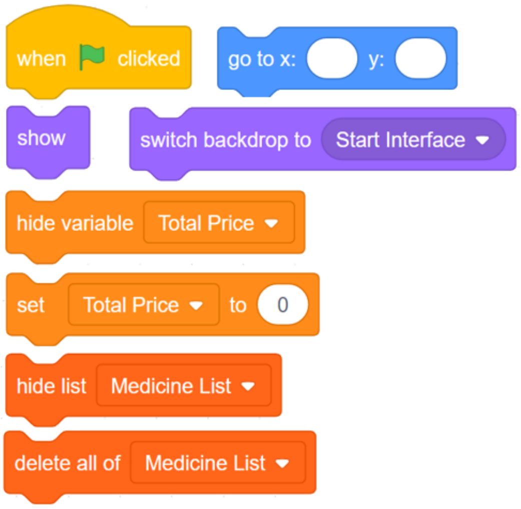
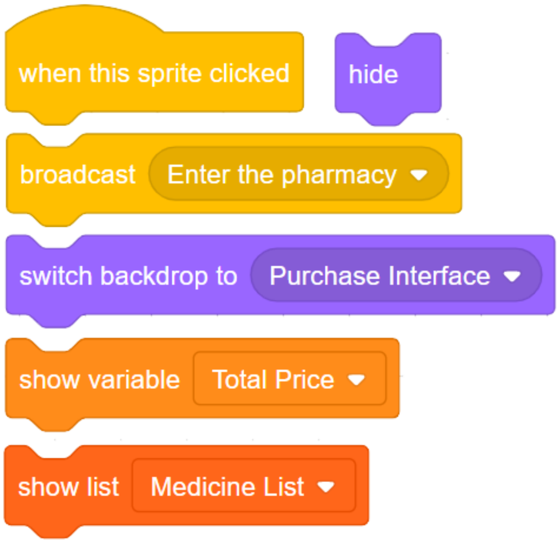
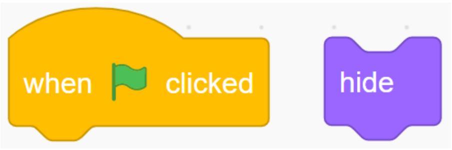
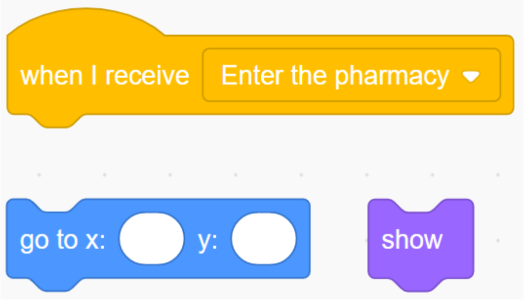

Start
Switch to “Start Interface”
Make variables and a list
Sprite “Enter the Pharmacy” shows
The rest sprites hide
Click sprite “Enter the Pharmacy”
Switch to “Purchase Interface”
Hide sprite “Enter the Pharmacy”
Broadcast “Enter the pharmacy”
End

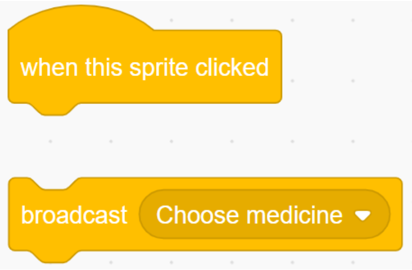
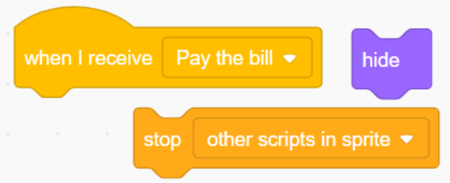
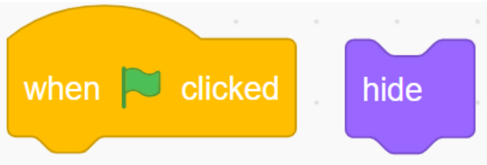
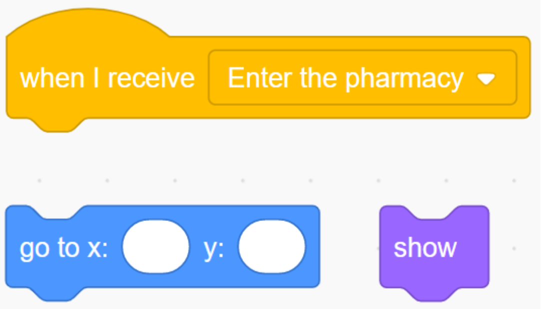
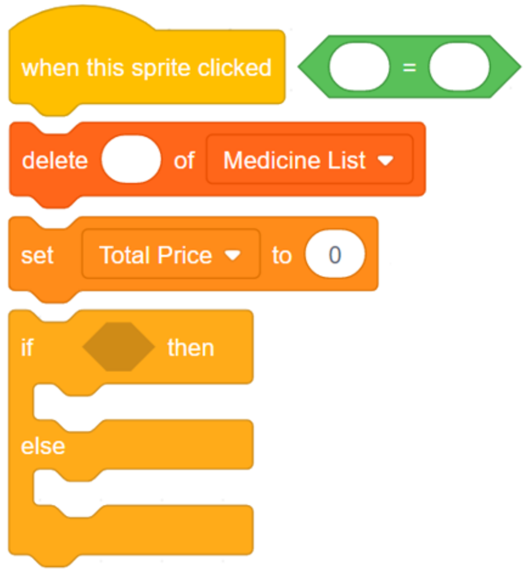
Figure 6.6

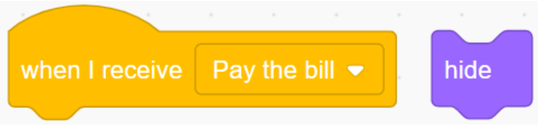
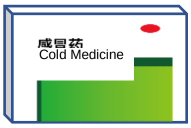
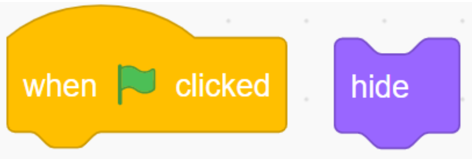
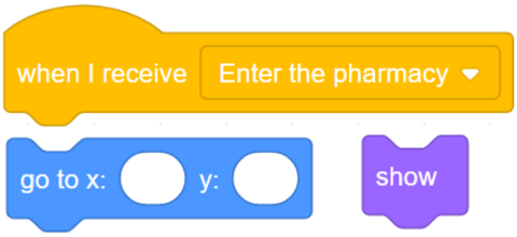
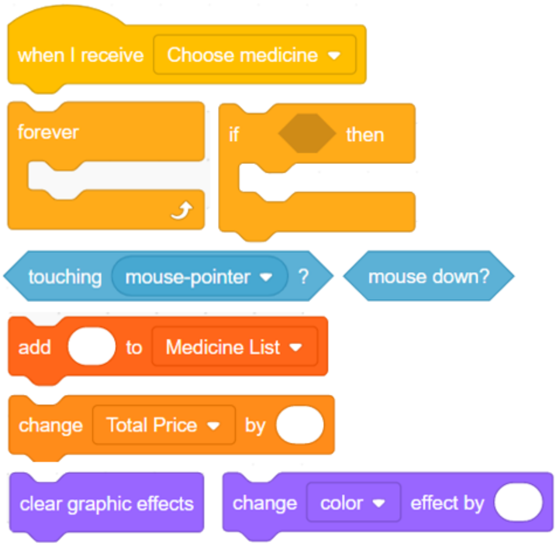
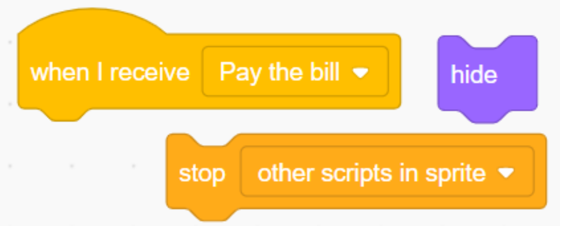
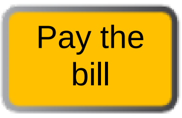
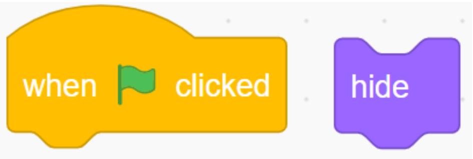
Let's do it

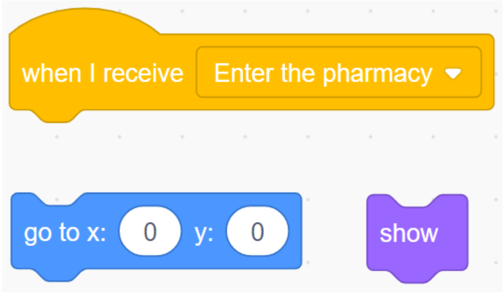
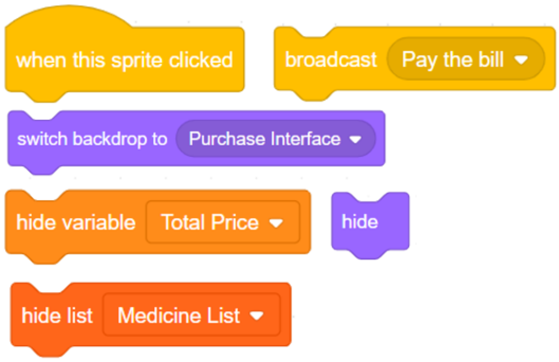
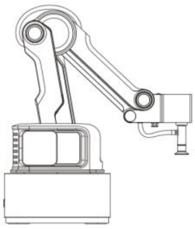
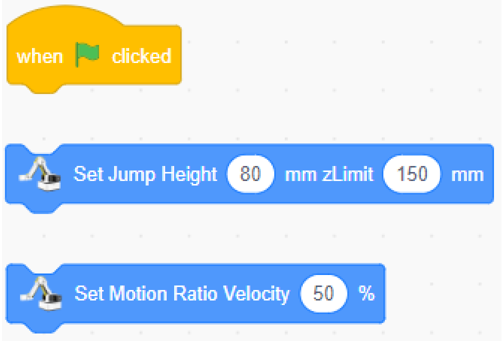
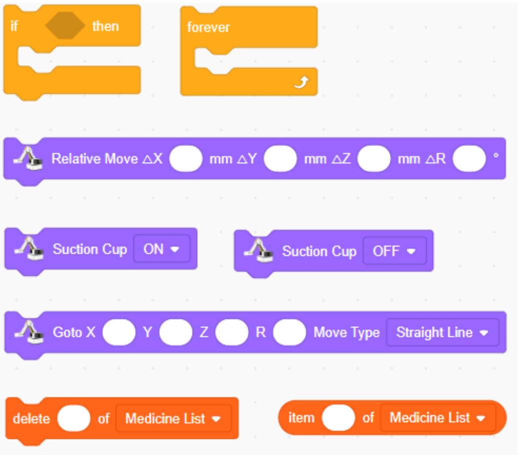
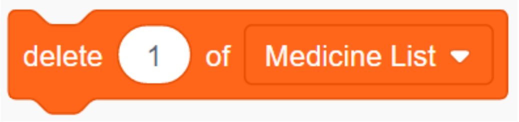
✧ Refer to the work flow chart of "Start Interface", and analyze and draw the work flow chart of "Purchase Interface", "End Interface" and "Auto Medicine Taking".

3. Script planning

Sprite and device	Task description	Block
	1. Initialize Start Interface.	 when 🏁 clicked go to x: ( ) y: ( ) show switch backdrop to [Start Interface ▾] hide variable [Total Price ▾] set [Total Price ▾] to (0) hide list [Medicine List ▾] delete all of [Medicine List ▾]
	2. Switch to Purchase Interface and hide, show variables and a list, and then broadcast "Enter the pharmacy".	 when this sprite clicked hide broadcast [Enter the pharmacy ▾] switch backdrop to [Purchase Interface ▾] show variable [Total Price ▾] show list [Medicine List ▾]
	1. Hide Start Interface.	 when 🏁 clicked hide
	2. When I receive "Enter the pharmacy", show at the designated place.	 when I receive [Enter the pharmacy ▾] go to x: ( ) y: ( ) show

Sprite and device	Task description	Block
	3. When clicked, broadcast "Choose medicine".	 when this sprite clicked broadcast Choose medicine ▾
	4. When I receive "Pay the bill", hide.	 when I receive Pay the bill ▾ hide stop other scripts in sprite ▾
 Delete medicine	1. Hide Start Interface	 when ⚑ clicked hide
	2. Show at the designated place after receiving the message "Enter the pharmacy"	 when I receive Enter the pharmacy ▾ go to x: ( ) y: ( ) show
	3. When clicked, delete the last item of Medicine List.	 when this sprite clicked ( ) = ( ) delete ( ) of Medicine List ▾ set Total Price ▾ to 0 if < > then else

Sprite and device	Task description	Block
	4. When I receive "Pay the bill", hide.	 when I receive Pay the bill ▾ hide
 	1. Hide Start Interface	 when ⚑ clicked hide
	2. Show at the designated place after receiving the message "Enter the pharmacy"	 when I receive Enter the pharmacy ▾ go to x: ( ) y: ( ) show
	3. When I receive "Choose medicine", click the sprite medicine, the sprite medicine will change color effect. Change the Total Price according to the medicine price and add the medicine to Medicine List.	 when I receive Choose medicine ▾ forever if ⟨ ⟩ then touching mouse-pointer ▾ ? mouse down? add ( ) to Medicine List ▾ change Total Price ▾ by ( ) clear graphic effects change color ▾ effect by ( )
	4. When I receive "Pay the bill", hide, and stop calculating the price.	 when I receive Pay the bill ▾ hide stop other scripts in sprite ▾
	1. Hide Start Interface	 when ⚑ clicked hide

Sprite and device	Task description	Block
	2. Show at the designated place after receiving the message "Enter the pharmacy"	 when I receive Enter the pharmacy go to x: 0 y: 0 show
	3. When clicked, broadcast "Pay the bill", switch backdrop to End Interface, and hide sprites, variables and list.	 when this sprite clicked broadcast Pay the bill switch backdrop to Purchase Interface hide variable Total Price hide hide list Medicine List
	1. Initialization	 when 🏁 clicked Set Jump Height 80 mm zLimit 150 mm Set Motion Ratio Velocity 50 %
	2. Identify the name of the first item in "Medicine List", take the corresponding medicine and put it in Laydown Area.	 if then forever Relative Move ΔX mm ΔY mm ΔZ mm ΔR ° Suction Cup ON Suction Cup OFF Goto X Y Z R Move Type Straight Line delete of Medicine List item of Medicine List
	3. Delete the first item in Medicine List	 delete 1 of Medicine List

Processing zone

Design interface backdrop

Design "Backdrop of Start Interface", "Backdrop of Purchase Interface" and "Backdrop of End Interface" respectively, see Figure 6.5. Find the design elements in "*Graphical Programming and Robots* \ Chapter 4" and design three interface backdrops; the design elements are shown in Figure 6.7.

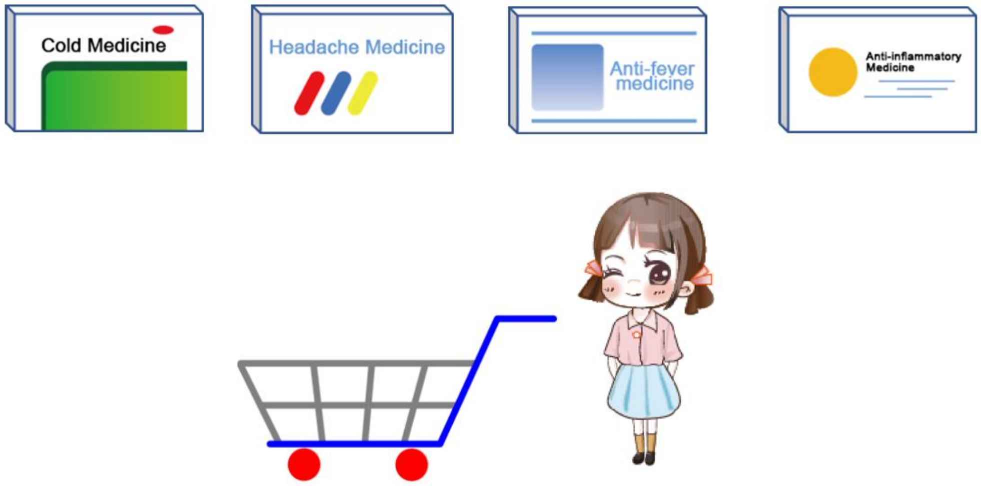


Figure 6.7

Add sprite

Find 8 sprite files of “Enter the Pharmacy", "Cold Medicine", "Anti-inflammatory Medicine", "Anti-fever Medicine", "Headache Medicine", "Choose Medicine", "Delete Medicine" and "Pay the Bill” in "*Graphical Programming and Robots* \ Chapter 4", (See Figure 6.8 for the 8 sprites), import them into Sprite, and delete the default sprite "Sprite 1”.

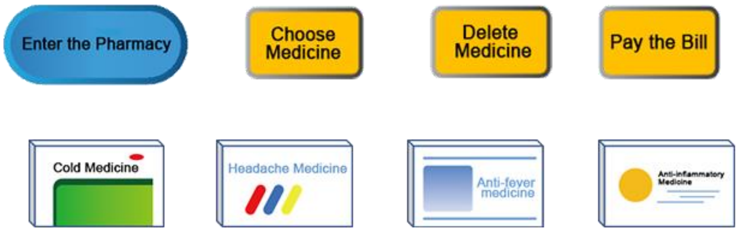
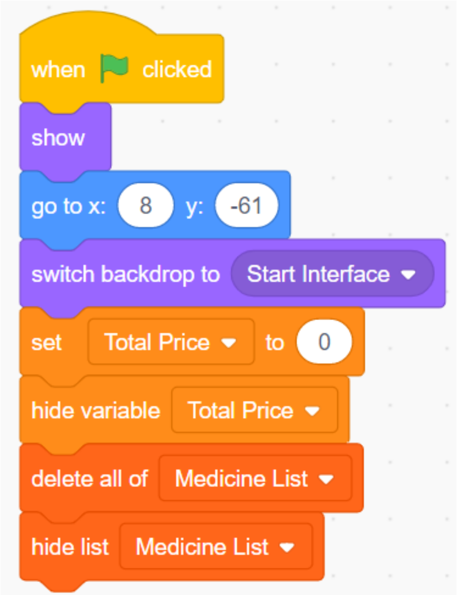


Figure 6.8

Write sprite script

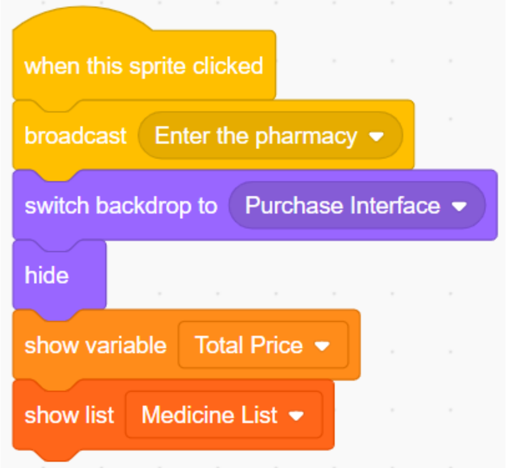
Step 1. Program for all sprites in the Start Interface.

Switch the stage backdrop to the backdrop of “Start Interface", set the sprite "Enter the Pharmacy" to show and the rest sprites to hide. Create and initialize variables "Total Price" and "Medicine List" in the script of sprite "Enter the Pharmacy", see Figure 6.9. When sprite "Enter the Pharmacy" is clicked, enter "Purchase Interface" and hide sprite "Enter the Pharmacy", see Figure 6.10. Now boys and girls, program for rest sprites on "Start Interface" by yourselves according to task analysis and script planning.



```
when ⚑ clicked
show
go to x: 8 y: -61
switch backdrop to Start Interface ▾
set Total Price ▾ to 0
hide variable Total Price ▾
delete all of Medicine List ▾
hide list Medicine List ▾
```

Figure 6.9



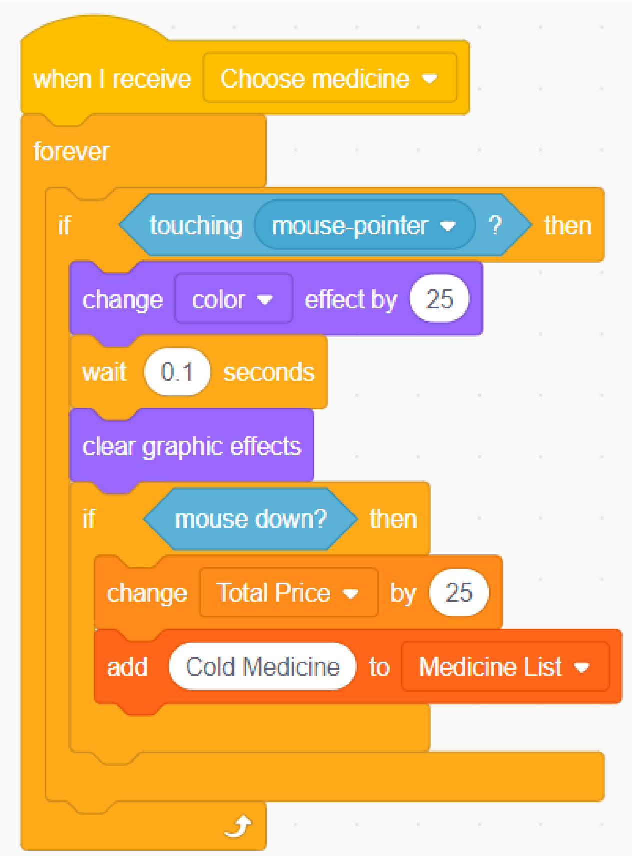
```
when this sprite clicked
broadcast Enter the pharmacy ▾
switch backdrop to Purchase Interface ▾
hide
show variable Total Price ▾
show list Medicine List ▾
```

Figure 6.10

Write sprite script

Step 2. Program for all sprites on "Purchase Interface".

We can see there are sprites of four medicines, "Cold Medicine", "Anti-Inflammatory Medicine", "Anti-fever Medicine" and "Headache Medicine", and sprites "Choose Medicine", "Delete Medicine", "Pay the Bill" as well as variable "Total Price" and "Medicine List" on the "Purchase Interface". Put corresponding medicines in the corresponding position of "Purchase Interface" and show. Click sprite "Choose Medicine" to start choosing medicines. When we click sprites of corresponding medicines, the Total Price and Medicine List will change accordingly. Click sprite "Delete Medicine", then we can delete medicines on the Medicine List. Click and broadcast "Pay the bill", all sprites will hide after receiving the message "Pay the bill", and the backdrop will switch to the "End Interface". Here we take the cold medicine program on "Purchase Interface" as an example, see Figure 6.11.



```
when I receive [Choose medicine ▾]
forever
    if <touching [mouse-pointer ▾] ?> then
        change [color ▾] effect by (25)
        wait (0.1) seconds
        clear graphic effects
        if <mouse down?> then
            change [Total Price ▾] by (25)
            add (Cold Medicine) to [Medicine List ▾]
        end
    end
end
```

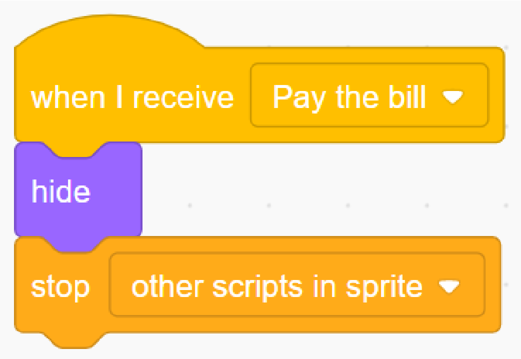
Figure 6.11

Write sprite script

We can program for other medicine sprites with the same ideas. However, it is important to note that different medicines have different prices. After choosing a medicine, the variable "Total Price” will increase by the amount of the medicine, and also the program will add an element the same as the medicine name to "Medicine List". So let's program for other medicines referring to the "Cold Medicine" program.

Step 3. Program for all sprites on "End Interface".

Switch the stage backdrop to the backdrop of "End Interface", hide all sprites, stop other scripts of sprite medicine, and hide variables and list. The program for sprite "Cold Medicine" on "Purchase Interface" is shown in Figure 6.12. Now boys and girls, please program for rest sprites on "End Interface" according to task analysis and script planning.



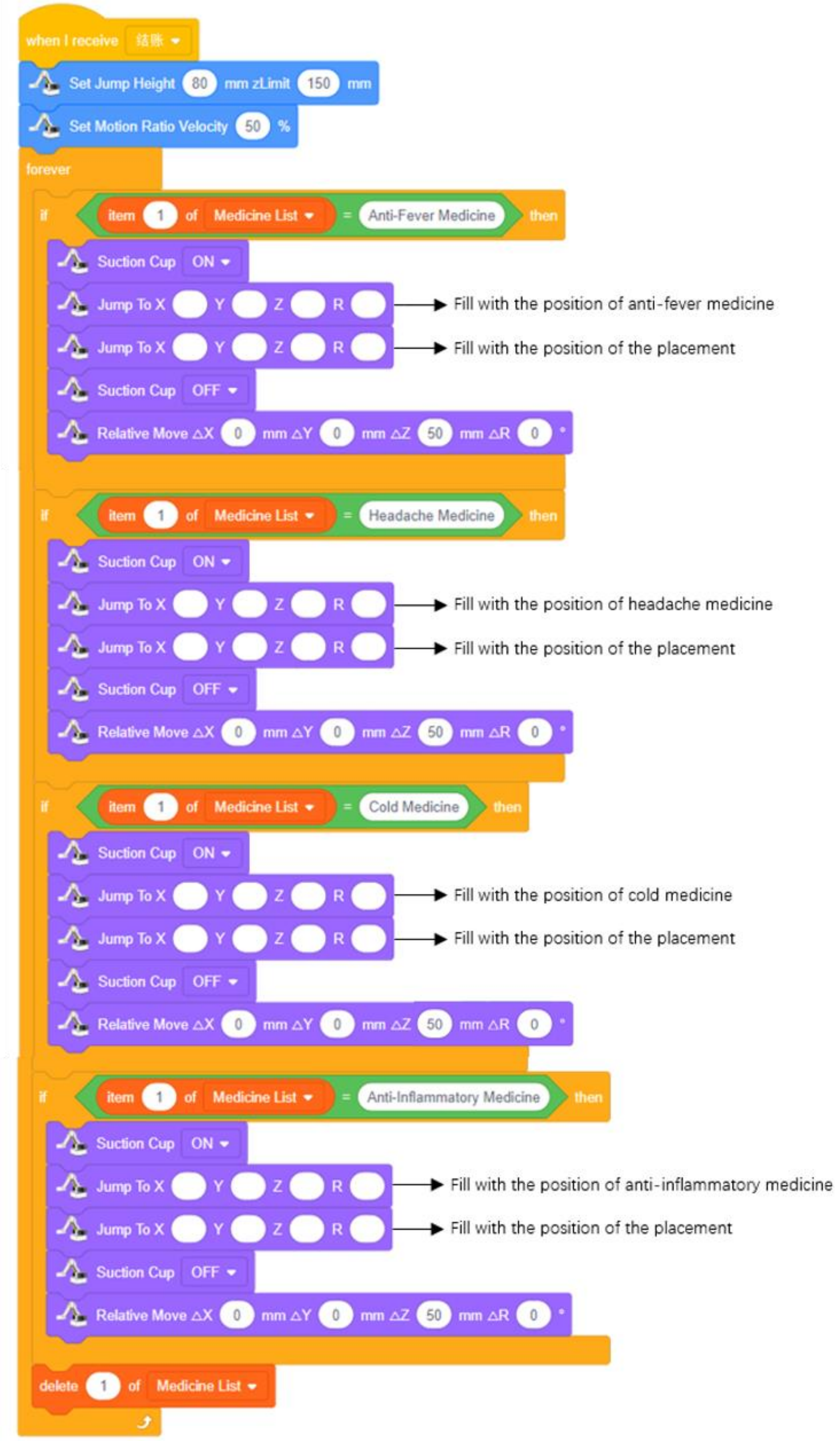
```
when I receive [Pay the bill ▾]
hide
stop [other scripts in sprite ▾]
```

Figure 6.12

Step 4. Program for Auto Medicine Taking.

The key in the program of "Auto Medicine Taking" is to identify each item in the "Medicine List", and then take the medicine according to the "Medicine List", see Figure 6.13 for the program. Boys and girls, lets' debug and improve the program.

Write sprite script



```
when I receive 结账
Set Jump Height 80 mm zLimit 150 mm
Set Motion Ratio Velocity 50 %
forever
    if item 1 of Medicine List = Anti-Fever Medicine then
        Suction Cup ON
        Jump To X ( ) Y ( ) Z ( ) R ( )
        Jump To X ( ) Y ( ) Z ( ) R ( )
        Suction Cup OFF
        Relative Move ΔX 0 mm ΔY 0 mm ΔZ 50 mm ΔR 0 °
    if item 1 of Medicine List = Headache Medicine then
        Suction Cup ON
        Jump To X ( ) Y ( ) Z ( ) R ( )
        Jump To X ( ) Y ( ) Z ( ) R ( )
        Suction Cup OFF
        Relative Move ΔX 0 mm ΔY 0 mm ΔZ 50 mm ΔR 0 °
    if item 1 of Medicine List = Cold Medicine then
        Suction Cup ON
        Jump To X ( ) Y ( ) Z ( ) R ( )
        Jump To X ( ) Y ( ) Z ( ) R ( )
        Suction Cup OFF
        Relative Move ΔX 0 mm ΔY 0 mm ΔZ 50 mm ΔR 0 °
    if item 1 of Medicine List = Anti-Inflammatory Medicine then
        Suction Cup ON
        Jump To X ( ) Y ( ) Z ( ) R ( )
        Jump To X ( ) Y ( ) Z ( ) R ( )
        Suction Cup OFF
        Relative Move ΔX 0 mm ΔY 0 mm ΔZ 50 mm ΔR 0 °
    delete 1 of Medicine List
```

Fill with the position of anti-fever medicine

Fill with the position of the placement

Fill with the position of headache medicine

Fill with the position of the placement

Fill with the position of cold medicine

Fill with the position of the placement

Fill with the position of anti-inflammatory medicine

Fill with the position of the placement

Figure 6.13

Capacity promotion

1. Run the program of "Auto medicine vending system”, discuss about its optimization in groups, record and then fill in Table 6.1.

Table 6.1 Record of Optimization Scheme

No.	Optimization Scheme

2. Discuss about the scheme optimization steps, and record them by flow chart.

3. Write the program for optimization scheme.

Let's share and summarize

Show the group scheme to your teachers and classmates in groups, and share your experience learned from this chapter.

Self-Assessment

Content	Result
I’ve designed the interactive interface of an auto medicine vending system.	☆☆☆☆☆
I’ve realized the communication between computer and robot.	☆☆☆☆☆
I’ve wrote the program "Auto medicine vending system".	☆☆☆☆☆
I’ve optimized the project "Auto medicine vending system".	☆☆☆☆☆