


# TEACHERS' GUIDE

# Table of Contents


**Organizing you work with Scottie Go!** — 7

**Notes for teachers** — 9

Module I - Europe 1 — 11

Module Introduction - commands — 12

Quest 1 - controlling the game character, first commands — 13

Quest 2 — 15

Quest 3 - digits, creating numbers, learning addition and subtraction — 16

Quest 4 - making a turn — 18

Quest 5 — 20

Quest 6 — 21

Quest 7 — 23

Quest 8 - walking backwards — 25

Quest 9 — 27

Quest 10 — 29

Module II - Europe 2 — 31

Module introduction - picking up objects from the game-board — 32

Quest 1 — 33

Quest 2 — 35

Quest 3 — 37

Quest 4 — 39

Quest 5 - jumping — 41

Quest 6 — 44

Quest 7 - note — 46

Quest 8 - note — 49

Quest 9 — 52

Quest 10 — 55

# Table of Contents


Module III - South America — 57

Module Introduction - the REPEAT loop — 58

Quest 1 - the first loop — 59

Quest 2 — 62

Quest 3 - working inside a loop and outside of a loop — 64

Quest 4 - Scottie GO! EDU as a tool for introducing notions in maths and physics — 67

Quest 5 — 70

Quest 6 — 72

Quest 7 - nested loops — 74

Quest 8 — 77

Module IV - North America — 79

Quest 1 - beaver as a bridge — 80

Quest 2 — 82

Quest 3 — 84

Quest 4 - beaver in motion — 86

Quest 5 — 89

Quest 6 - using REPEAT loop and the SWITCH TO tiles together — 92

Quest 7 - the Beaver makes a turn — 95

Quest 8 — 98

Quest 9 — 100

Quest 10 — 102

Module V - Antarctic — 104

Module Introduction - interaction with Scottie's surroundings - ACTIVATE and PLACE tiles — 105

Quest 1 - the ACTIVATE tile — 107

Quest 2 - the PLACE tile — 110

Quest 3 — 113

...

# Table of Contents


Module V - Antarctic ctnd. — 104

...

Quest 4 — 115

Quest 5 — 117

Quest 6 — 120

Quest 7 - SWITCH TO the beavers and move ice floes — 122

Quest 8 — 125

Quest 9 — 127

Quest 10 — 130

**Module VI - Asia** — 132

**Module introduction - conditional loop - REPEAT WHILE** — 133

**Quest 1 - using the REPEAT WHILE tile** — 135

**Quest 2** — 138

**Quest 3** — 141

**Quest 4 - REPEAT WHILE with other conditions** — 144

**Quest 5 - REPEAT WHILE nested in a loop** — 147

**Quest 6** — 150

**Quest 7** — 153

**Quest 8** — 155

**Module VII - Australia 1** — 158

**Module introduction - VARIABLES** — 159

**Quest 1 - building aphalt roads** — 162

**Quest 2** — 165

**Quest 3** — 167

**Quest 4 - X and Y** — 170

**Quest 5** — 173

...

# Table of Contents


**Module VII - Australia 1 ctnd.** — 158

...

**Quest 6 - variables and loops** — 175

**Quest 7** — 178

**Quest 8 - mathematical operations on variables** — 180

**Quest 9** — 183

**Quest 10** — 185

**Module VIII - Australia 2** — 187

**Module introduction - conditional statements - IF, ELSE** — 188

**Quest 1 - action square** — 192

**Quest 2** — 195

**Quest 3** — 197

**Quest 4** — 199

**Quest 5** — 201

**Quest 6 - using multiple conditional statements** — 204

**Quest 7** — 207

**Quest 8 - ELEVATED SQUARE and HERE tiles** — 209

**Module IX - North America 2** — 212

**Module introduction (A) - ELSE IF** — 213

**Module introduction (B) - the ELSE tile** — 216

**Module introduction (C) - [optional] - REPEAT (FOREVER)** — 219

**Quest 1 - ELSE** — 222

**Quest 2 - ELSE IF and MARKS** — 226

**Quest 3** — 231

**Quest 4** — 233

**Quest 5** — 235

...

# Table of Contents

**Module IX - North America 2 ctnd.** — 212

...

**Quest 6** — 238

**Quest 7 - conveyor belts** — 240

**Quest 8 - switches** — 243

**Quest 9** — 246

**Module X - Africa** — 248

...

# Organizing your work with Scottie Go! (1/2)

### Make sure the workplace is well-lit

Playing with Scottie Go! requires your pupils to take photos of the code they wrote. Not enough lighting may make it impossible to scan the code properly.

### Never combine tiles from different kits

Each tile in a kit has its unique ID number and graphics recognized by the camera of your mobile device. If you mix up tiles from several sets in one code, they may not be recognized properly. We recommend that you make a mark on the back of tiles from each kit with a separate colour to distinguish them from tiles that belong to other kits.

**NEVER COMBINE THE HOME VERSION WITH THE EDUCATIONAL VERSION TILES**

### In the optimal model we recommend to use one kit box per 2 pupils

Before you BEGIN using Scottie Go! EDU with your pupils, make sure they know the rules, e.g. you may want to make a contract with your pupils. If you do so, make sure that one of the rules tells your pupils not to lend each other tiles from their kits. When the class finishes, make sure they pack the tiles back into their respective boxes so as not to mix up the tiles from different kits. If you work with younger children, we recommend that you prepare all the required tiles in advance.

### Create a special teacher account that gives you access to all of the quests

You can create a teacher account that will give you access to all quests in all modules. You will be able to access any quest even if you haven't completed the whole Scottie Go! course before. The options accessible through the teacher account allow you to use Scottie Go! while teaching various subjects to cover the programming concepts of your choice, depending on your pupils' age and skill level.

To create the **Teacher** account, choose the following option in the **Scottie Go! Edu** app.

**SELECT PLAYER -> NEW PLAYER** and then enter this code:

## Teacher_13_cps#

**DO NOT REVEAL THIS CODE TO PUPILS!**

# Organizing your work with Scottie Go! (2/2)

The Teacher account allows your pupils to work in teams. In this model you will need only one device to display the Quest description and one mobile device to scan the programs created by your pupils.

The Teacher account allows you to make the best use of Scottie Go! progress cards. To download additional materials, log into your account at:
**www.scottiego.com/education**

# Notes for teachers (1/2)

### There is more than one correct solution

This guide provides you with solutions to each quest. Remember that there are numerous ways to solve the same problems with various programs - some of them may even be shorter than the suggested solution.

### The game comprises 179 tiles that are active in every module and in every quest

You can solve each quest again and again, using the new programming concepts as well as new tiles introduced in subsequent quests and modules.

### Scottie Go! Edu app employs the mechanics of a game, but is a full-fledged educational tool

While using Scottie GO! Edu app, you will be able to both meet your teaching curriculum goals in terms of key coding competencies and tap into the latest world developments in engaging computational thinking in other contexts:

• understanding, analysing and solving problems based on logical and abstract thinking, computational thinking and visual presentation of data,

• programming and problem-solving with the use of computers and other digital devices: assembling and programming algorithms, organising, searching for and sharing information, using computer applications,

• developing social competences, such as: group communication and collaboration, including virtual environments, participation in group projects and project management,

**REMEMBER - OUR GAME IS JUST A TOOL THAT ASSISTS YOUR PUPILS IN LEARNING, NOT A GOAL IN ITSELF!**

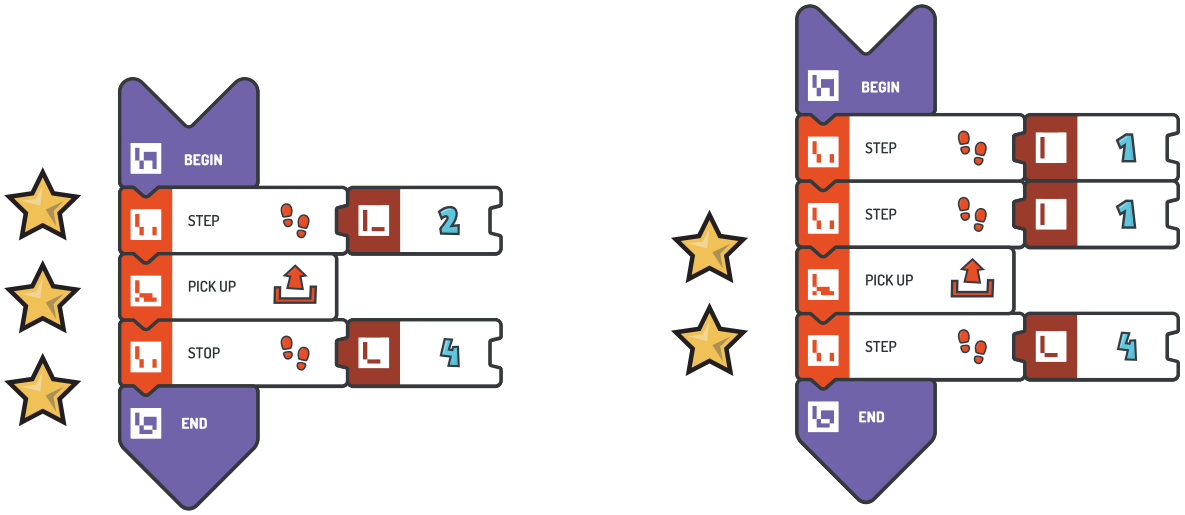
**While using Scottie GO! Edu during your lessons, we recommend that you follow the rules related to active methods, lesson objectives, lesson management as well as evaluation.** Scottie Go! EDU follows the spiral approach to the newly introduced notions, i.e. in one quest introduces a command which is also used in the subsequent quests in different context. The difficulty level of quests increases with each quest as the game progresses, which allows the teacher and his/her pupils to develop their algorythmic skills. Each time a new command is introduced, it is signalled by an exclamation mark in the top right-hand corner of the game character on your screen.

# Notes for teachers (2/2)

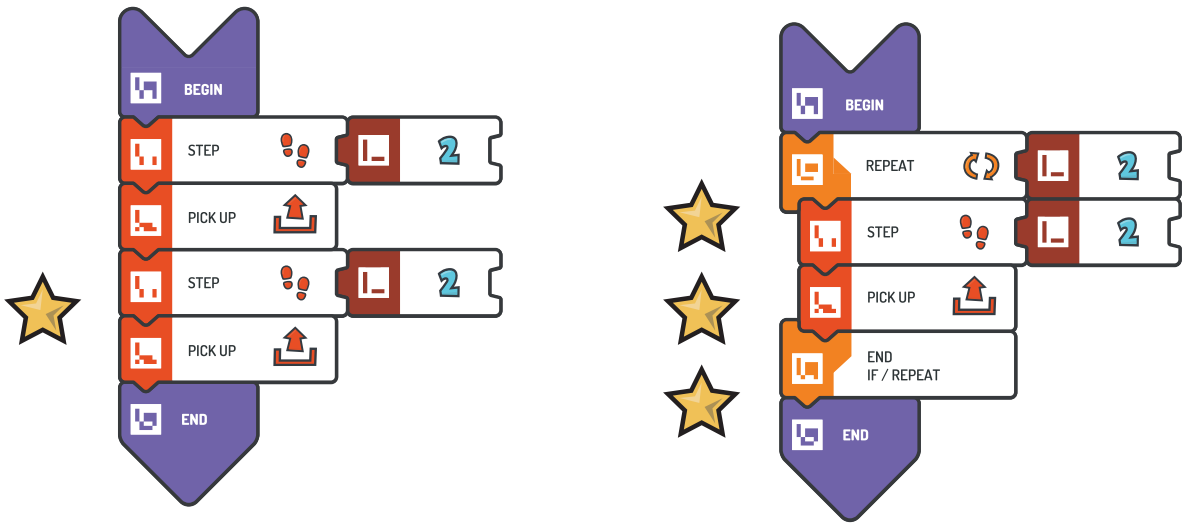
**A good coder and an ambitious pupil should always look for an optimal way to solve a problem. The program is "optimal" when it is constructed using as few lines of code as possible and it still achieves its objective.**

During their lessons teachers should create the atmosphere that facilitates learning. The **ScottieGo! Edu** app features some elements of formative assessment in an attractive game-based environment. After completing each quest, your pupils will receive from 1 to 3 stars as a reward for the quality and optimisation of their codes. It's a lot of fun without making them compete against each other.

Three stars if the program is the best, two stars if the solution is a bit less efficient. One star means that the quest was solved, but the program was too long.




In some quests, if players want to be granted more than one star, they will need to use specific tiles, e.g. the **REPEAT** tile,  in their solution.



# Module VI
# Asia



I need your help. Extremely flexible and strong, bamboo stems will make perfect scaffolding to repair my spaceship. Please, collect all the bamboo stems for me.

# MODULE INTRODUCTION 1/2: Conditional statement - REPEAT WHILE

Similarly to the **REPEAT** loop, the **REPEAT WHILE** loop makes Scottie (or other character) repeat commands within the loop many times, but they will be repeated until a certain condition is met. In this way we can create **conditional loops.**

An everyday example of a **conditional loop**:

Let's imagine that our pupils' parents are driving them home from school. Sometimes it's hard to definitely determine which steps should be performed and how many times should they be performed in order to get home. In most cases like that we subconsciously use the **REPEAT WHILE** loop. How so?.

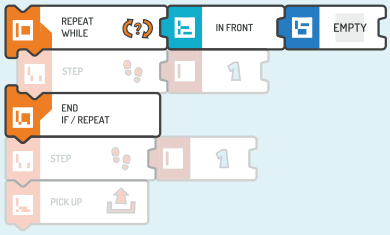
1. We check the condition **"is there a straight road ahead of me?"**
2. If this condition is met, we go straight and go back to point (1) again – i.e. we start checking again if the road ahead of us is straight.
3. If the condition is not met, e.g. there is a turn ahead of us, the loop is broken. We no longer check for condition from point (1) and we start performing the next command outside of the loop - in this case we turn in the correct direction.

### Conditional loops in Scottie Go!

Let's look at how conditional loop is used in this example. In this quest Scottie is only allowed to use the combination of tiles **STEP** and the number **1** tile. It will make it easier to explain how this loop is used.

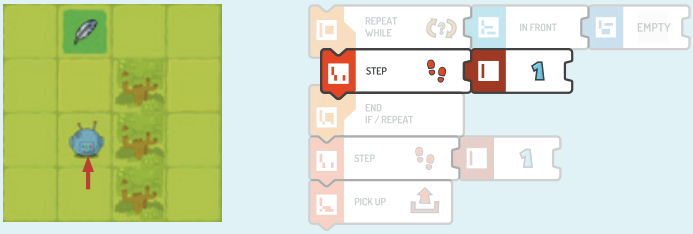
1. We check the condition **"is there a straight road ahead of me?"**. The condition is met, i.e. the square in front of Scottie is empty.



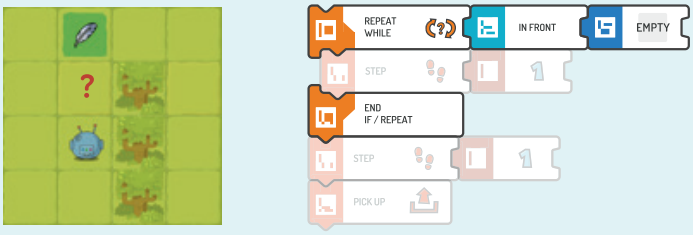


# MODULE INTRODUCTION 2/2: Conditional statement - REPEAT WHILE

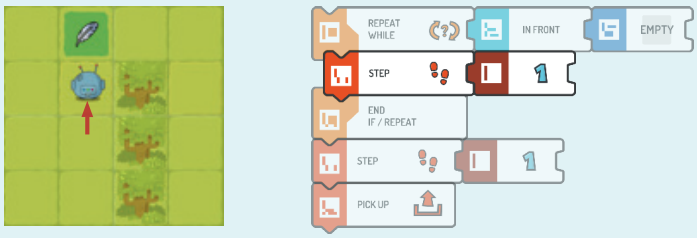
2. Scottie begins performing the commands located inside the loop. **He moves one step forward.** The program goes back to checking the condition again (point 1.)



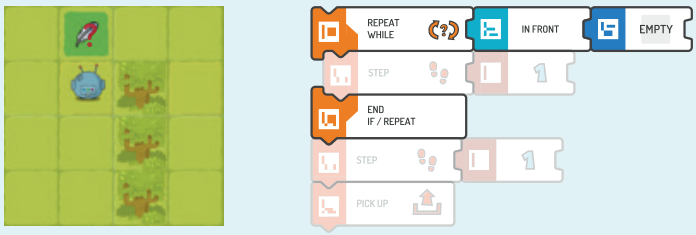

3. Scottie checks the condition ”free road ahead” again. **The condition is met.**




4. Scottie performs the command inside the loop, i.e. - **STEP 1**, again, and then he goes back to checking the condition in point 1.




5. Scottie checks the condition ”free road ahead” again. **This time this condition is not met.**



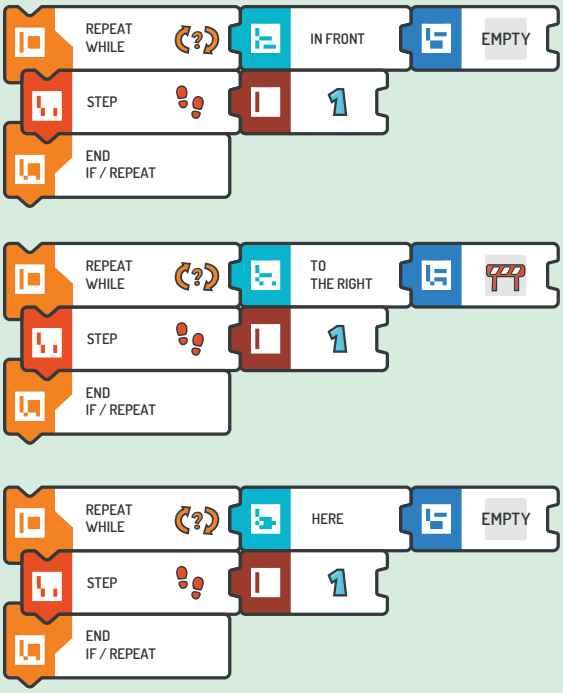

6. **The loop is broken.** Scottie begins to follow the program outside of the loop: **STEP (1)** and **PICK UP.**



# INTRODUCTION:
# using the REPEAT WHILE tile

In the quests ahead of you use only the combination of the **STEP (1)** tiles and the **REPEAT WHILE** loop.

Sometimes instead of the **REPEAT WHILE (IN FRONT < EMPTY)** tile combination you can use the **REPEAT WHILE (ON THE RIGHT < OBSTACLE)** or **REPEAT WHILE (HERE < EMPTY)**




In the last example you can potentially shorten the program by one line.

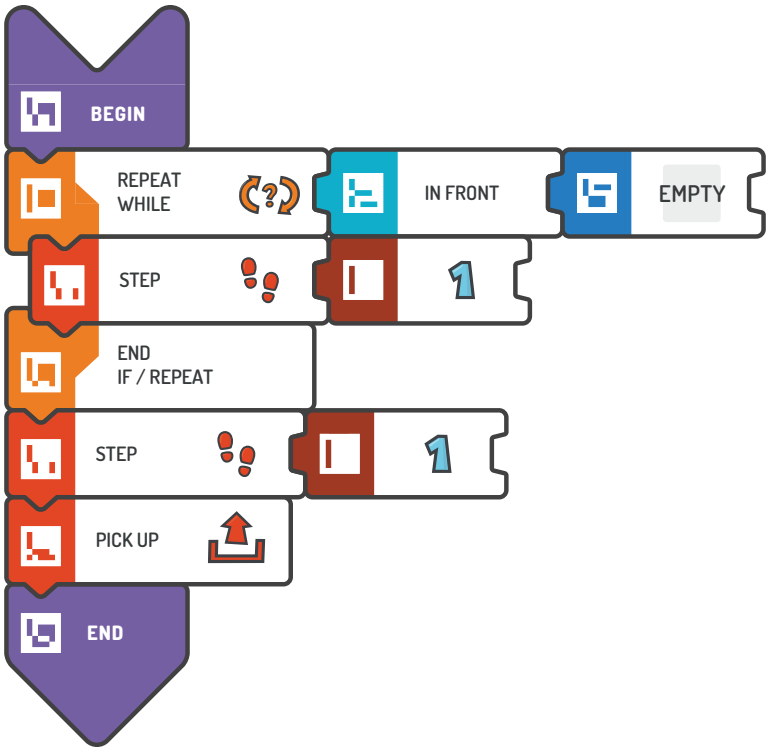
**While introducing the REPEAT WHILE loop, be sure to use the ”real life” example cited in the module introduction or think about your own example.**

Being consistent in using the combination of the **STEP (1)** tiles with the **REPEAT WHILE** tile in numerous subsequent quests will help make the loop structure feel natural to your pupils. Remeber to require your pupils to use only **STEP (1)** tiles.

Challenge:




Example solution:



Challenge:




Example solution (in text format):

```
BEGIN
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
    }
    STEP (1)
    PICK UP
END
```

# HINT

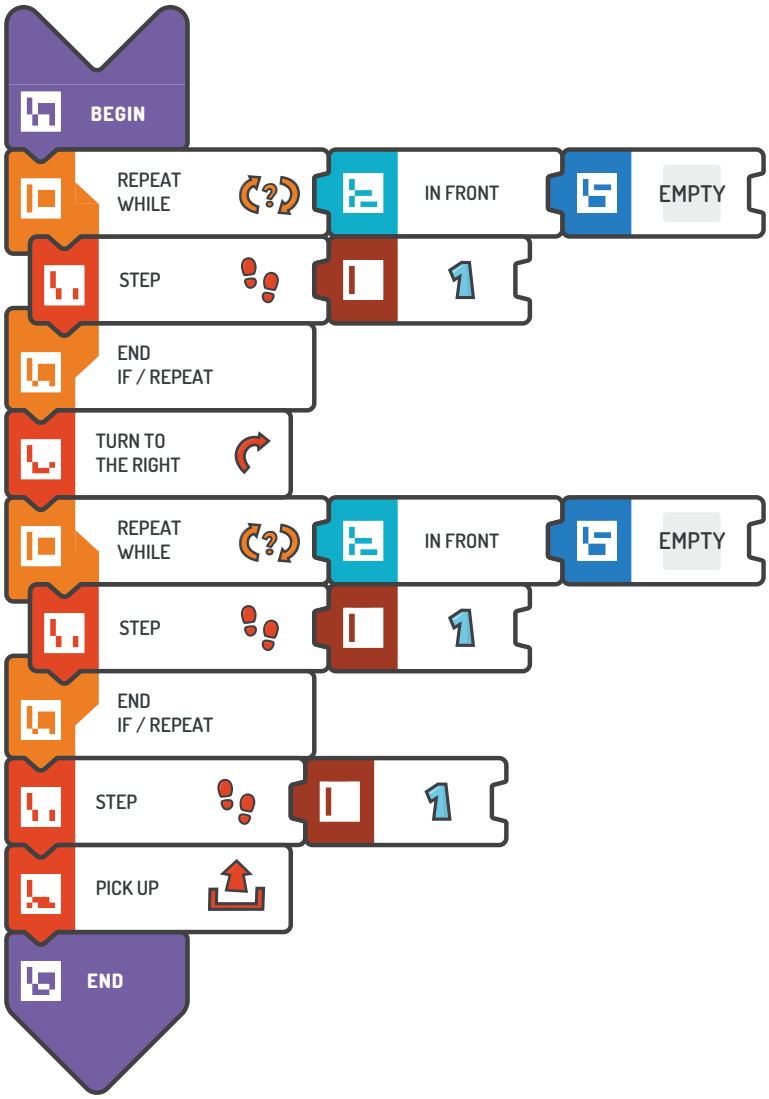
Use two **REPEAT WHILE** loops in this quest.

Optionally, instead of using the **REPEAT WHILE (IN FRONT < EMPTY)** combination in the second loop in this quest, your pupils can use the **REPEAT WHILE (HERE < EMPTY)** combination. After some alterations the program should become one line shorter.

Challenge:




Example solution:



Challenge:




Example solution (in text format):

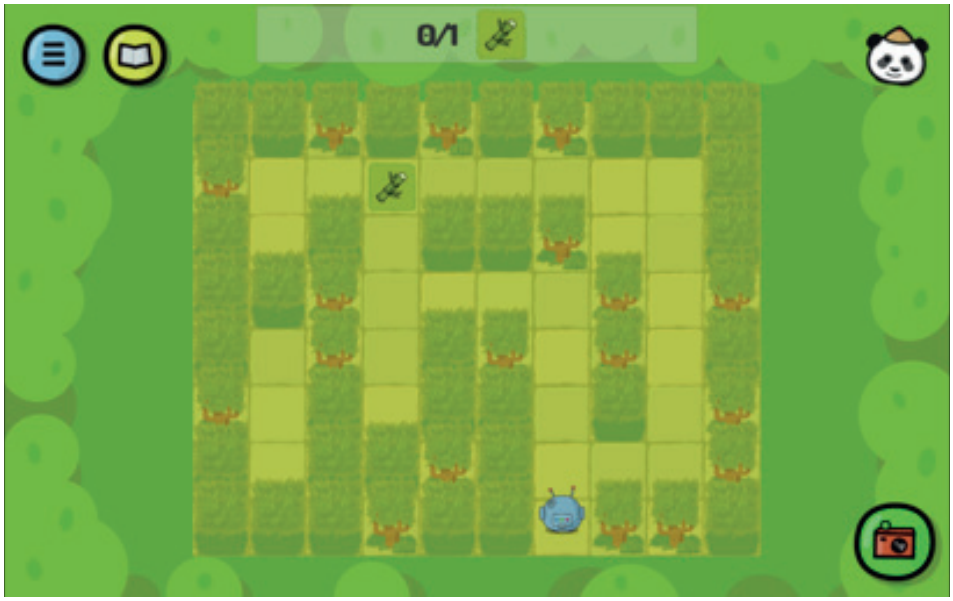
```
BEGIN
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
    }
    TURN TO THE RIGHT
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
    }
    STEP (1)
    PICK UP
END
```

# HINT

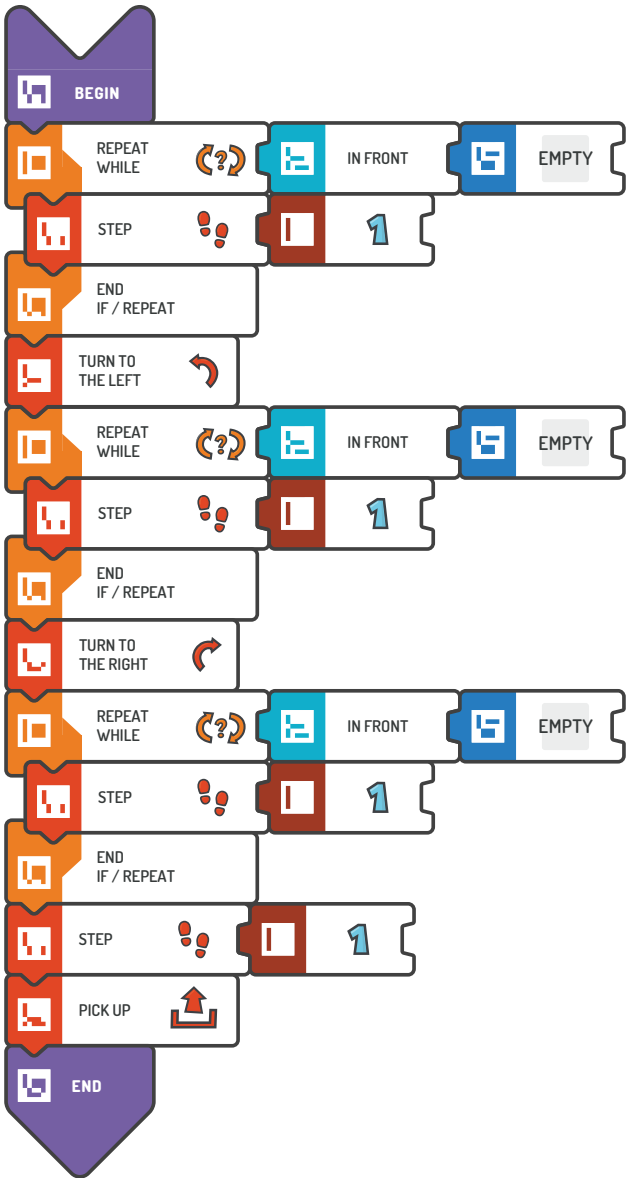
As we could see in the previous quests, the **REPEAT WHILE** loops is a smarter version of the **REPEAT** loop, as the sequence of commands within the loop is performed as long as the condition (if statement) is met. When the condition is no longer true, the loop is broken.

Use three **REPEAT WHILE** loops in this quest.

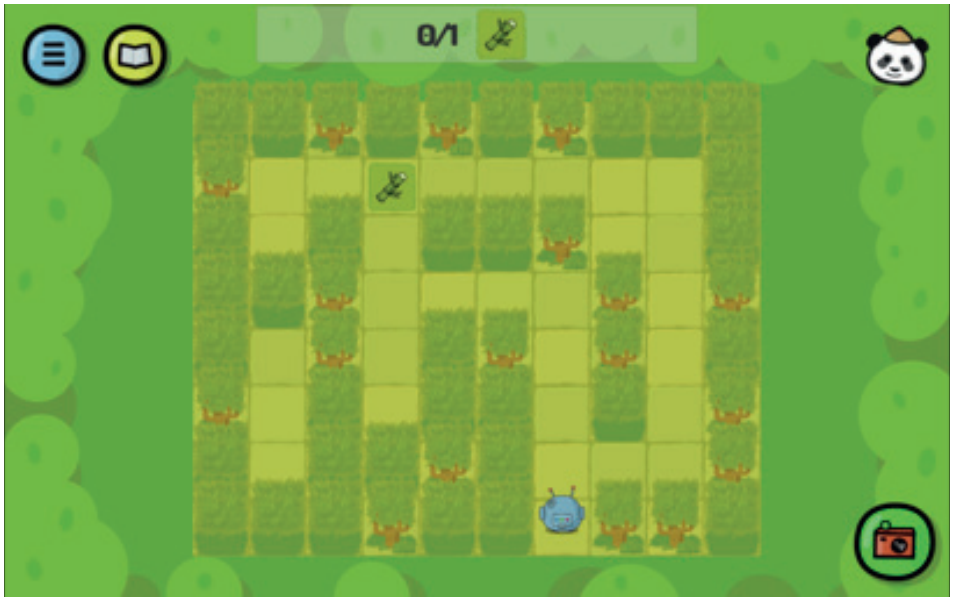
Challenge:




Example solution:



Challenge:



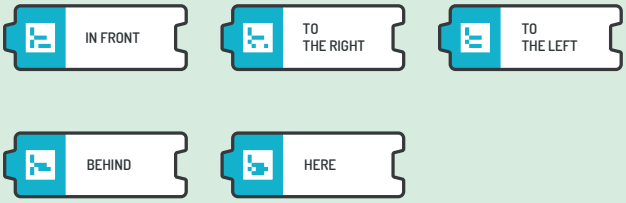

Example solution (in text format):

```
BEGIN
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
    }
    TURN TO THE LEFT
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
    }
    TURN TO THE RIGHT
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
    }
    STEP (1)
    PICK UP
END
```

# INTRODUCTION:
# REPEAT WHILE with other conditions

You can build **REPEAT WHILE** loops with a number of conditions in order for the commands within the loop to be executed.

### Where (considering Scottie's point of view)?




### What?




**EMPTY**

**COLLECTABLE OBJECT**

**OBSTACLE**

**ELEVATED SQUARE (crucial in later modules)**



**MARK - SQUARE (crucial in later modules)**

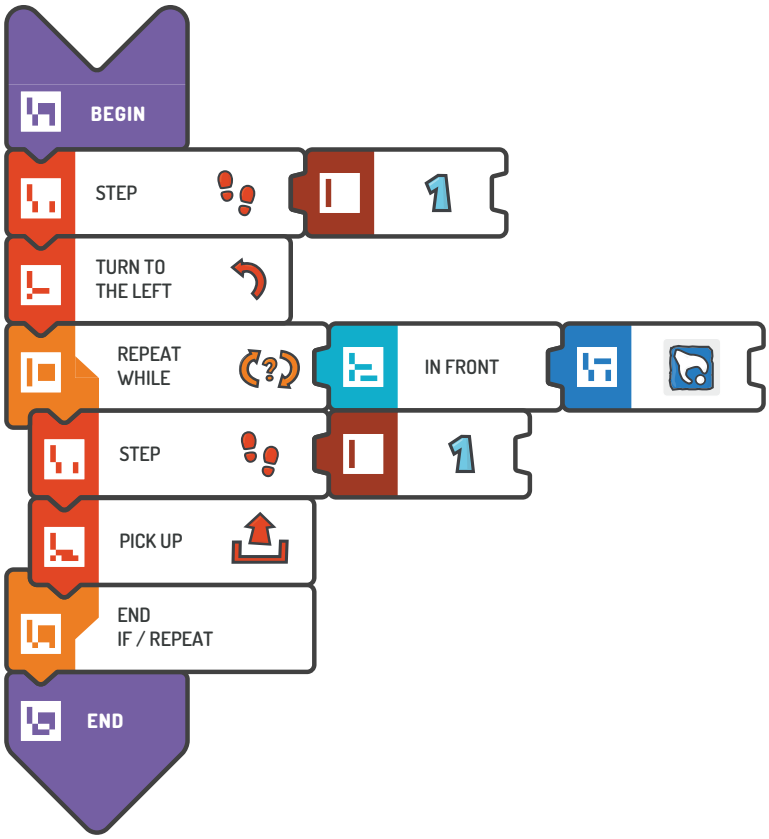
**MARK - CROSS (crucial in later modules)**

**ACTION SQUARE (crucial in later modules)**

Challenge:




Example solution:



Challenge:




Example solution (in text format):

```
BEGIN
    STEP (1)
    TURN TO THE LEFT
    REPEAT WHILE (IN FRONT < COLLECTABLE OBJECT) {
        STEP (1)
        PICK UP
    }
END
```

# INTRODUCTION:
# REPEAT WHILE nested in a loop

If you need, you can use the **REPEAT WHILE** loops multiple times. Create a **REPEAT** loop and then place another **REPEAT WHILE** loop inside it. A loop inside another loop - try it, it's not that hard!

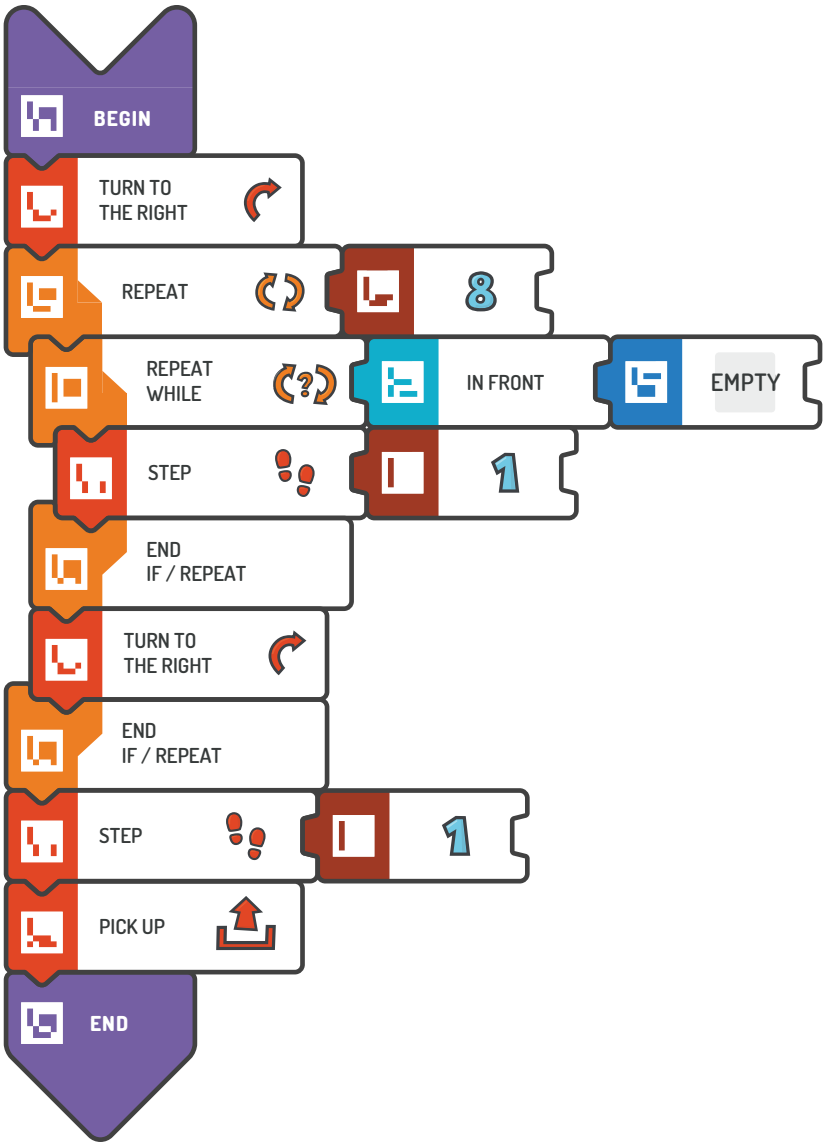
More advanced pupils may try to solve this quest using conditional statements like this:

```
IF (ON THE RIGHT < EMPTY) {
        TURN TO THE RIGHT
}
```

Challenge:




Example solution:



Challenge:



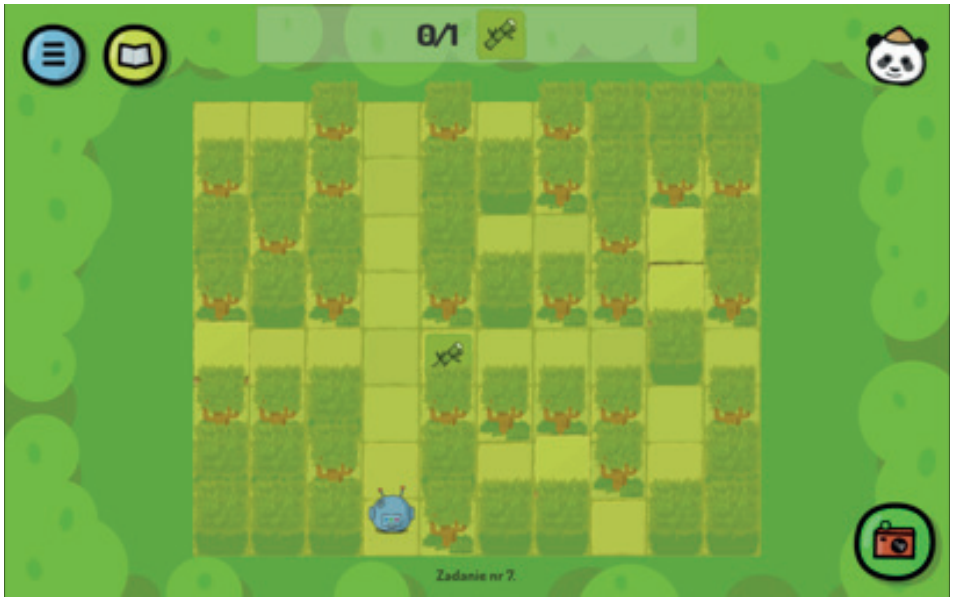

Example solution (in text format):

```
BEGIN
    TURN TO THE RIGHT
    REPEAT (8) {
        REPEAT WHILE (IN FRONT < EMPTY)  {
            STEP (1)
        }
        TURN TO THE RIGHT
    }
    STEP (1)
    PICK UP
END
```

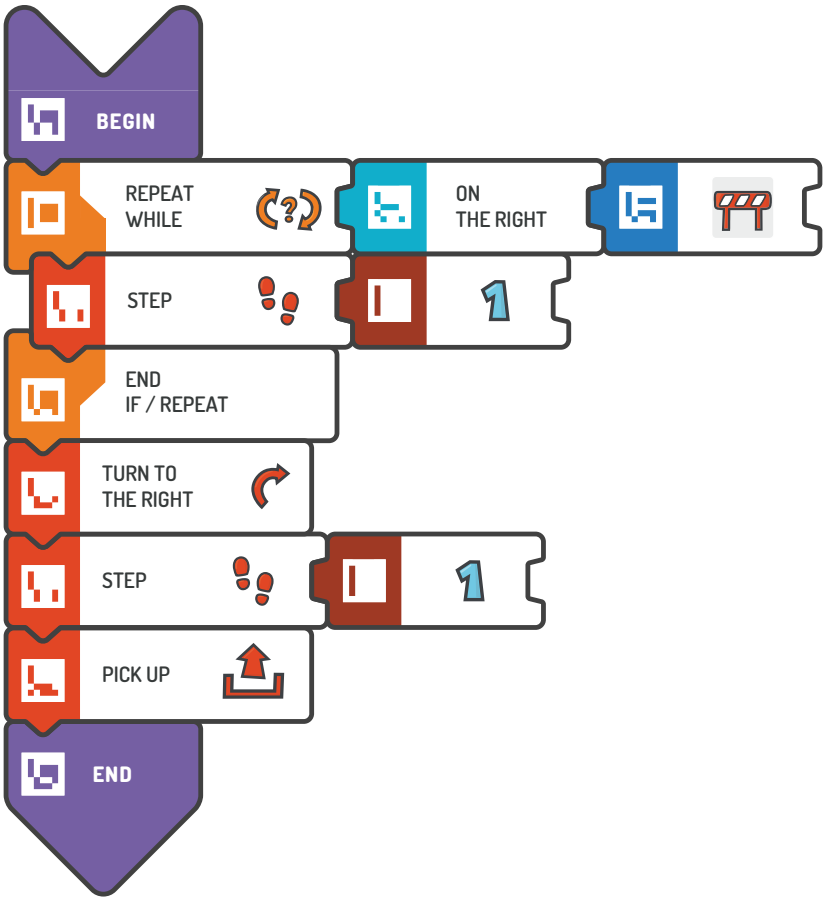
# NOTE!

Use the **REPEAT WHILE** loop only once.

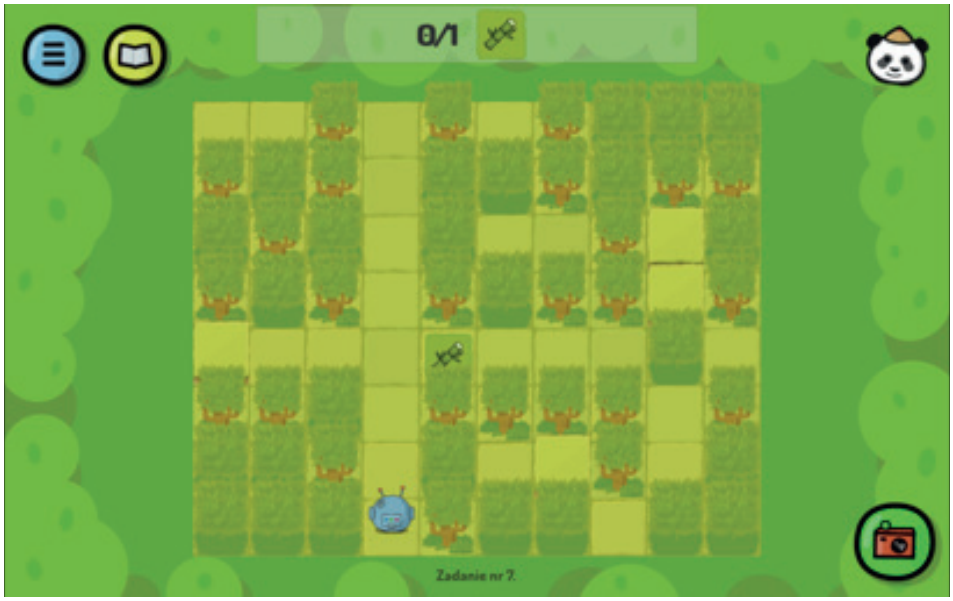
Challenge:




Example solution:



Challenge:



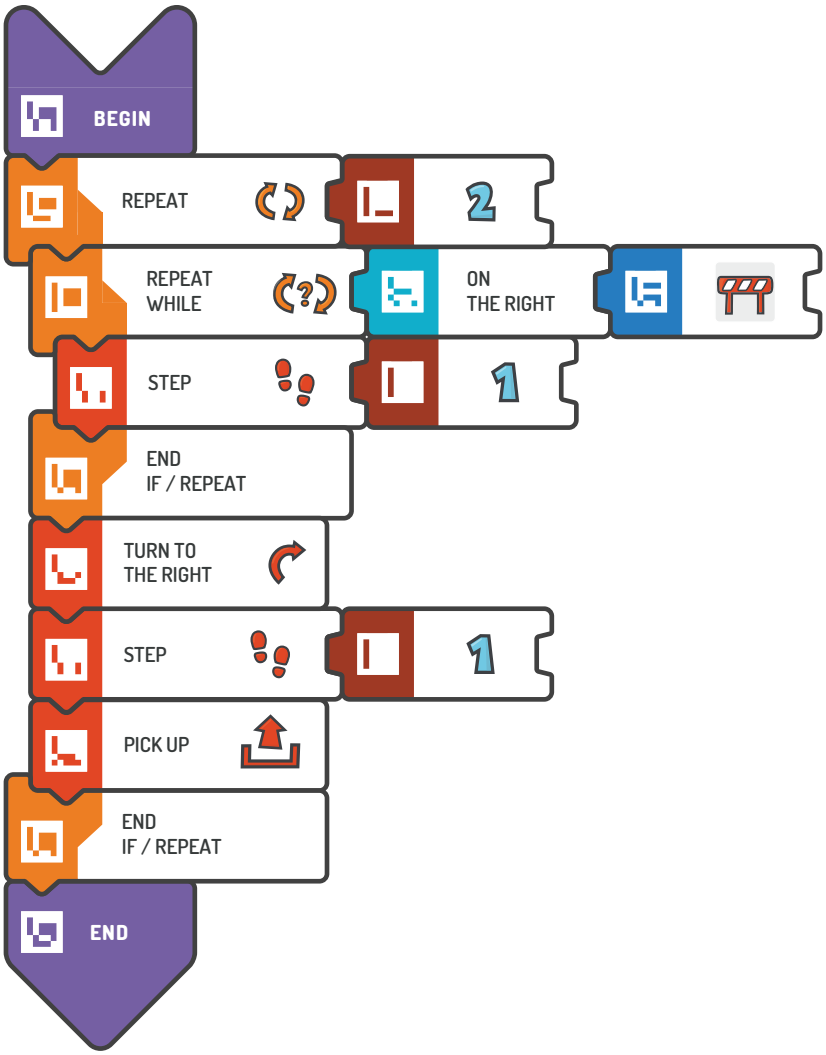

Example solution (in text format):

```
BEGIN
    REPEAT WHILE (ON THE RIGHT < OBSTACLE) {
        STEP (1)
    }
    TURN TO THE RIGHT
    STEP (1)
    PICK UP
END
```

Challenge:




Example solution:



Challenge:




Example solution (in text format):

```
BEGIN
    REPEAT (2) {
        REPEAT WHILE (ON THE RIGHT < OBSTACLE) {
            STEP (1)
        }
        TURN TO THE RIGHT
        STEP (1)
        PICK UP
    }
END
```

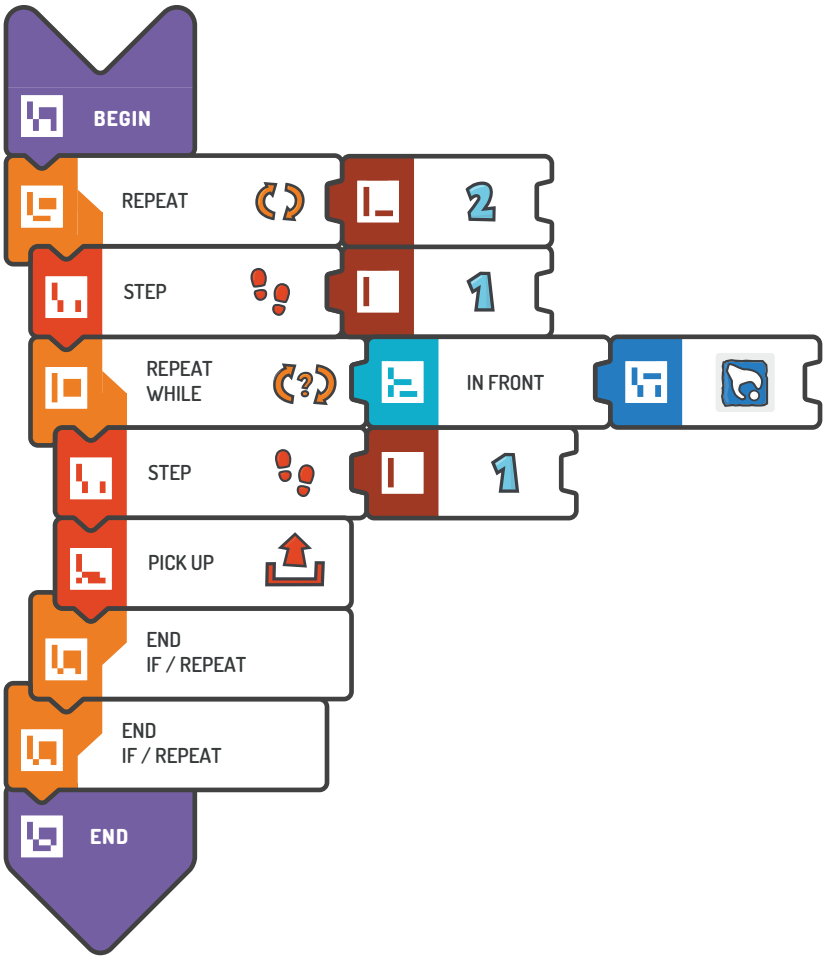
# NOTE!

Use both the **REPEAT WHILE** loop and the **REPEAT** loop only once in this quest.

Challenge:




Example solution:



Challenge:




Example solution (in text format):

```
BEGIN
    REPEAT (2) {
        STEP (1)
        REPEAT WHILE (IN FRONT < COLLECTABLE OBJECT) {
            STEP (1)
            PICK UP
        }
    }
END
```

# Module VII
# Australia 1



In this module the notion of a VARIABLE is introduced.

Build asphalt paths for Scottie so that he can move around the village and collect parts for his spaceship.

# MODULE INTRODUCTION 1/3: new concept: VARIABLES

You can imagine that **variables are containers**. You can place various data into such a container, e.g. numbers, and then store them in order to be used later. While running, a computer program may look into those containers to check and read their current contents or to change their contents. In order for the program to know where to look (it's easy to imagine that there are more than one variable in a program), variables need to have their names.

You can use a variable, e.g. to determine the number of steps that Scottie should take. Let's suppose that in the quest presented below Scottie will repeatedly perform two instructions - two steps forward and **TURN TO THE LEFT**.

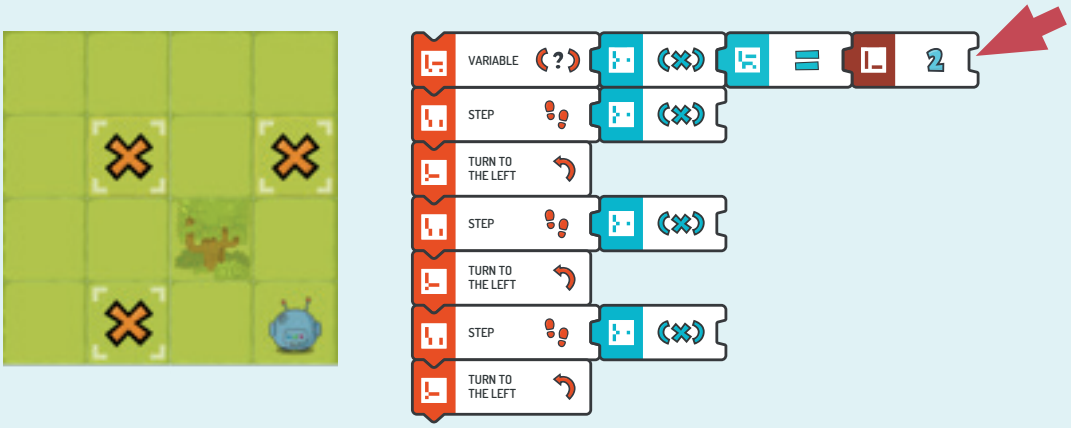


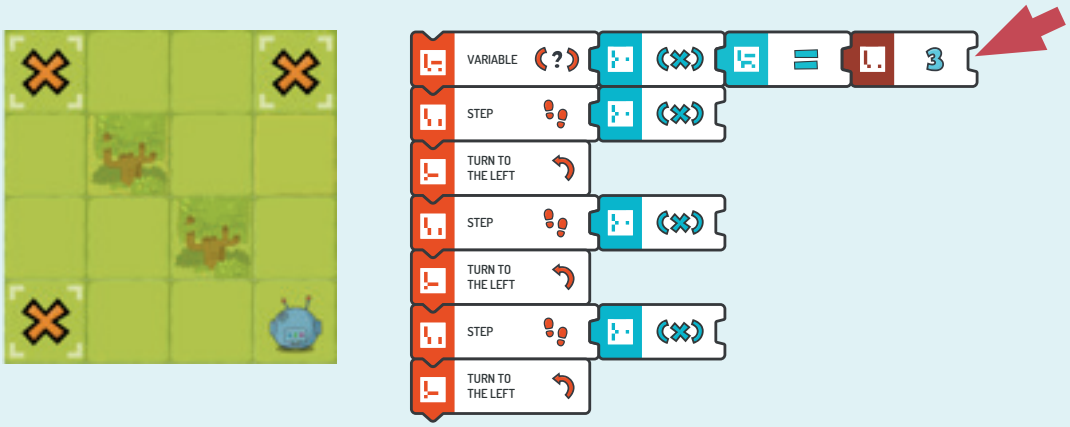

In the next quest (see the image below) his moves are almost identical, but the only thing that changes is the number of steps (three steps and **TURN TO THE LEFT**). It's evident that you need to change the program presented before as many as three times in order for it to work in this quest.



# MODULE INTRODUCTION 2/3: new concept: VARIABLES

Let's get back to the first quest presented as an example (see the image below), but this time let's try to simplify the solution by using variables. With each step, instead of using the number, let's use a variable. In the first part of the program let's define this variable using the equality sign **'='** and let's define its value as 2 (the number of steps).




To solve the next quest when the variable is used, it's enough to introduce a change in the first line of code, where a value is defined for the variable **X**.

As you can see, variables can often shorten the time needed to modify the program.

**To sum up: thanks to variables the program is neater and much easier to modify.**

# MODULE INTRODUCTION 3/3: new concept: VARIABLES

**To see how variables work, you need to perform the following steps:**

1. Create a variable by giving it a name and defining which value it originally holds.



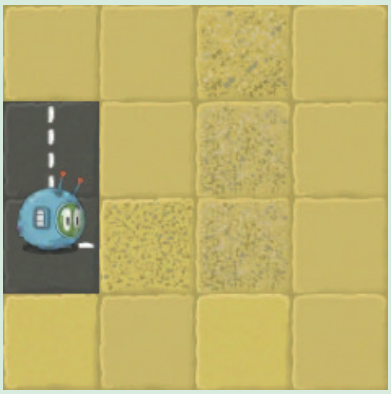

2. Use this variable in the program by calling its name instead of its value.



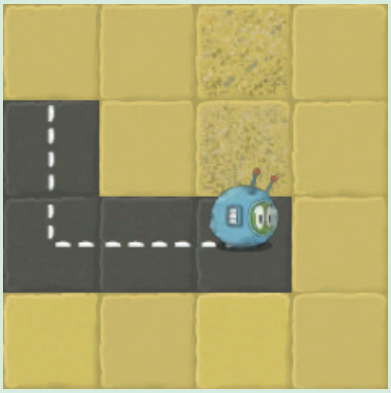
# INTRODUCTION - building asphalt roads

**Walking on gravel, Scottie will build an asphalt road.**

Remember to create the variable **X =** number of steps and then call it in the program.



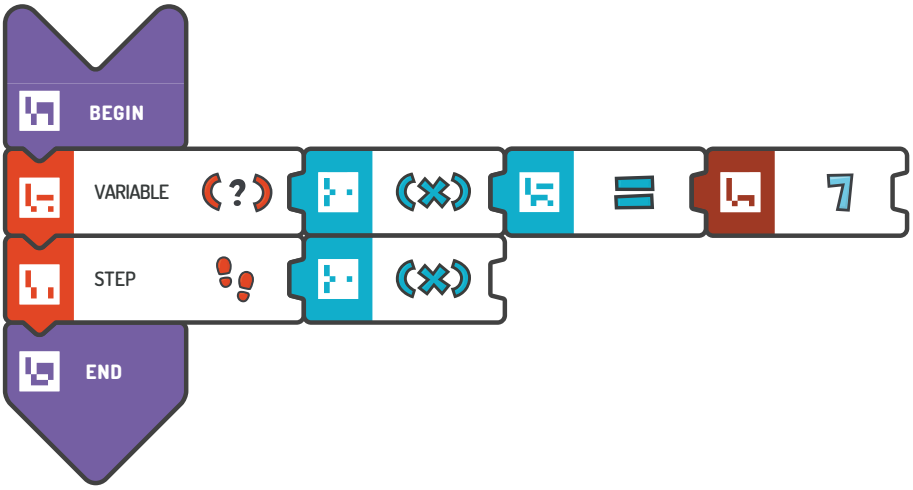


Challenge:




Example solution:



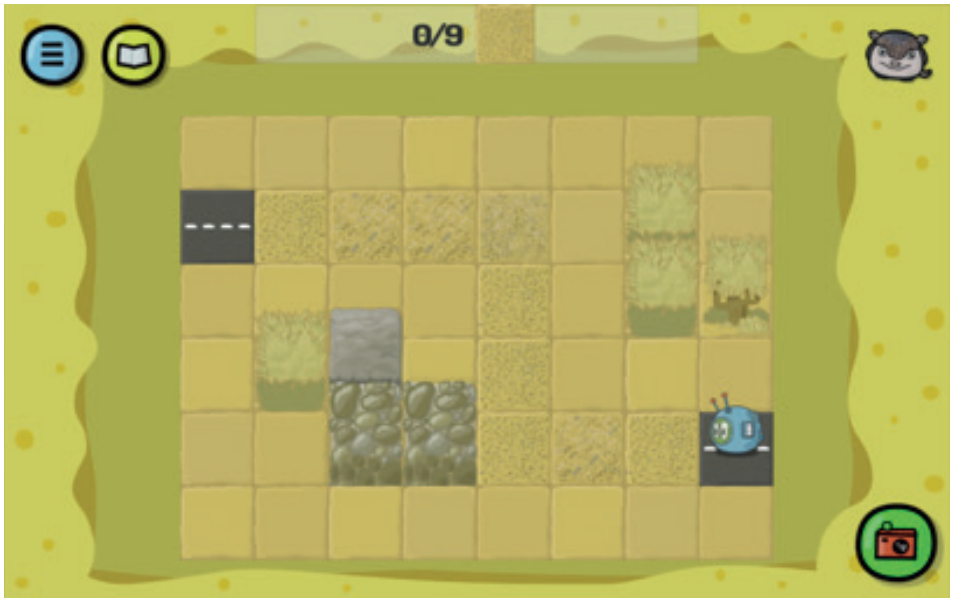
Challenge:



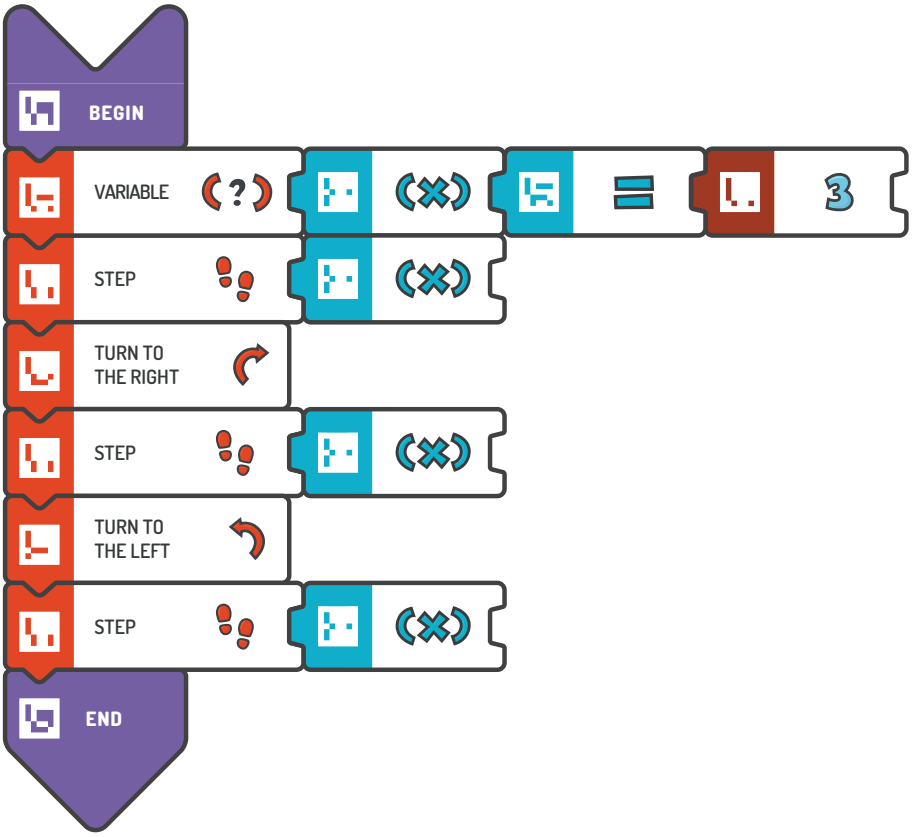

Example solution (in text format):

```
BEGIN
    VARIABLE X=7
    STEP (X)
END
```

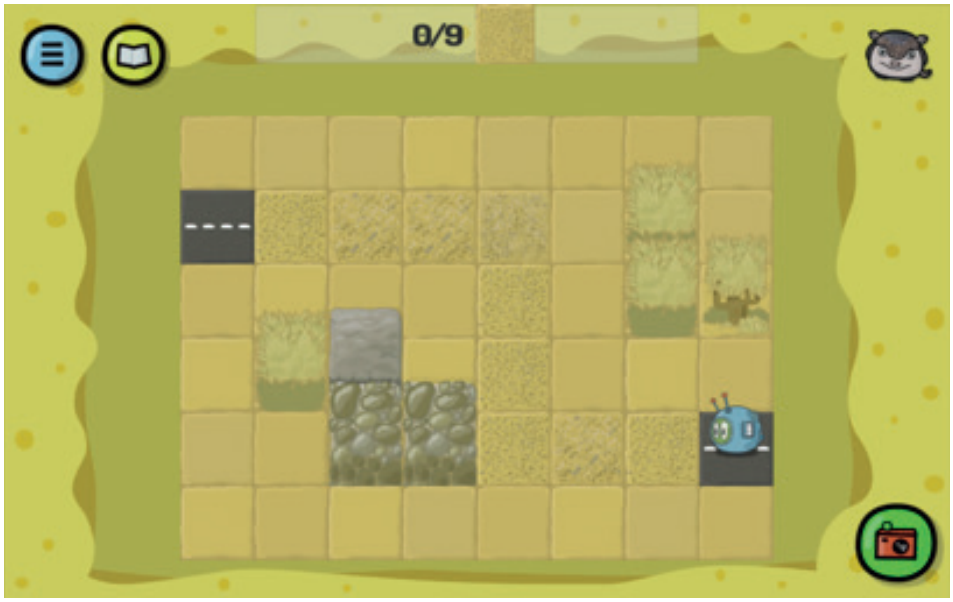
Challenge:




Example solution:



Challenge:




Example solution (in text format):

```
BEGIN
    VARIABLE X=3
    STEP (X)
    TURN TO THE RIGHT
    STEP (X)
    TURN TO THE LEFT
    STEP (X)
END
```

# NOTE!

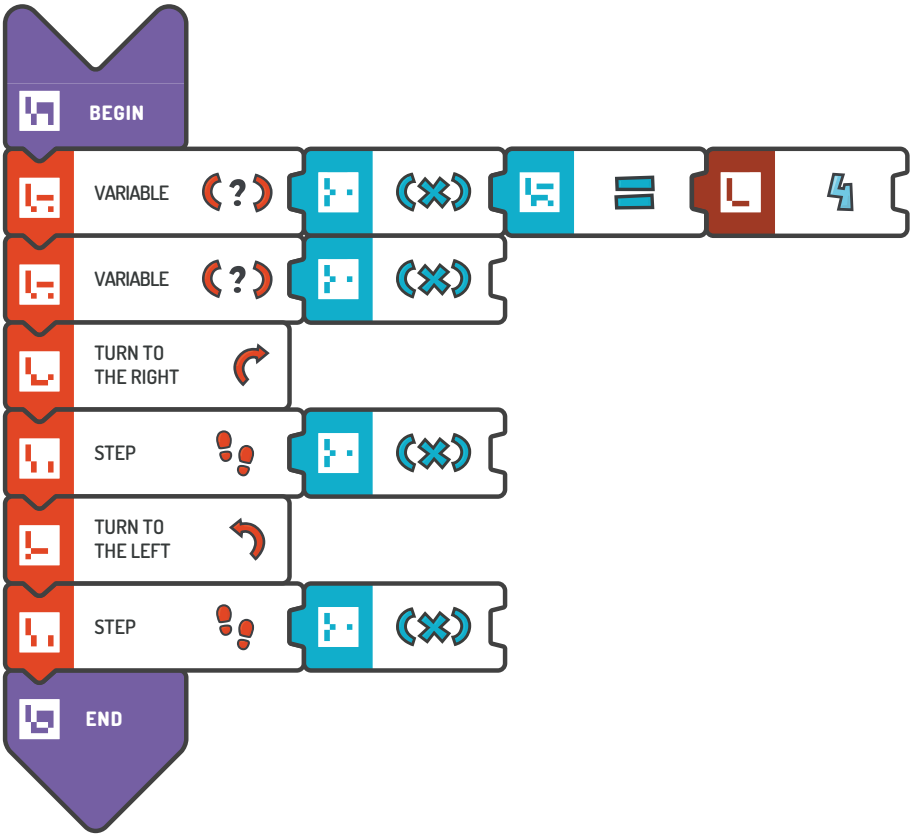
Your previous program can solve this quest too.

**You just need to modify the value of the variable X.**

Challenge:




Example solution:



Challenge:




Example solution (in text format):

```
BEGIN
    VARIABLE X=4
    STEP (X)
    TURN TO THE RIGHT
    STEP (X)
    TURN TO THE LEFT
    STEP (X)
END
```

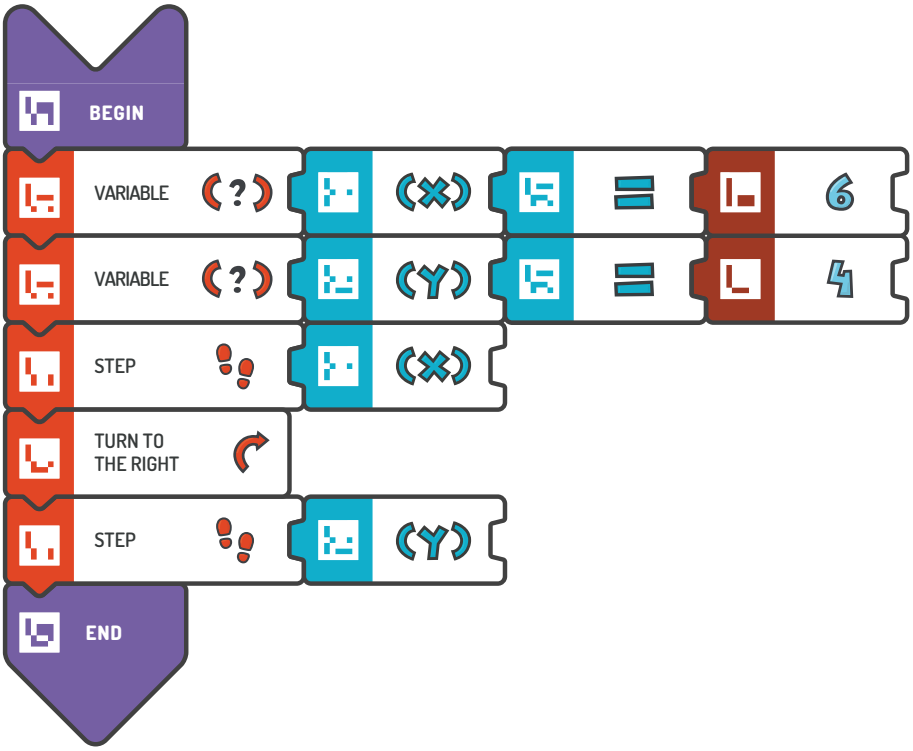
# INTRODUCTION:
# X and Y

In Scottie Go! You can create and use two variables: **X** and **Y** in the same program.

Challenge:




Example solution:



Challenge:



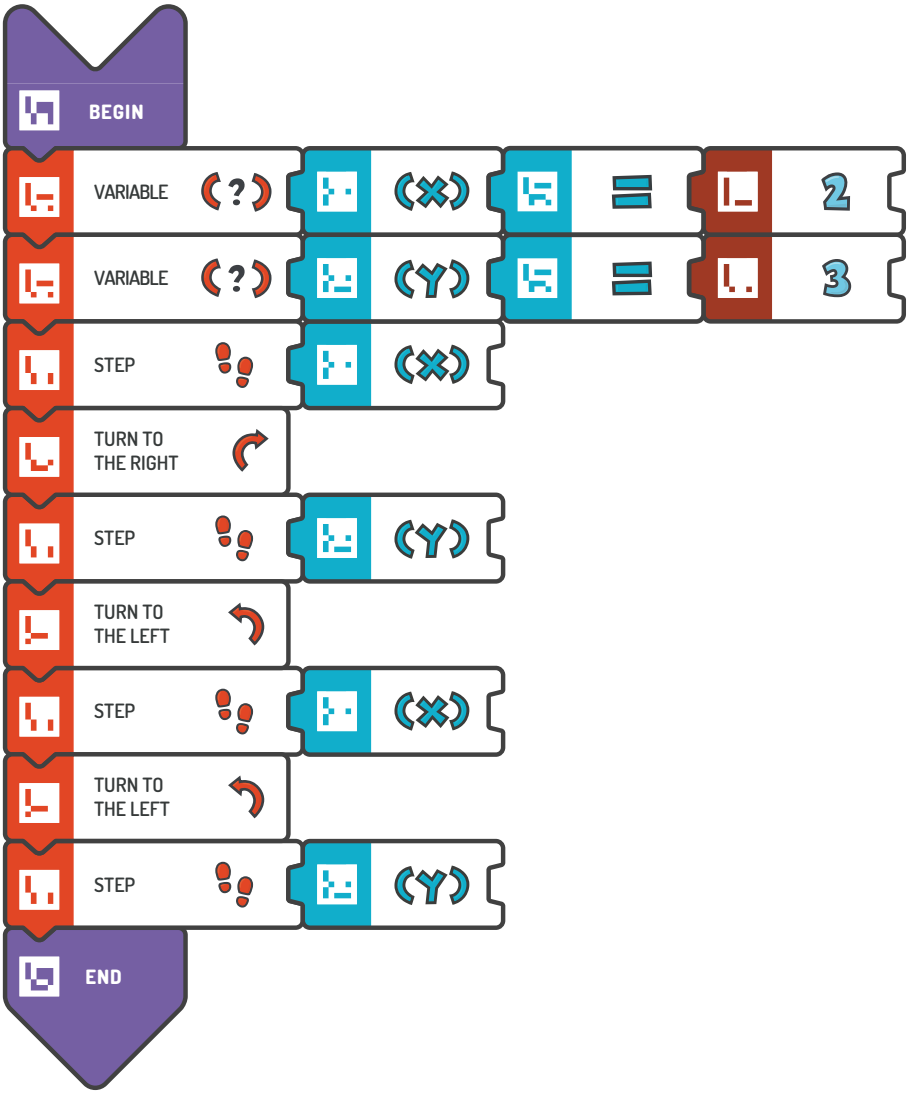

Example solution (in text format):

```
BEGIN
    VARIABLE X=6
    VARIABLE Y=4
    STEP (X)
    TURN TO THE RIGHT
    STEP (Y)
END
```

Challenge:




Example solution:



Challenge:



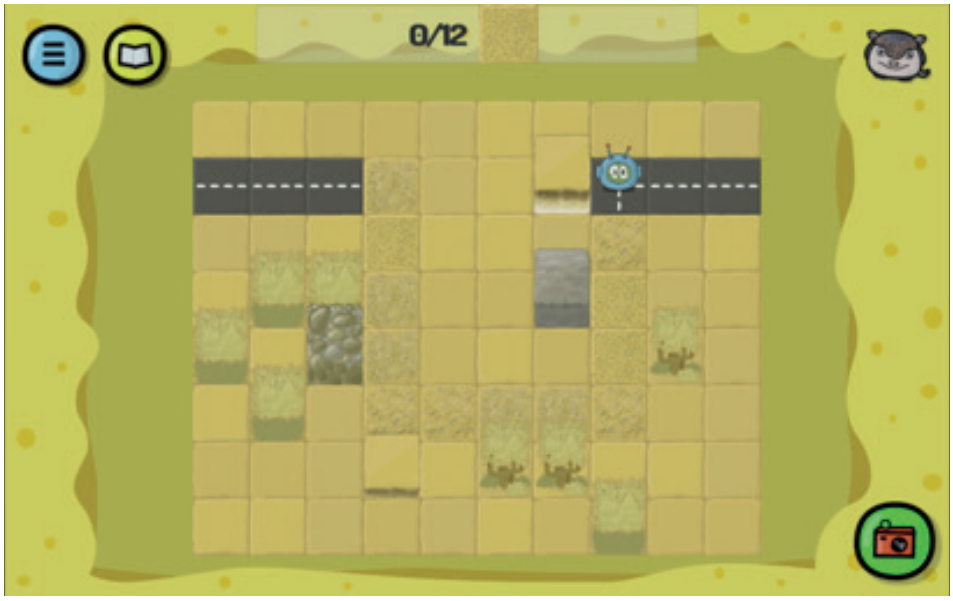

Example solution (in text format):

```
BEGIN
    VARIABLE X=2
    VARIABLE Y=3
    STEP (X)
    TURN TO THE RIGHT
    STEP (Y)
    TURN TO THE LEFT
    STEP (X)
    TURN TO THE LEFT
    STEP (Y)
END
```

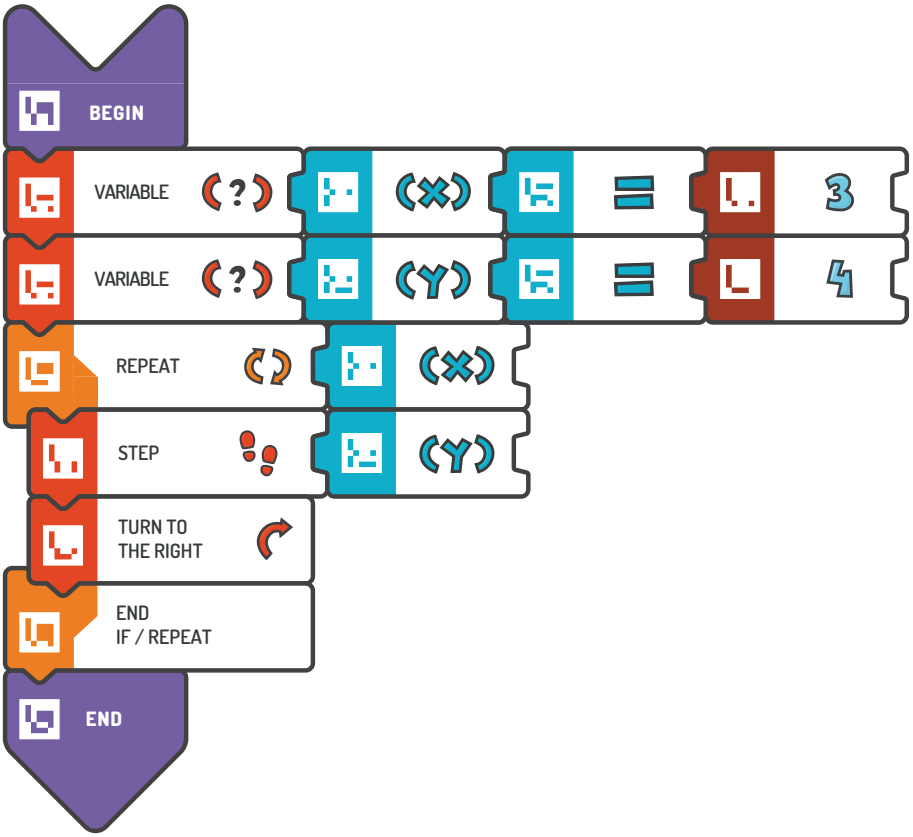
# INTRODUCTION:
# variables and loops

Use two variables: **X** and **Y** and a **REPEAT** loop in this quest. Use variable **X** to hold the number of repeats of the loop, and use variable **Y** to hold the number of steps.

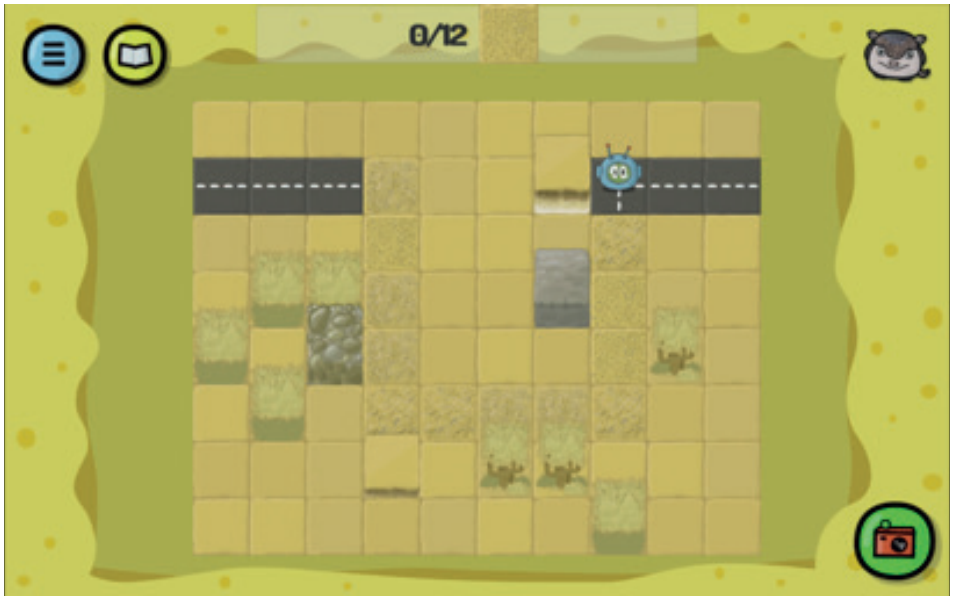
Challenge:




Example solution:



Challenge:




Example solution (in text format):

```
BEGIN
    VARIABLE X=3
    VARIABLE Y=4
    REPEAT (X) {
        STEP (Y)
        TURN TO THE RIGHT
    }
END
```

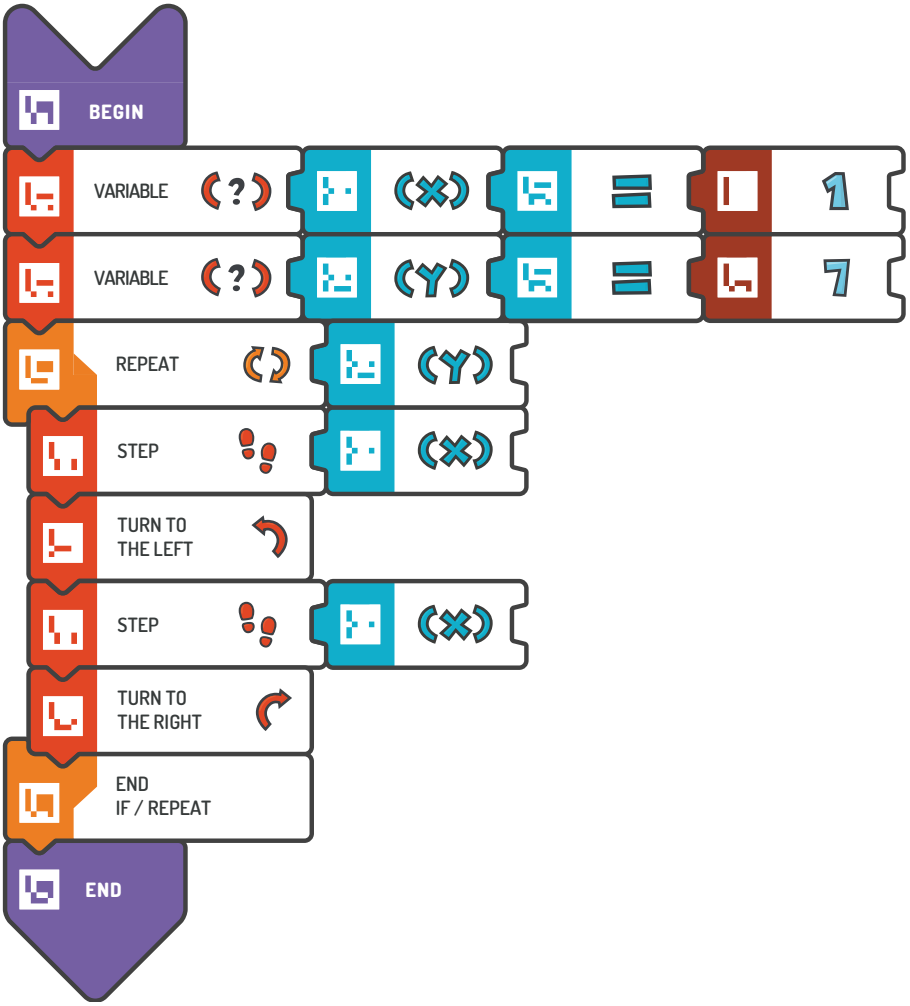
Challenge

(**In this quest players are supposed to use two variables.**

**This time one of them will hold the number of times the loop should be repeated**):




Example solution:



Challenge

(**In this quest players are supposed to use two variables.**

**This time one of them will hold the number of times the loop should be repeated)**:



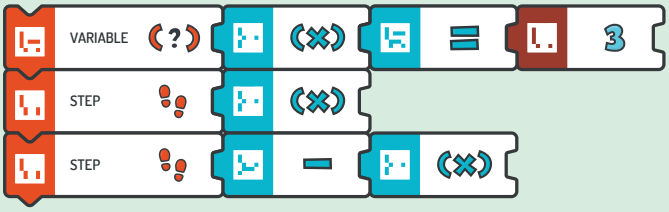

Example solution (in text format):

```
BEGIN
    VARIABLE X=1
    VARIABLE Y=7
    REPEAT (Y) {
        STEP (X)
        TURN TO THE LEFT
        STEP (X)
        TURN TO THE RIGHT
    }
END
```

# INTRODUCTION:
# mathematical operations on variables

You can also perform mathematical operations on variables.

In the part of the program presented below, by adding the **MINUS** tile in front of the variable **X** when it's called the second time we change the number of steps to the negative value of the variable.



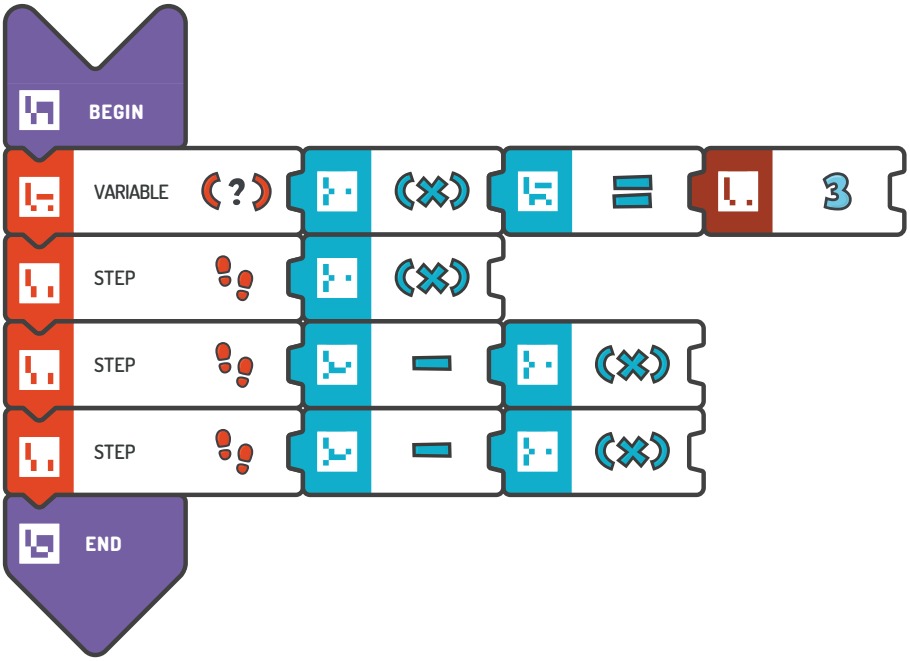

**In this quest players are supposed to use the tile with the number 3 only once.**

Challenge

(**In this quest players are supposed to use the tile with the number 3 only once**):




Example solution:



Challenge

(**In this quest players are supposed to use the tile with the number 3 only once**):




Example solution (in text format):

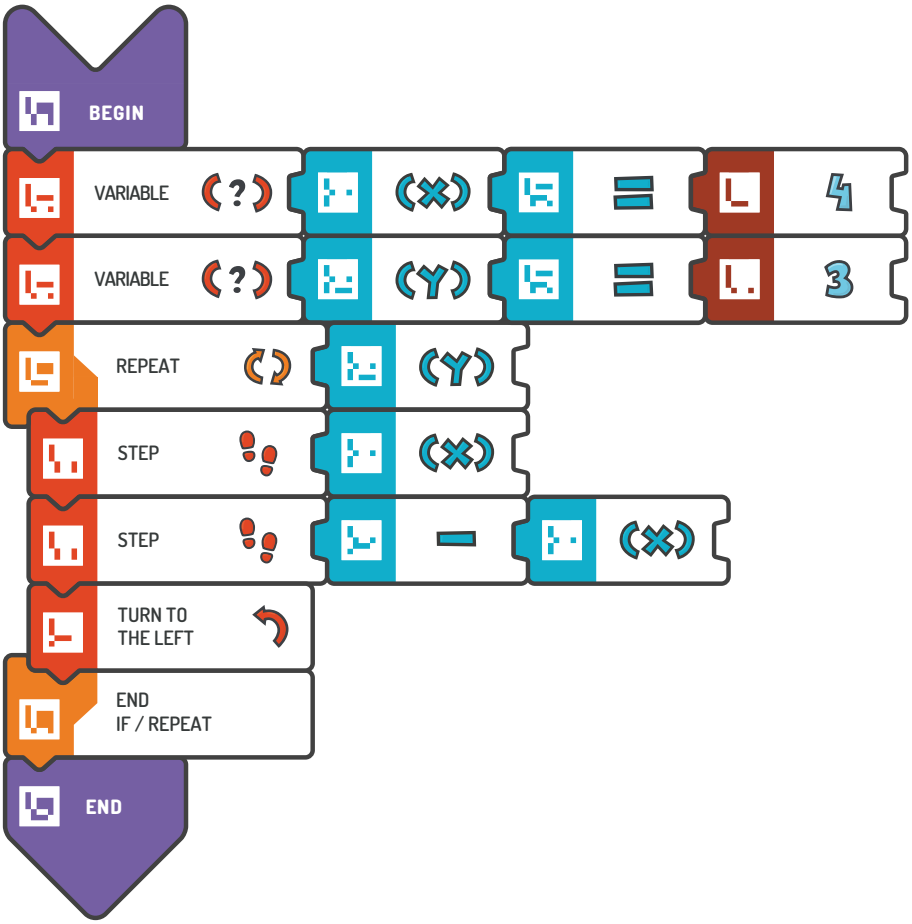
```
BEGIN
    VARIABLE X=3
    STEP (X)
    STEP (-X)
    STEP (-X)
END
```

Challenge

**(In this quest players are supposed to use two variables: one to hold the number of steps and the other for the number of the times the loop should be repeated):**




Example solution:



Challenge

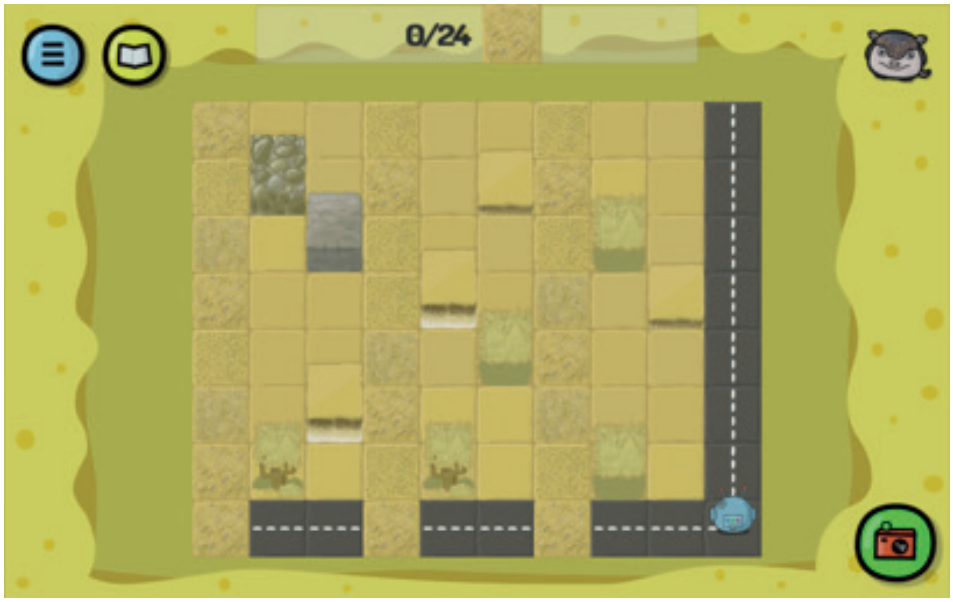
**(In this quest players are supposed to use two variables: one to hold the number of steps and the other for the number of the times the loop should be repeated):**



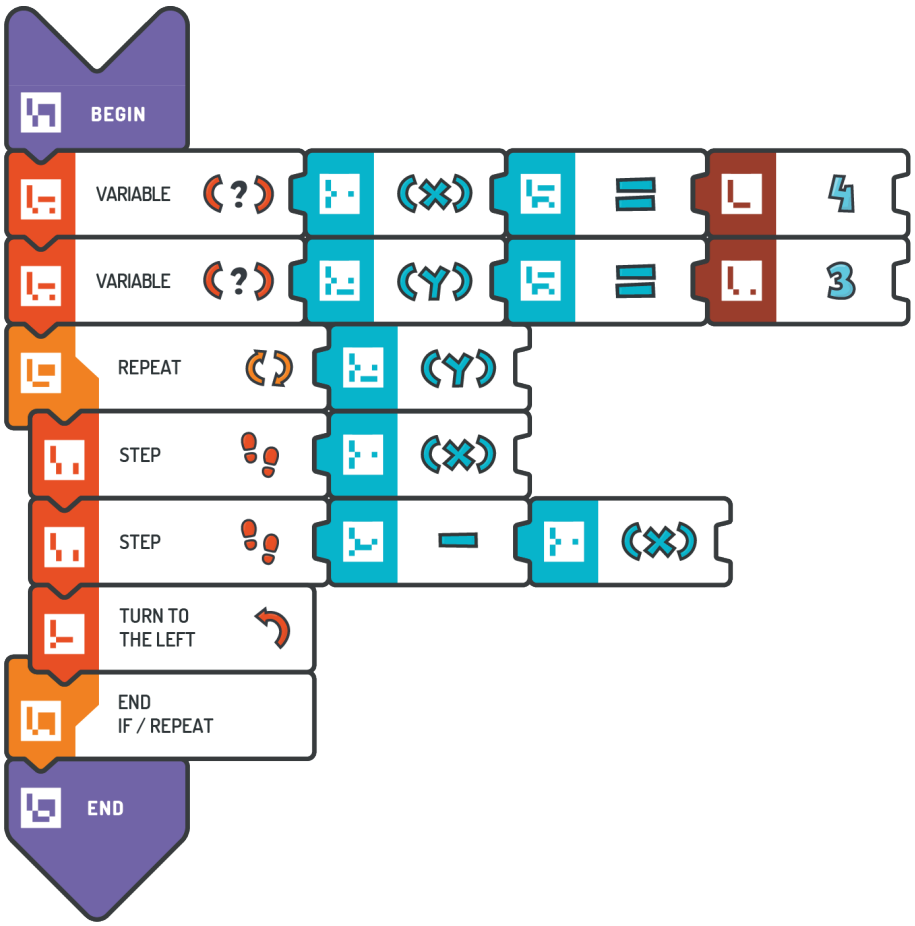

Example solution (in text format):

```
BEGIN
    VARIABLE X=4
    VARIABLE Y=3
    REPEAT (Y) {
        STEP (X)
        STEP (-X)
        TURN TO THE LEFT
    }
END
```

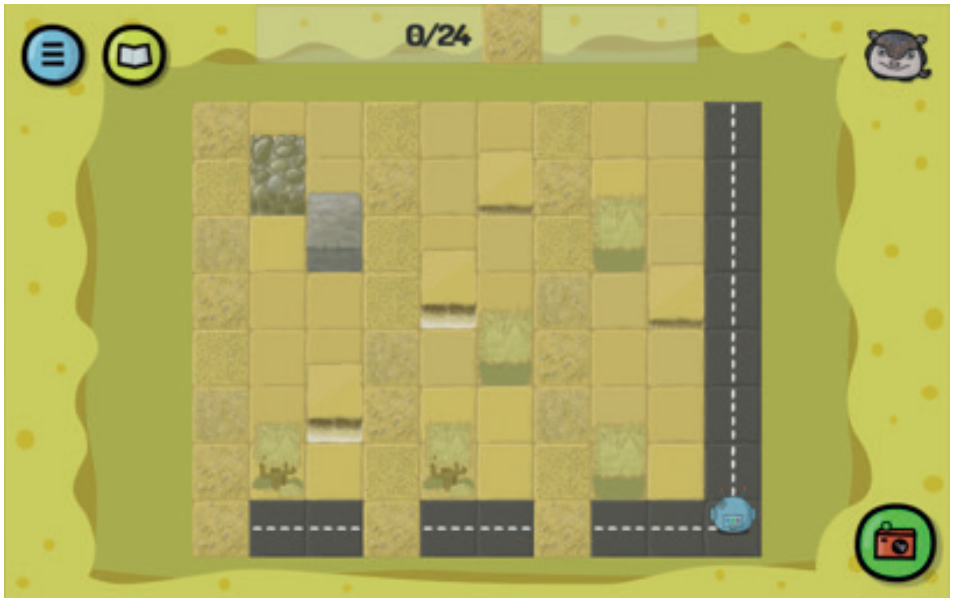
Challenge:




Example solution:



Challenge:




Example solution (in text format):

```
BEGIN
    VARIABLE Y=3
    VARIABLE X=7
    REPEAT (Y) {
        TURN TO THE LEFT
        STEP (3)
        TURN TO THE RIGHT
        STEP (X)
        STEP (-X)
    }
END
```

# Module VIII
# Australia 2



**Conditional statements - IF, ELSE**

**Plant missing trees in the appropriate squares.**

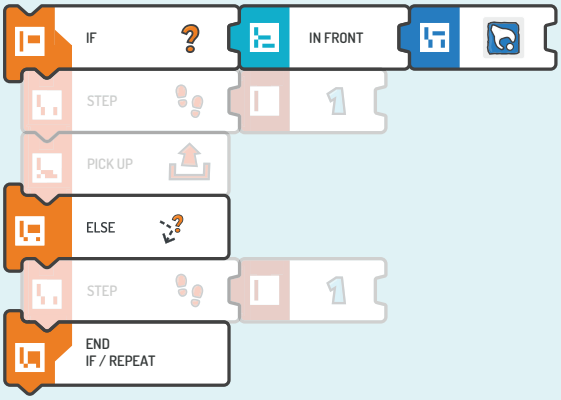
**In this module players are supposed to use a conditional command to complete each quest.**

# MODULE INTRODUCTION 1/4: conditional statements - IF, ELSE

Each of us takes some decisions every day. Should I put a jacket or a sweater when I go to school? How many sandwiches should I have for breakfast? Should I take a bus or an underground when I go to school? Computer programs always check for something, calculate something and make decisions all the time. When writing a program, if you decide to use a conditional statement, you must explain everything to the computer clearly so that the program knows exactly what to do.

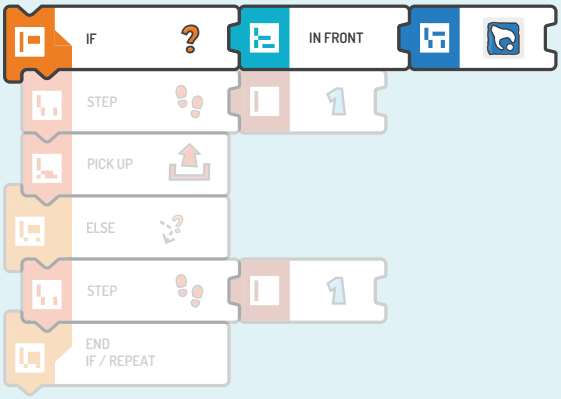
**The IF, ELSE command combination**

The IF, ELSE are two commands that often go together and constitute one of the most basic logical structures when making decisions. A conditional statement is a powerful tool and hence, it is going to appear in most computer programs.



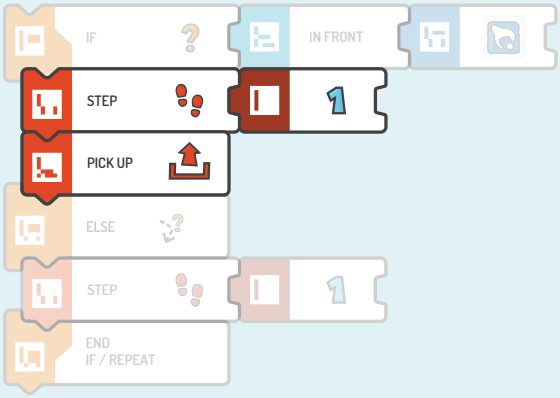

**A typical conditional statement comprises the following key elements:**

**1.** A condition to be reviewed,

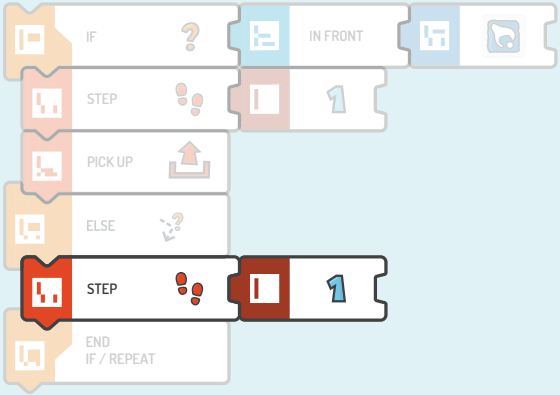


# MODULE INTRODUCTION 2/4: conditional statements

**2.** Commands that will be performed if the condition is met,

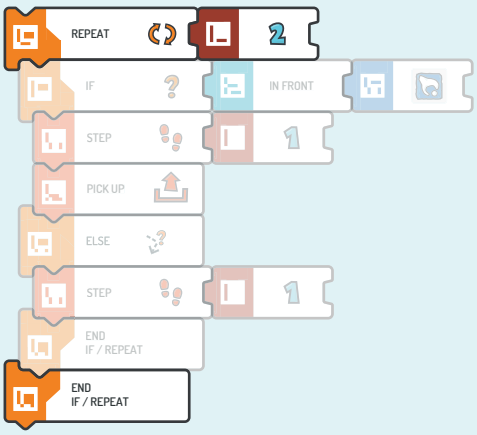



**3.** Commands that will be performed if the condition is not met.




**Let's look at the example below.** To make the notion of a conditional statement easier to understand, we will use (call) a conditional statement twice **by placing it inside a REPEAT (2) loop.**



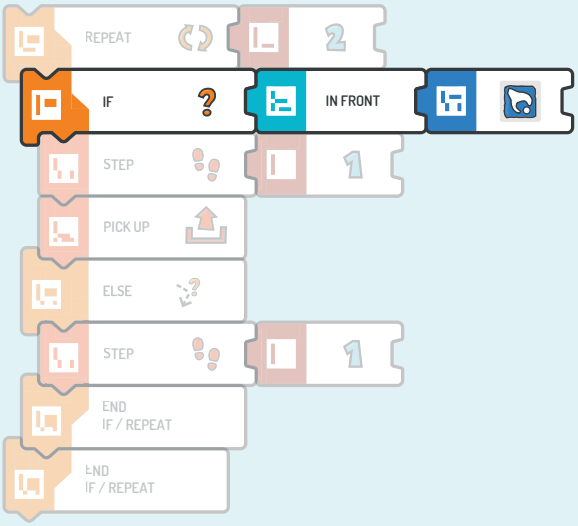


# MODULE INTRODUCTION 3/4: conditional statements

**A conditional statement in action:**

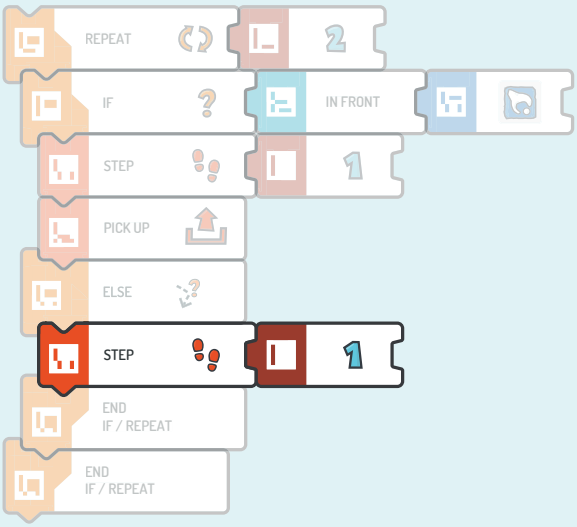
**1.** Scottie checks for a condition: **"Is there a COLLECTIBLE in front of me?"** for the first time. In the example below this condition is not met






**2.** If the condition is not met, the **ELSE** instruction is called into action. Consequently, Scottie moves one square ahead - **STEP (1).**



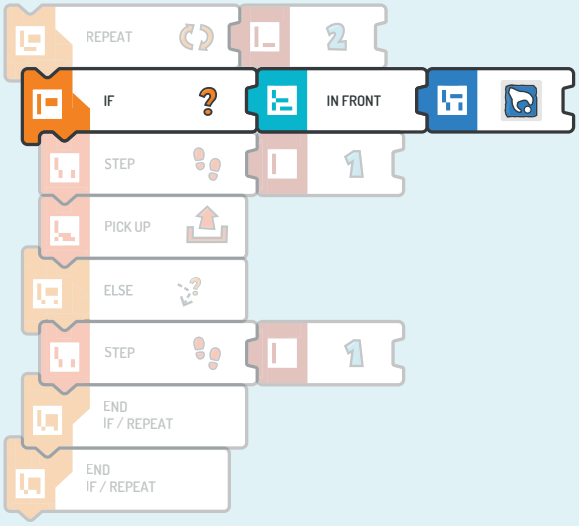


# MODULE INTRODUCTION 4/4: conditional statements

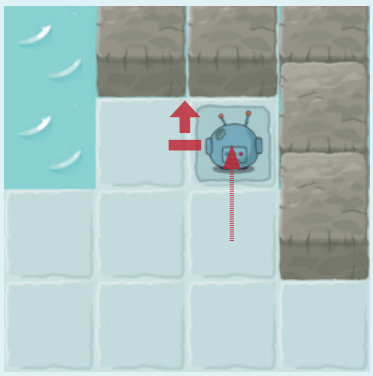
**A conditional statement in action, ctnd:**

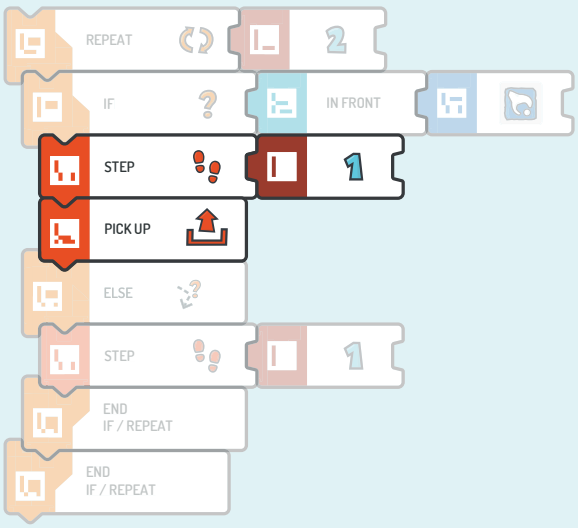
**3.** The **REPEAT (2)** loop requires that we check for the condition **"Is there a COLLECTABLE OBJECT in front of me?"** one more time. This time **the condition is met.**






**4.** As the condition is met, Scottie begins to perform the commands directly below the **IF** line, namely **STEP (1)** and **PICK UP.**






**NOTE**

**If the first condition from the conditional statement is met, other conditions are no longer checked or performed.**

# INTRODUCTION:
# action square

One of Scottie's animal friends has a beautiful orchard. In exchange for his friend's assistance, Scottie promised to plant a few trees for him.

Plant the missing trees in the designated squares, using new tiles to achieve this goal. This quest focuses on using the **IF** conditional statement in connection with the **ACTION SQUARE** tile. The condition that will be checked is the location of the action square in relation to Scottie, e.g.:



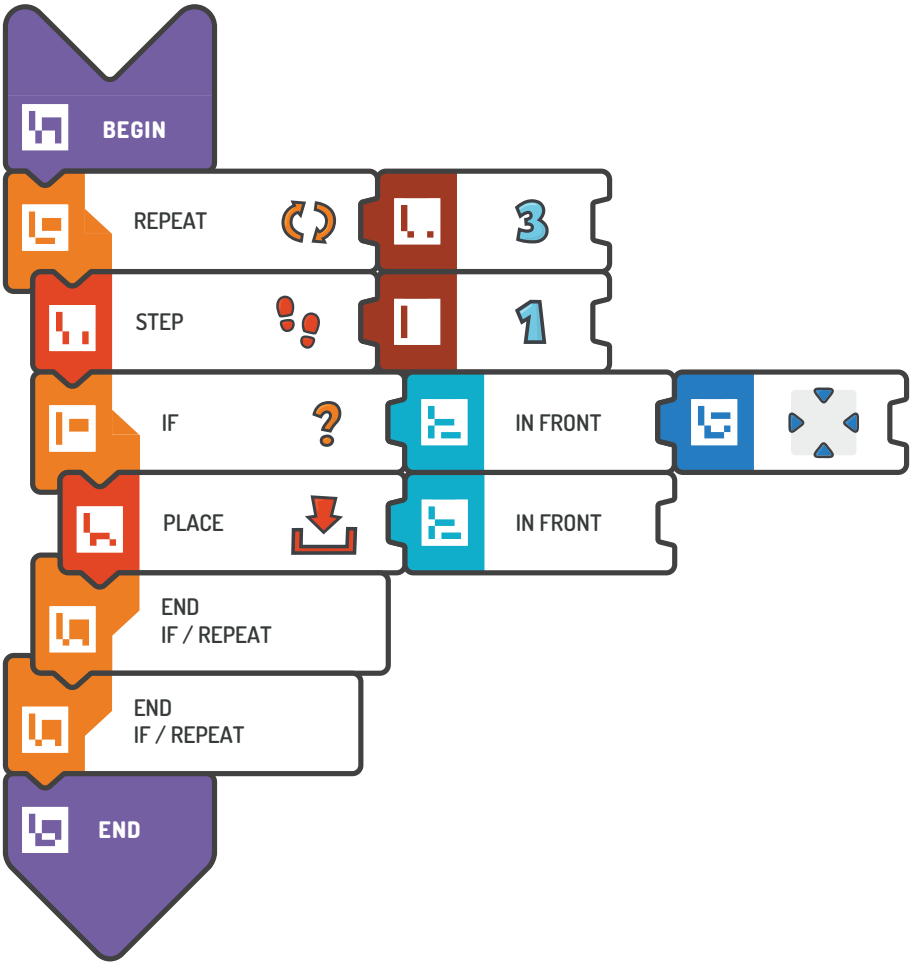

**REMEMBER!**

You should always place the **END IF / REPEAT** tile at the end of a conditional statement.

Challenge **(Remember to use a conditional statement in this quest)**:




Example solution:



Challenge **(Remember to use a conditional statement in this quest)**:



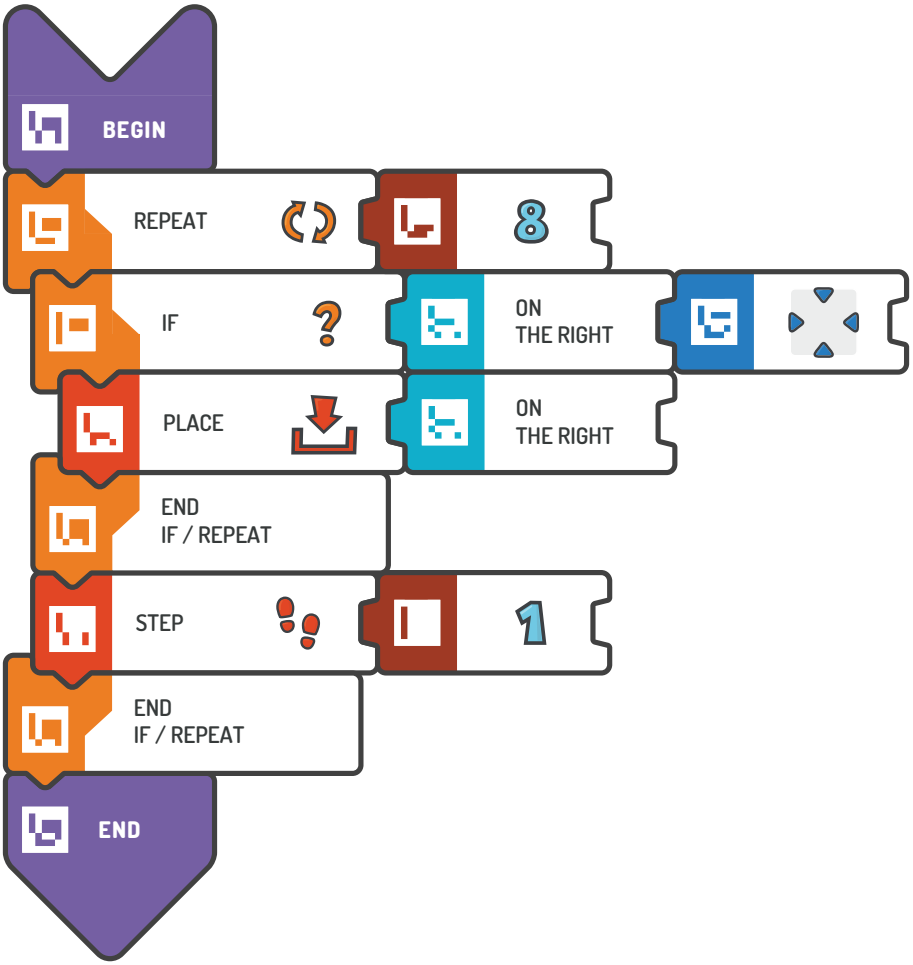

Example solution (in text format):

```
BEGIN
    REPEAT (3) {
        STEP (1)
        IF (IN FRONT < ACTION SQUARE) {
            PLACE (IN FRONT)
        }
    }
END
```

Challenge **(Remember to use a conditional statement in this quest)**:




Example solution:



Challenge **(Remember to use a conditional statement in this quest)**:




Example solution (in text format):

```
BEGIN
    REPEAT (8) {
        IF (ON THE RIGHT < ACTION SQUARE) {
            PLACE (ON THE RIGHT)
        }
        STEP (1)
    }
END
```

Challenge **(Remember to use a conditional statement in this quest)**:




Example solution:



Challenge **(Remember to use a conditional statement in this quest)**:



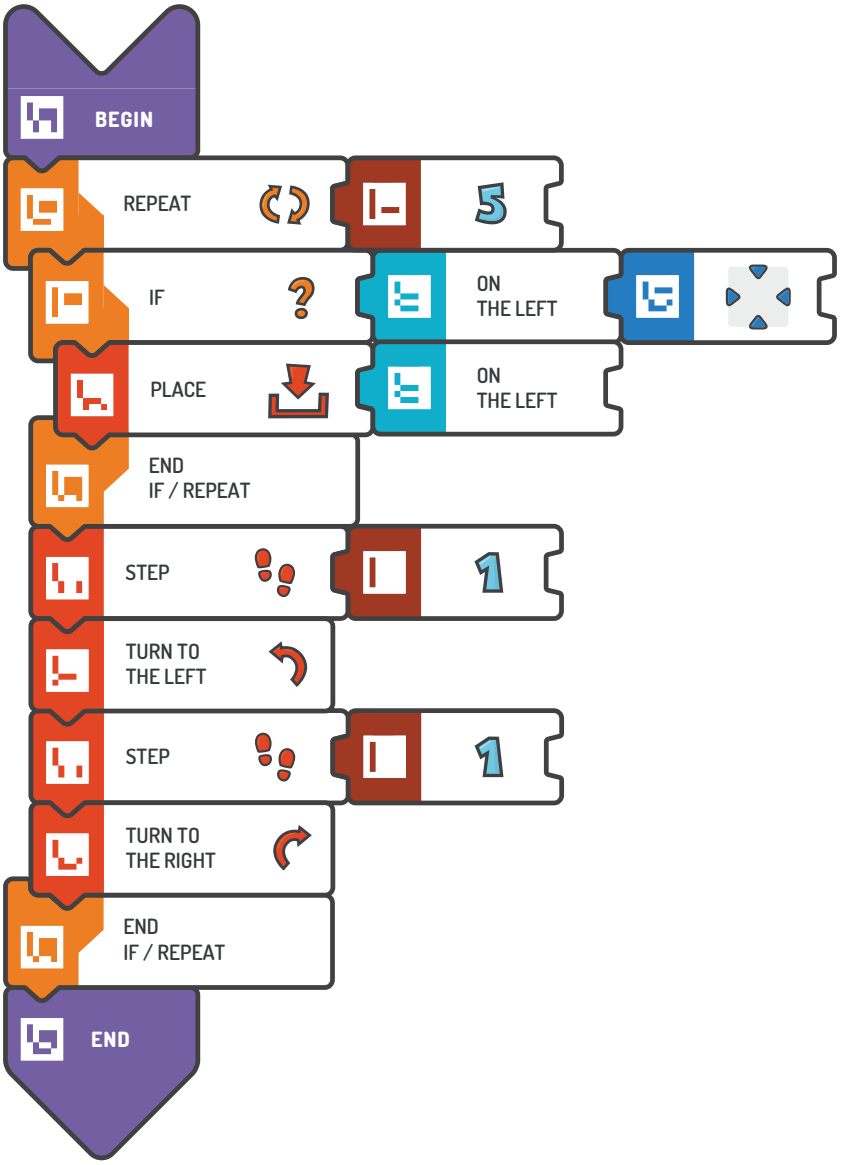

Example solution (in text format):

```
BEGIN
    REPEAT (7) {
        IF (ON THE LEFT < ACTION SQUARE) {
            TURN TO THE LEFT
            STEP (1)
            PLACE (IN FRONT)
            STEP (-1)
            TURN TO THE RIGHT
            PLACE (ON THE LEFT)
        }
        STEP (1)
    }
END
```

Challenge **(Remember to use a conditional statement in this quest)**:




Example solution:



Challenge **(Remember to use a conditional statement in this quest)**:




Example solution (in text format):

```
BEGIN
    REPEAT (5) {
        IF (ON THE LEFT < ACTION SQUARE) {
            PLACE (ON THE LEFT)
        }
        STEP (1)
        TURN TO THE LEFT
        STEP (1)
        TURN TO THE RIGHT
    }
END
```

# HINT

In this quest you need to connect using a conditional loop with the **IF** statement that will add another condition.

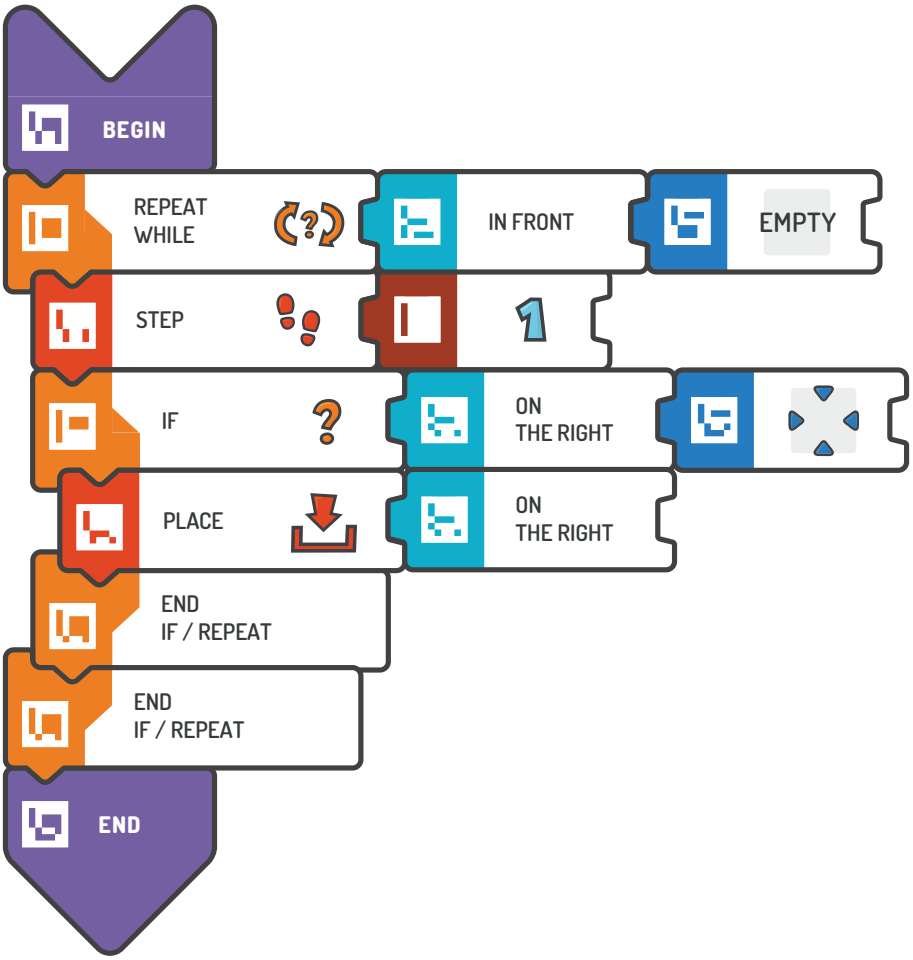
As there is an obstacle on Scottie's way, you can use the **REPEAT WHILE** loop instead of the regular **REPEAT** loop.

Challenge

**(Look, there is an obstacle at the end of the game board. Use it to your advantage and use the REPEAT WHILE loop instead of the regular REPEAT loop)**:




Example solution:



Challenge

(**Look, there is an obstacle at the end of the game board. Use it to your advantage and use the REPEAT WHILE loop instead of the regular REPEAT loop**):

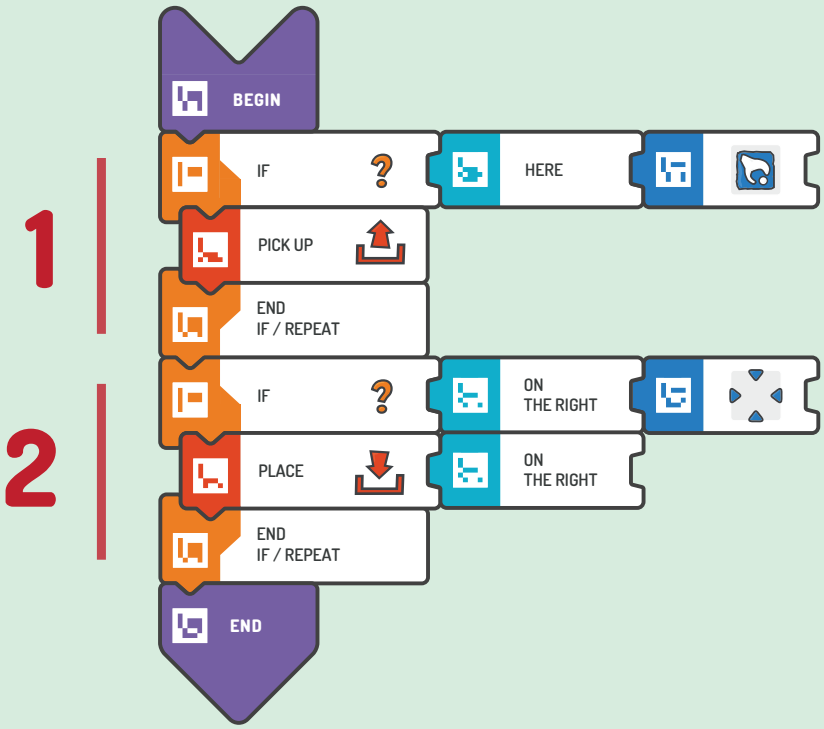



Example solution (in text format):

```
BEGIN
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
        IF (ON THE RIGHT < ACTION SQUARE) {
            PLACE (ON THE RIGHT)
        }
    }
END
```

# INTRODUCTION:
# using multiple conditional statements

You can use more than one conditional statement in a program. It's an excellent way to test and improve you logic. It is also a great way to practice what is a key component of syntax for every popular coding language. Each conditional statement can check for a different condition, e.g.:




**In the program above Scottie performs the following actions:**

1. **First** he will check if there is a collectable object on the square where he is currently standing:
   - **if yes,** Scottie will pick up the object (and move on to check the next condition),
   - **if no,** Scottie will skip the command directly below the **IF** line (and move on to check the next condition).

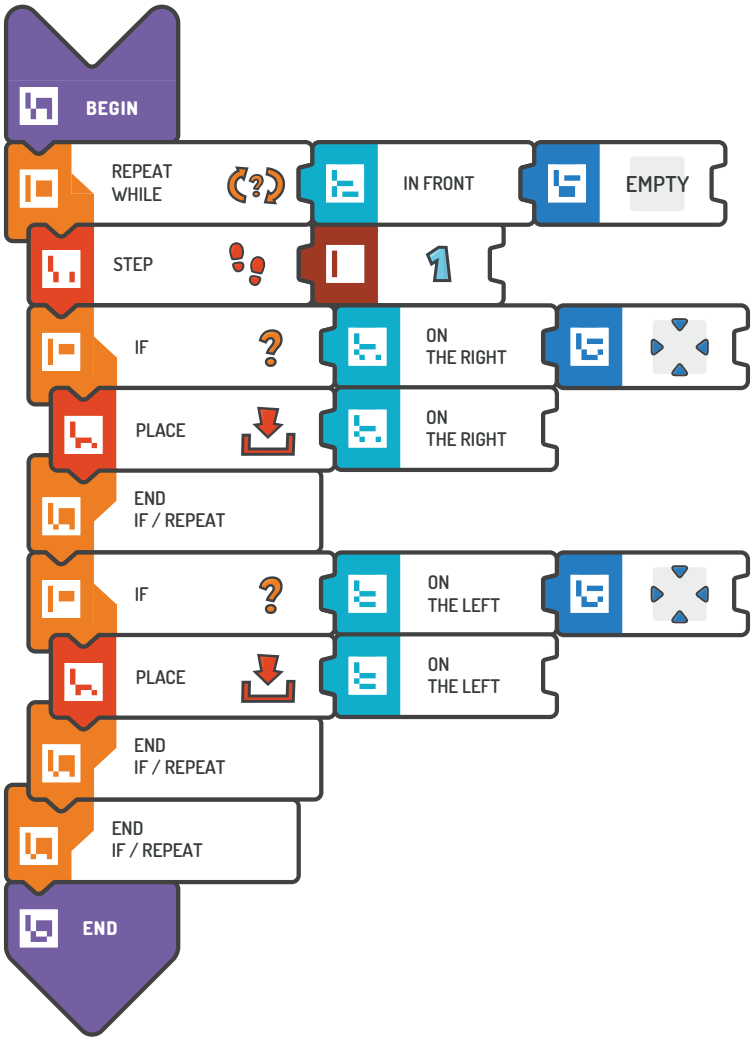
2. **Then** Scottie will check if there is an action square on the square to his right:
   - **if yes,** Scottie will place the object from his inventory on this square (and move on to check the next condition)
   - **if no,** Scottie will skip the command directly below the second **IF** line (and in this case, the program will terminate with the **END** tile.

Challenge

**(Use more than one conditional statement in this quest. Remember that each conditional statement can check for a different condition)**:




Example solution:



Challenge

**(Use more than one conditional statement in this quest. Remember that each conditional statement can check for a different condition)**:




Example solution (in text format):

```
BEGIN
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
        IF (ON THE RIGHT < ACTION SQUARE) {
            PLACE (ON THE RIGHT)
        }
        IF (ON THE LEFT < ACTION SQUARE) {
            PLACE (ON THE LEFT)
        }
    }
END
```

Challenge

**(Scottie no longer has new seedlings in his inventory, so he must pick up a seedling before he can plant it in a new place)**:




Example solution:



Challenge

**(Scottie no longer has new seedlings in his inventory, so he must pick up a seedling before he can plant it in a new place)**:




Example solution (in text format):

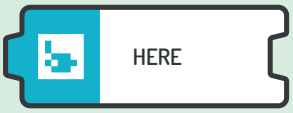
```
BEGIN
    REPEAT WHILE (IN FRONT < EMPTY) {
        STEP (1)
        IF (ON THE RIGHT < ACTION SQUARE) {
            PLACE (ON THE RIGHT)
        }
        IF (ON THE LEFT < COLLECTABLE OBJECT) {
            TURN TO THE LEFT
            STEP (1)
            PICK UP
            STEP (-1)
            TURN TO THE RIGHT
        }
    }
END
```

# INTRODUCTION:
# ELEVATED SQUARE and HERE tiles

Another part of Scottie's friend in Australia is a rocky and hilly area. Show Scottie how to move around hills. Use the **ELEVATED SQUARE** tile



and the **HERE** tile.

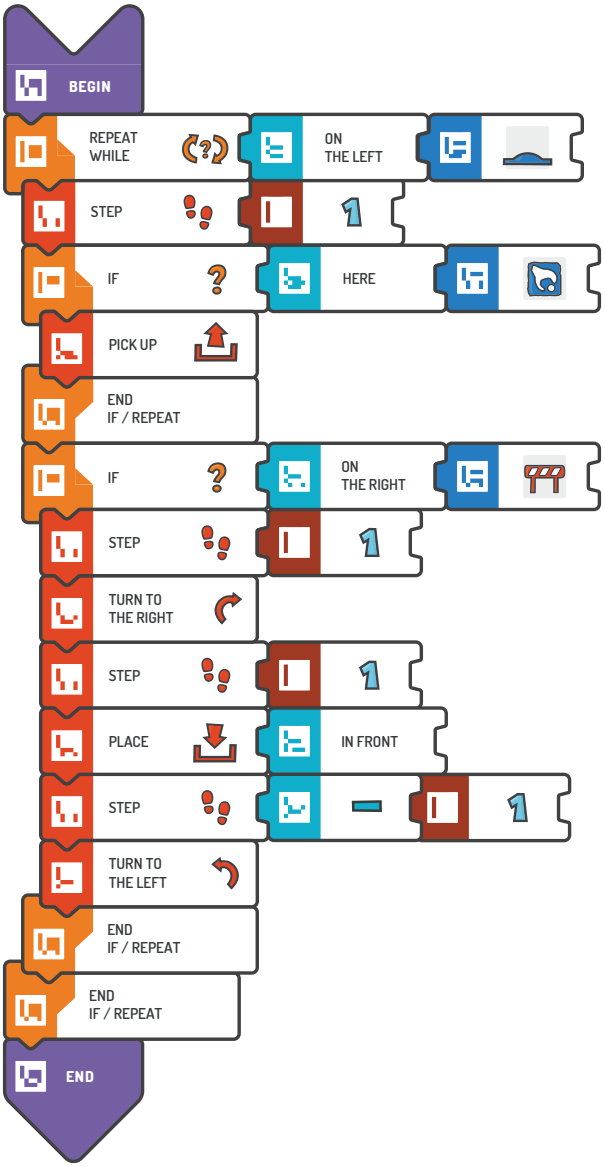



Use the new tiles in connection with the conditional statements which you have learned before.

Challenge **(Use a conditional statement in this module)**:




Example solution:



Challenge **(Use a conditional statement in this module)**:




Example solution (in text format):

```
BEGIN
    REPEAT WHILE (ON THE LEFT < ELEVATED SQUARE) {
        STEP (1)
        IF (HERE < COLLECTABLE OBJECT) {
            PICK UP
        }
        IF (ON THE RIGHT < OBSTACLE) {
            STEP (1)
            TURN TO THE RIGHT
            STEP (1)
            PLACE (IN FRONT)
            STEP (-1)
            TURN TO THE LEFT
        }
    }
END
```

# Module IX
# North America 2



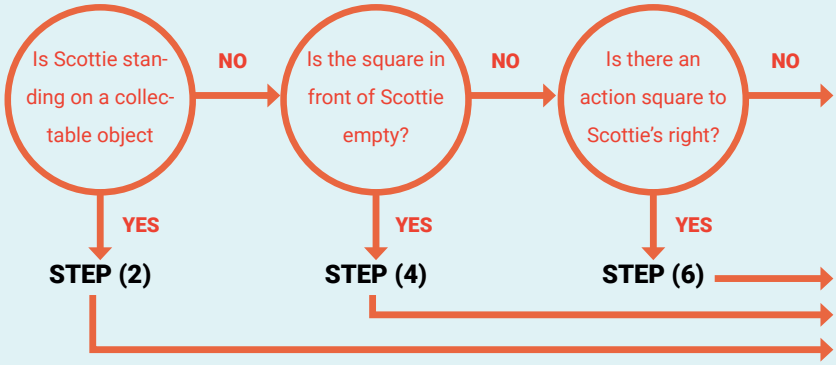
Loops and conditional statements: IF, ELSE IF, ELSE

Scottie's spaceship is almost ready! However, workers have left the workshop in a horrible mess. Help Scottie to collect screws and tools scattered on the floor.

# MODULE INTRODUCTION (A): ELSE IF 1/3

In **MODULE VIII** we saw that it is often necessary to use multiple conditional statements in a program. This need resulted in a new coding structure evolving, wherein only one out of a number of conditions will be executed.

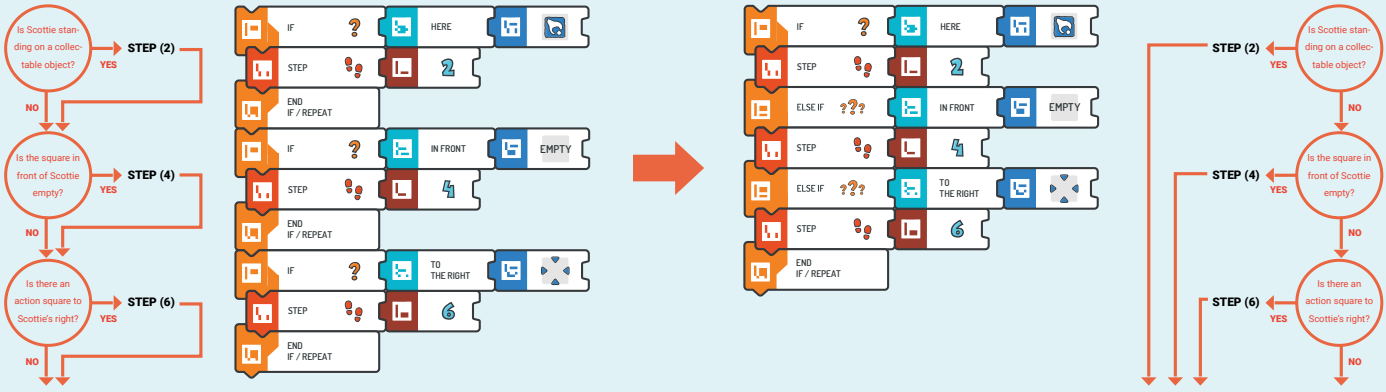
**Let's check for multiple conditions in order as in the example below.**




In such cases we have to use the conditional statement **ELSE IF.**




By using the **ELSE IF** tile, we can simplify the program by extending one conditional statement instead of compiling muliple conditional statements.

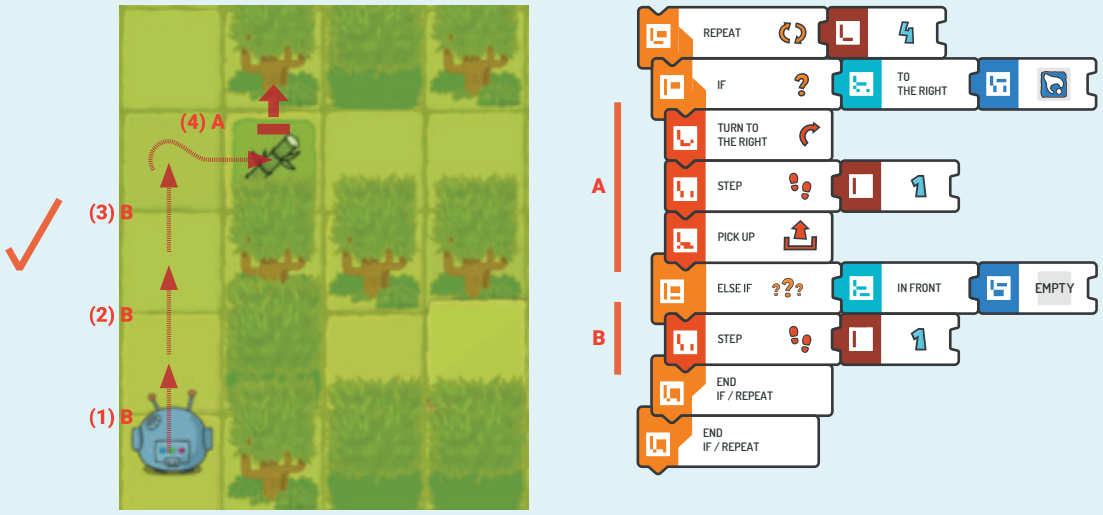



**When we use the ELSE IF tile, the order of the conditions is of key importance.**
If the first condition is met (is true), **the second and third conditions won't be checked or performed**. If the first condition is not met, the program moves on to check the second condition and if it's met (is true), **the third condition won't be checked or performed.**

# MODULE INTRODUCTION (A): ELSE IF 2/3

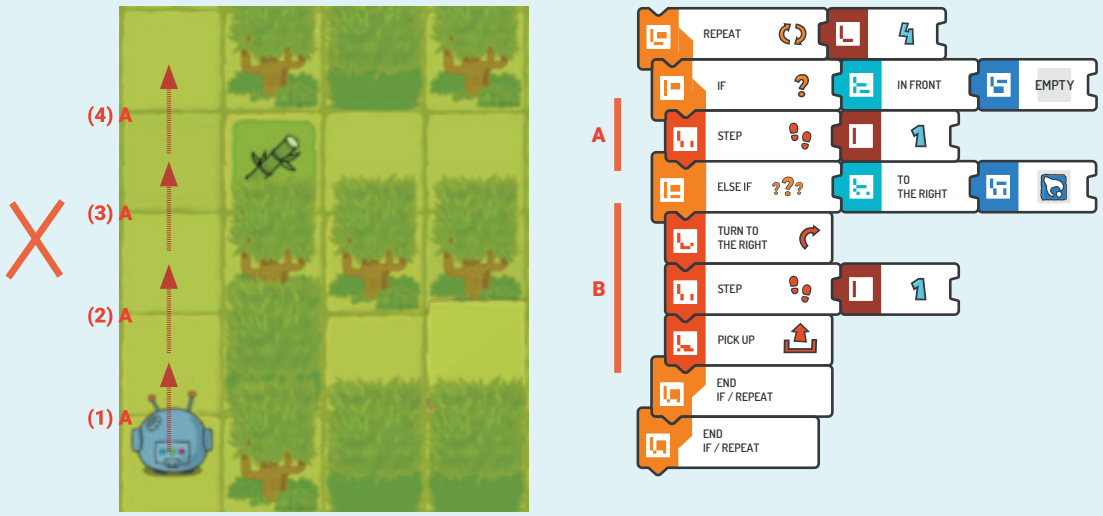
**Example I** - using **ELSE IF** statement. **Scottie is supposed to pick up the bamboo:**

**Correct solution.**




The difference between the correct and incorrect solution **is only in the order of the conditions to be checked** (and the commands corresponding to those conditions). Note that **each repetition (iteration) of the loop results in performing commands related to only one condition.**

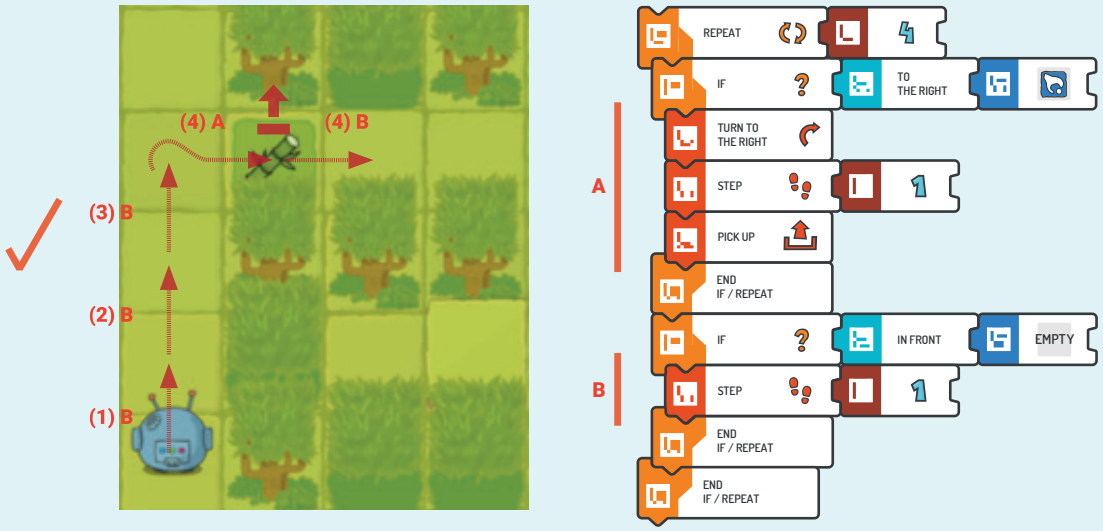
**Incorrect solution.**



# MODULE INTRODUCTION (A): ELSE IF 3/3

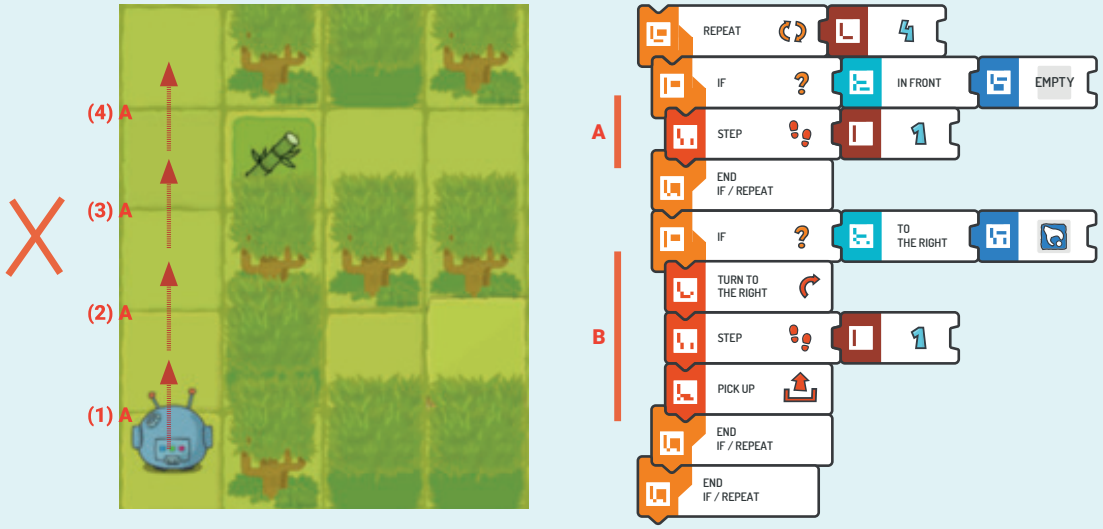
**Example II** - using two IF statements. **Scottie is supposed to pick up the bamboo:**

**Correct solution.**



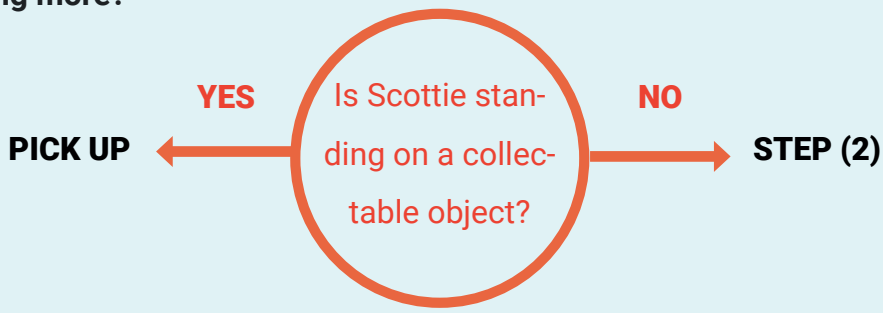

The correct solution **without using the ELSE IF tile is one line longer than before. Note that in the fourth repetition (iteration) of the loop both conditions are met (are true).** Of course the order of conditions remains of key importance in such cases.

**Incorrect solution.**



# MODULE INTRODUCTION (B): the ELSE tile 1/3

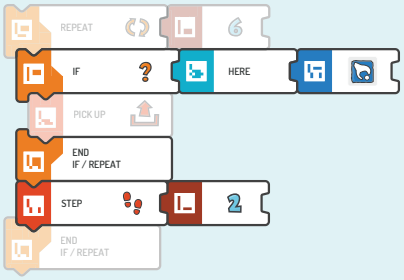
Apart from having to work with multiple conditional statements, sometimes you will need Scottie to do something **only if the previous conditions are not met**, e.g. **how to code two alternative actions and nothing more?**



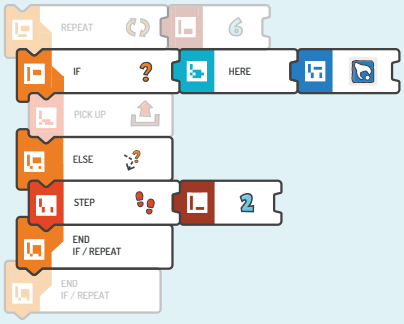

To build the structure presented above, in coding we use the **ELSE** instruction.




**It's an effective way to describe "default" commands for Scottie, i.e. commands that he will perform any time the condition specified earlier in the program is not met.**
**If some commands are located outside of the conditional statement, they will always be performed. This can be observed especially well in a loop.**




**In each repetition (iteration) of the loop, Scottie will ALWAYS take two steps.**



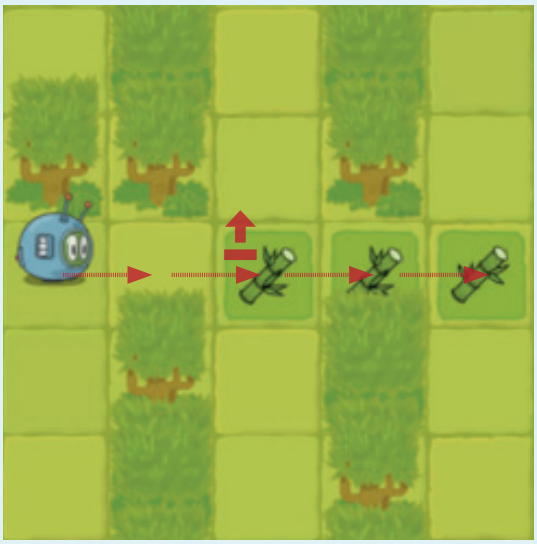

**In each repetition (iteration) of the loop, Scottie will take two steps ONLY if he is not standing on a collectable object.**

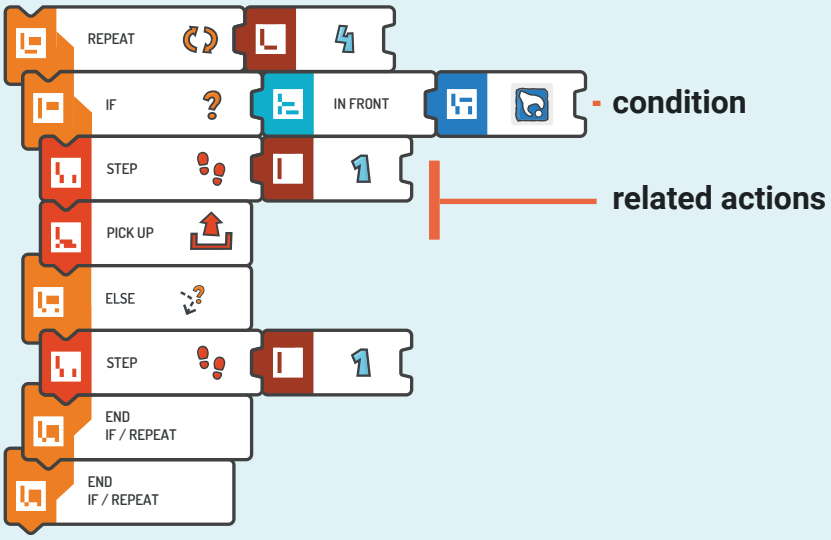
In a typical programming language **(we don't do that in Scottie Go!)** you can also use the word **NO** when creating similar conditional statements.

# MODULE INTRODUCTION (B): the ELSE tile 2/3

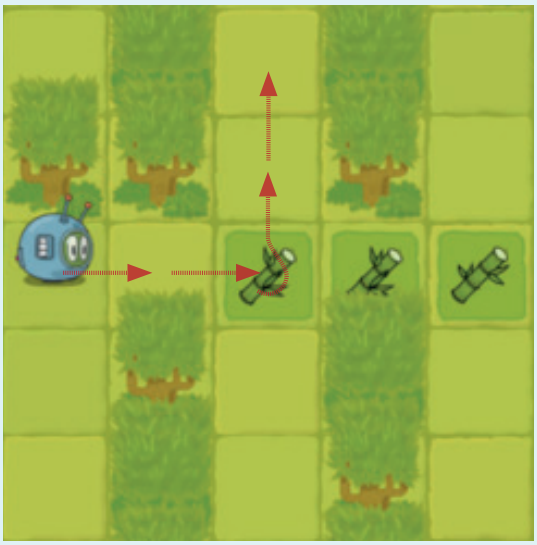
Examples of using the **ELSE** statement:

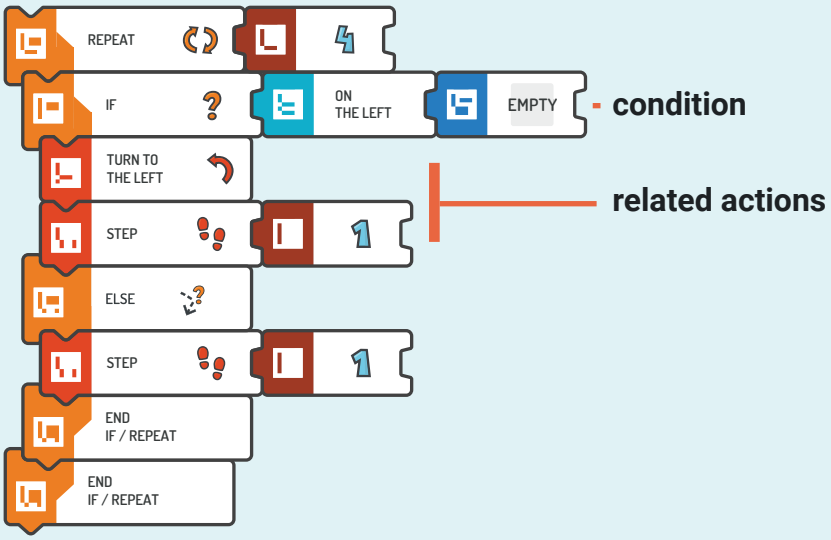
Let's write a program in which Scottie will move forward until something requires his action, e.g. **Scottie will check if there is a collectable object in front of him or he will want to turn left if there is an empty square to the left of him**. The program should be easy to modify and adjust to our (new) needs.





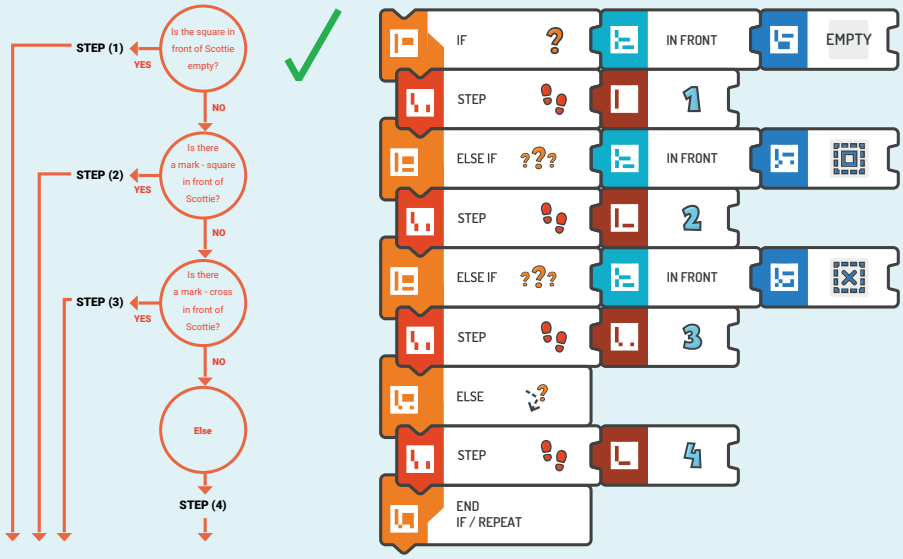

You don't have to define the condition needed for Scottie to walk forward. **It's enough to specify to which situation Scottie should react and what his reaction should be.**



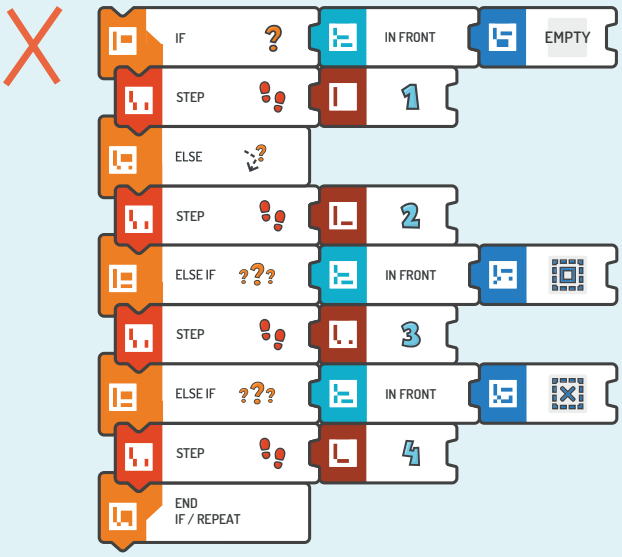


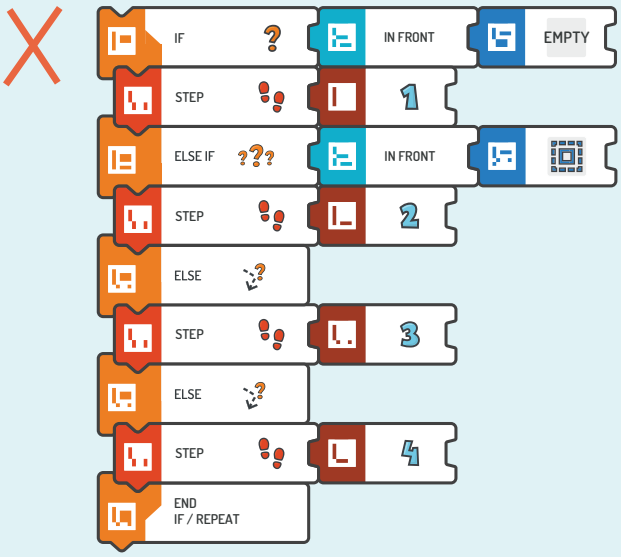
# MODULE INTRODUCTION (B): the ELSE tile 3/3

We can use both statements: **ELSE IF** and **ELSE** in one program.,




but remember that **you are allowed to use ELSE only once and it must be placed at the end of the program**. The parts of programs presented below are incorrect.



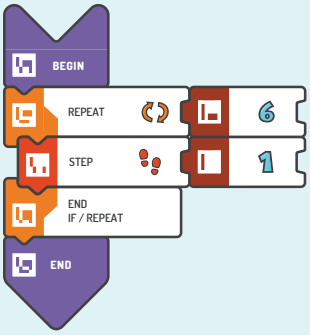
# MODULE INTRODUCTION (C): [optional] - REPEAT (FOREVER) 1/3

In programs that contain a conditional statement without a **REPEAT** loop, the conditional statement will be checked only once and then the program will end. But we want the condition to be checked multiple times until Scottie finds and collects all the screws and tools.
In order for that to happen, we can use a new tile: **FOREVER**.

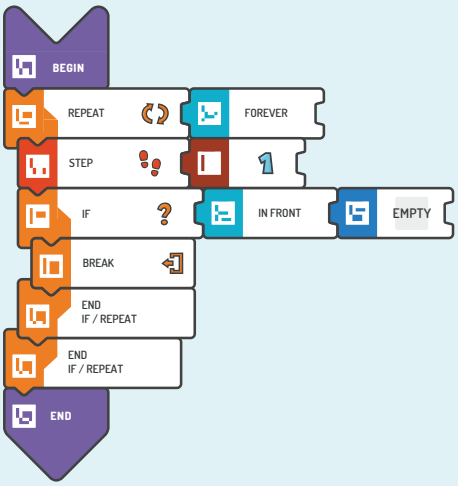



In Module IX quests you need to use the **REPEAT** loop in one of the two manners described below, depending on your pupils' age and experience:

**1. REPEAT (specific number of times)** - the command that your pupils know well from Module III. The challenge is to specify how many times the commands inside the loop should be execured.
**In the example below Scottie will take 1 step 6 times paying no attention to his surroundings.**



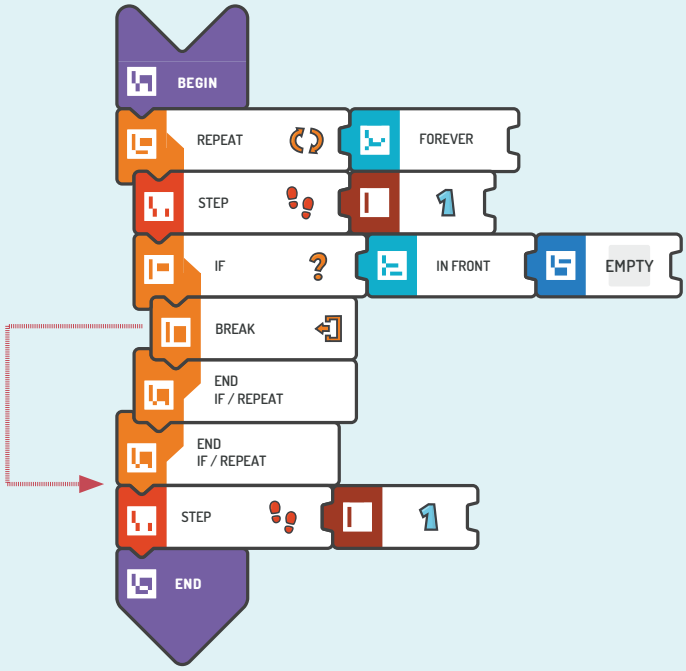

**2. REPEAT (FOREVER)** with the **BREAK** command.




The **REPEAT (FOREVER)** command is not introduced in the tutorial, but it's a more interesting and the most universal solution. **A similar program will be analysed in detail on the next page.**

# MODULE INTRODUCTION (C): [optional] - REPEAT (FOREVER) 2/3

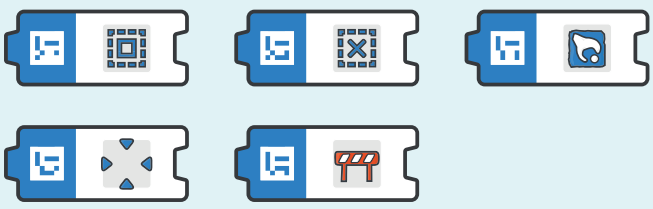
**REPEAT (FOREVER)** with the **BREAK** command.




In the program above the **STEP 1** command will be repeated until the program calls the **BREAK** command (which we usually place within a conditional statement, e.g. one related to an object on the game board). **In our example Scottie will stop performing the loop if there is an empty square in front of him.**

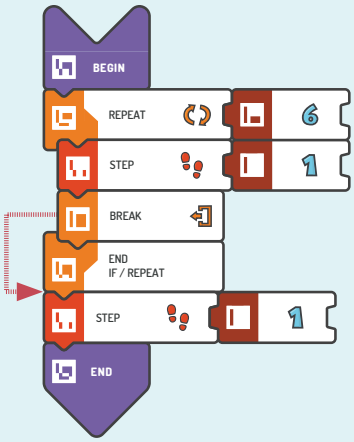
The condition that will call the **BREAK** command, placed in a strategic location, will be of key importance for our program to function correctly. **The commands that are placed inside the loop, but after the BREAK command, will not be executed.**

In this module on the workshop floor you can spot: **MARK - SQUARE, MARK - CROSS, collectable object objects, action squares** and **obstacles**. You can use them as reference points to construct appropriate **conditional statements**.

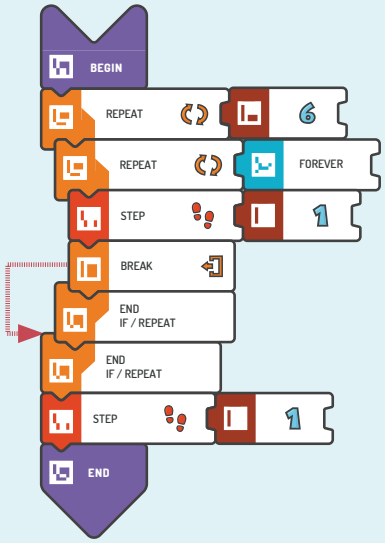


# MODULE INTRODUCTION (C): [optional] - REPEAT (FOREVER) 3/3

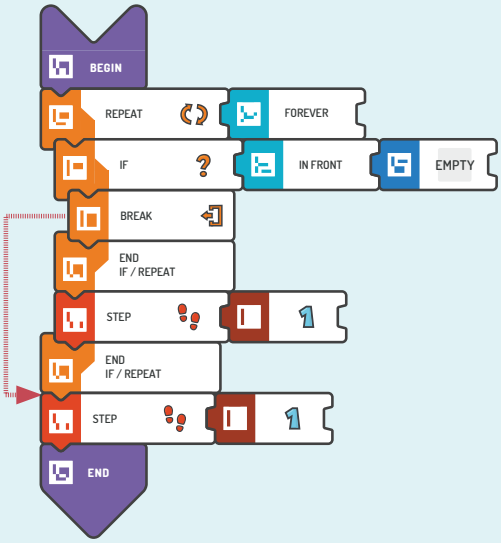
Executing the **BREAK** command will make the program **jump outside of any loop in which it is placed.**




**Scottie will take 2 steps.**
(1 in the loop;
1 outside of the loop)




**Scottie will take 7 steps.**
(1 in the loop for each iteration of the **FOREVER** loop - which is just once - repeated 6 times;
1 outside of the loops)



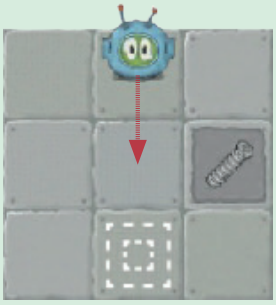

**Scottie will move forward until he stops on an empty square.**
(If the square in front of Scottie is not empty, Scottie will always take 1 step;
if there is an empty square in front of him, the program goes outside of the loop and makes Scottie take one last step before the **END** command)

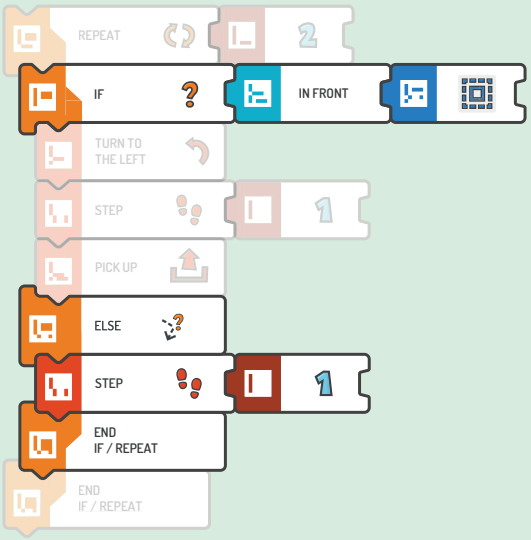
# INTRODUCTION: ELSE (1/2)

The quests in Module IX require you to use a **REPEAT** loop in connection with **conditional statements**. In this quest you are supposed to use the **ELSE** statement.

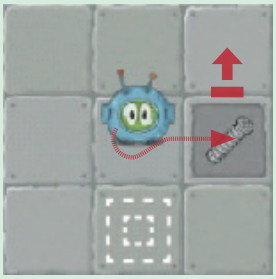



The **ELSE** tile lets you define a set of commands which Scottie will perform only in case the previous conditions are not met. Let's look at an example of a program **in which Scottie performs actions based on the relation between where he is standing and SQUARE MARKS located on the game board**.






**1st ITERATION OF THE LOOP:**
There is no MARK - SQUARE in front of Scottie ->
**Scottie executes the commands located between the ELSE and END IF / REPEAT tiles.**





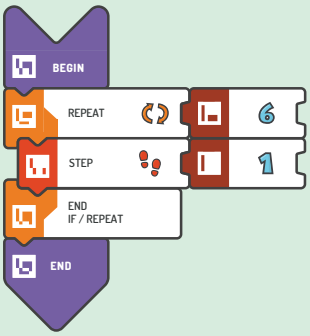

**2nd ITERATION OF THE LOOP:**
There is a MARK - SQUARE in front of Scottie ->
**Scottie executes the commands located between the IF and ELSE tiles.**

# INTRODUCTION: ELSE (2/2)

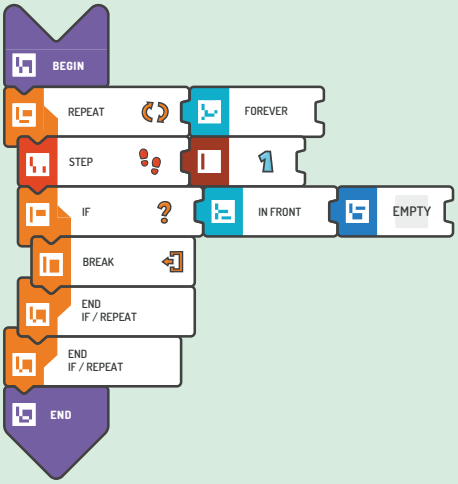
> **NOTE!!**
>
> **In this and subsequent quests you can use the REPEAT (FOREVER) loop.**

In Module IX quests you need to use the **REPEAT** loop in one of the two manners described below, depending on your pupils' age and experience:

**1. REPEAT specified number of times** - the command that your pupils know well from Module III. The challenge is to specify how many times the commands inside the loop should be repeated. In the example below Scottie will take 1 step 6 times paying no attention to his surroundings.




**2. REPEAT (FOREVER)** with the **BREAK** command.

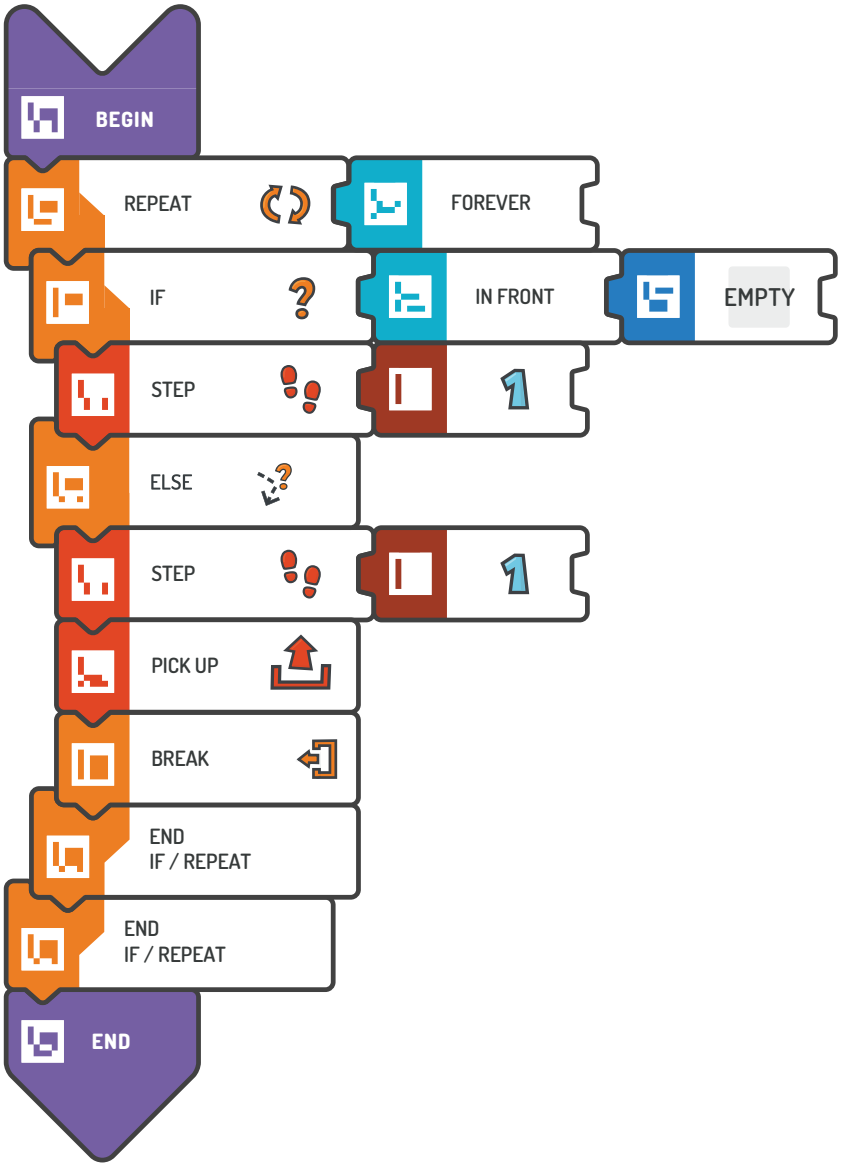



The **REPEAT (FOREVER)** command is not introduced in the tutorial, but it's a more interesting and the most universal solution. **You can read more about it in the Module introduction (C)**.

Challenge **(Use the ELSE tile in this quest)**:




Example solution:



Challenge **(Use the ELSE tile in this quest)**:




Example solution (in text format):

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < EMPTY) {
            STEP (1)
        } ELSE {
            STEP (1)
            PICK UP
            BREAK
        }
    }
END
```

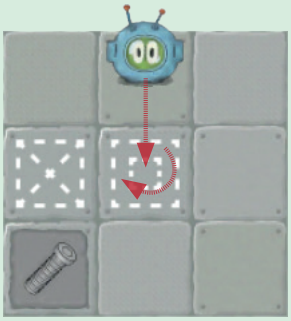
# INTRODUCTION:
# ELSE IF and MARKS (1/3)

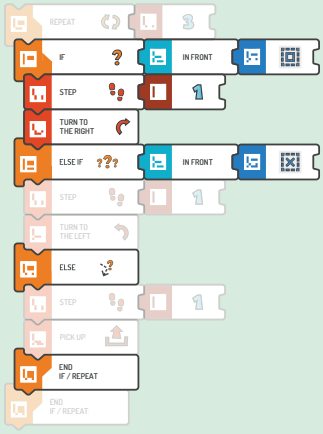
The quests in Module IX require you to use a **REPEAT** loop in connection with **conditional statements.** In this quest you are supposed to use the **ELSE IF** statement.




The **ELSE IF** tile lets you define **additional conditions that Scottie will check** in case the first condition is not met. Scottie will check conditions one by one **until one of them is met**.

For example:






**1st ITERATION OF THE LOOP:**

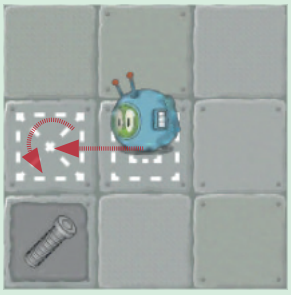
There is a MARK - SQUARE

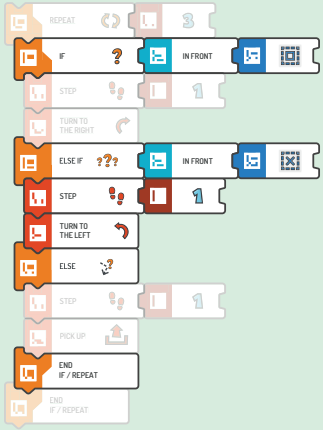
in front of Scottie ->

**Scottie executes:**

**STEP (1)**

**TURN TO THE RIGHT**






**2nd ITERATION OF THE LOOP:**

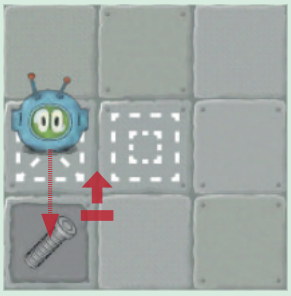
There is a MARK - CROSS

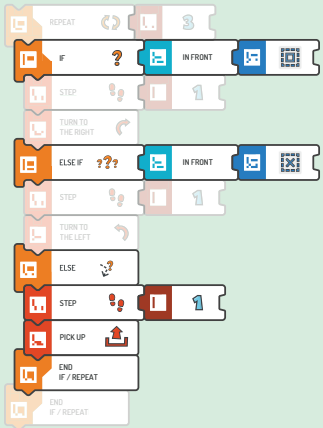
in front of Scottie ->

**Scottie executes:**

**STEP (1)**

**TURN TO THE LEFT**






**3rd ITERATION OF THE LOOP:**

There is no MARK

in front of Scottie ->

**Scottie executes:**

**STEP (1)**

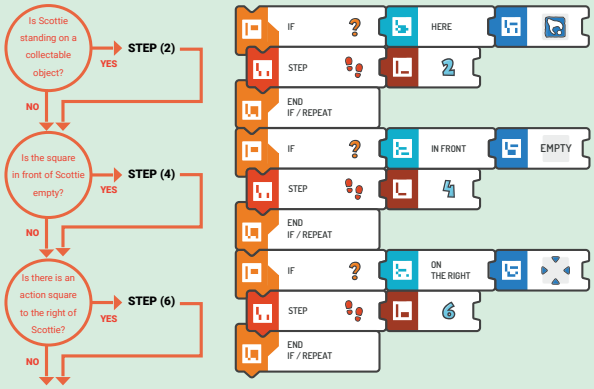
**PICK UP**

# INTRODUCTION:
# ELSE IF and MARKS (2/3)

The **ELSE IF** command lets you define alternative sets of commands **out of which only one set will be performed.**

**Without using**

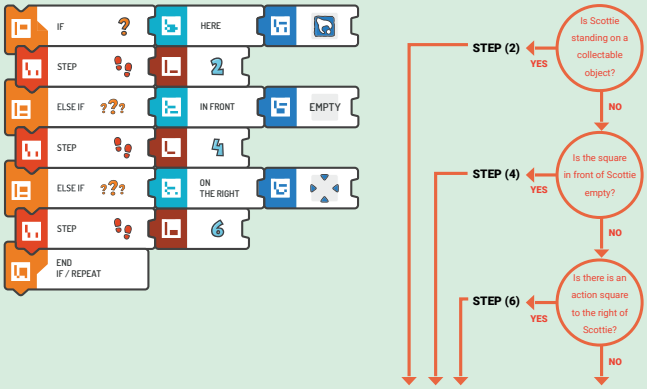
**ELSE IF**




Scottie will check the conditions one by one.
It's possible that Scottie will take as many as 2+4+6=12 steps!

**While using**

**ELSE IF**




Scottie will check the conditions one by one. Meeting any of those conditions excludes performing actions connected with the other conditions.
Scottie will take 2, 4, or 6 steps.

**REMEMBER**

**In this and the subsequent quests in this module you can use the REPEAT (FOREVER) tiles.**

The **REPEAT (FOREVER)** command is not introduced in the tutorial, but it's a more interesting and the most universal solution. You can read more about it in **the Module introduction (C)**.

# INTRODUCTION:
# ELSE IF and MARKS (3/3)

In this module, apart from the objects you already know, like **collectable objects**, **action squares** and **obstacles**, **Scottie will have a chance to interact with MARK - SQUARE** and **MARK - CROSS** that he will find on the workshop floor.



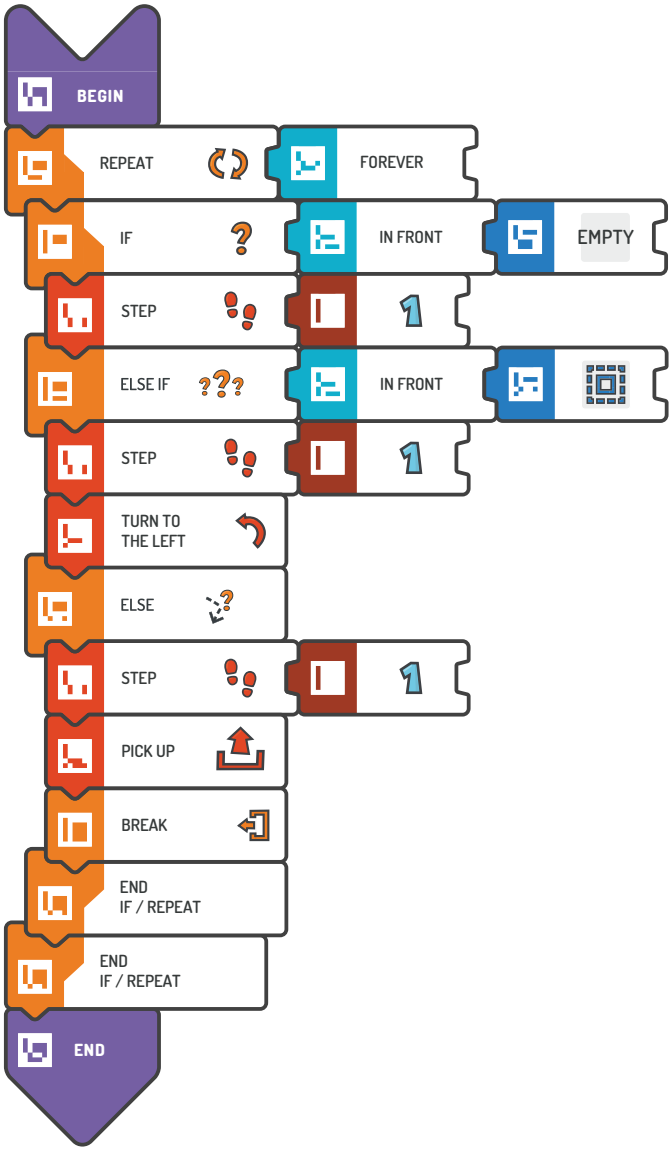
Workshop employees use the mark squares as orientation points to help organise the workspace. You will be able to use them as reference points to construct commands for Scottie with the use of **conditional statements IF and ELSE IF**.

Use these tiles to construct conditional statements in this quest.

Challenge **(In this quest you are supposed to use the ELSE IF tile)**:




Example solution:



Challenge **(In this quest you are supposed to use the ELSE IF tile)**:



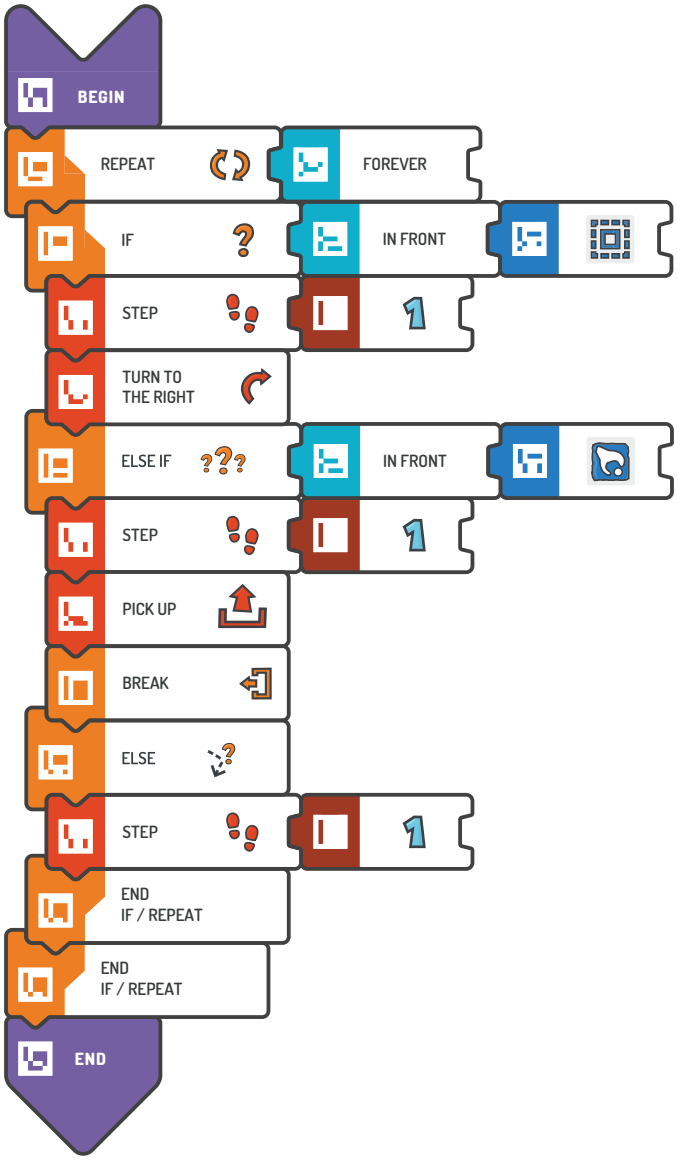

Example solution (in text format):

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < EMPTY) {
            STEP (1)
        } ELSE IF (IN FRONT < MARK - SQUARE) {
            STEP (1)
            TURN TO THE LEFT
        } ELSE {
            STEP (1)
            PICK UP
            BREAK
        }
    }
END
```

Challenge **(Use compound conditional statements in this quest)**:




Example solution:



Challenge **(Use compound conditional statements in this quest)**:



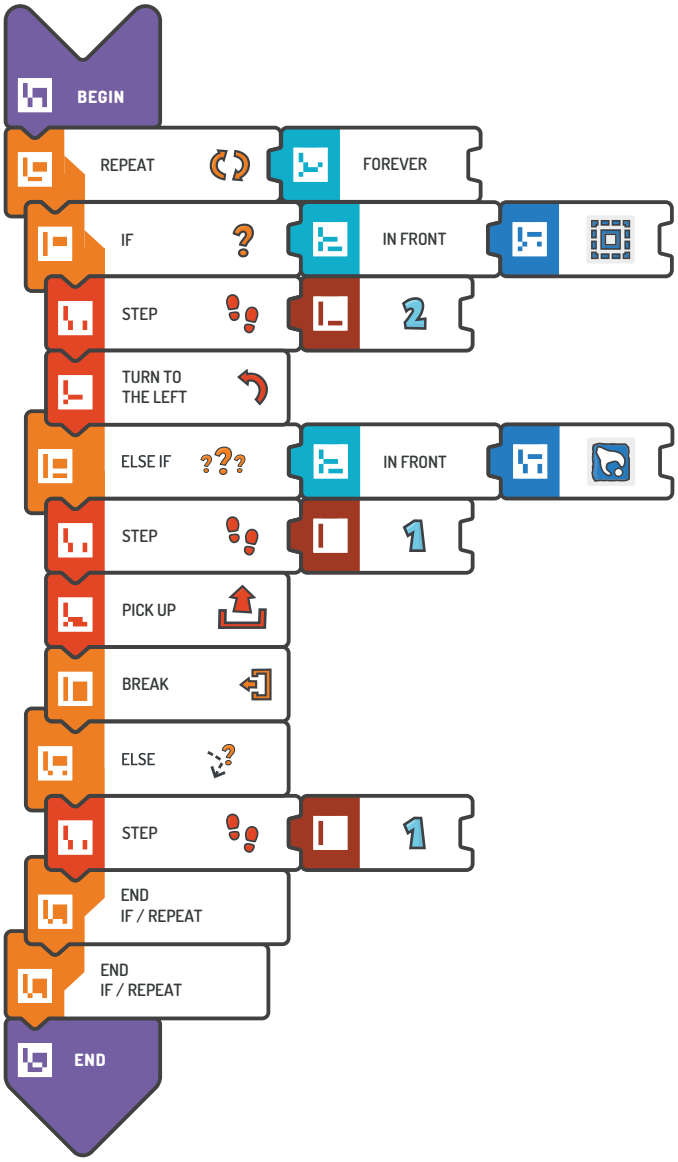

Example solution (in text format):

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < MARK - SQUARE) {
            STEP (1)
            TURN TO THE RIGHT
        } ELSE IF (IN FRONT < COLLECTABLE OBJECT) {
            STEP (1)
            PICK UP
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

Challenge **(Use compound conditional statements in this quest)**:




Example solution:



Challenge **(Use compound conditional statements in this quest)**:




Example solution (in text format):

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < MARK - SQUARE) {
            STEP (2)
            TURN TO THE LEFT
        } ELSE IF (IN FRONT < COLLECTABLE OBJECT) {
            STEP (1)
            PICK UP
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

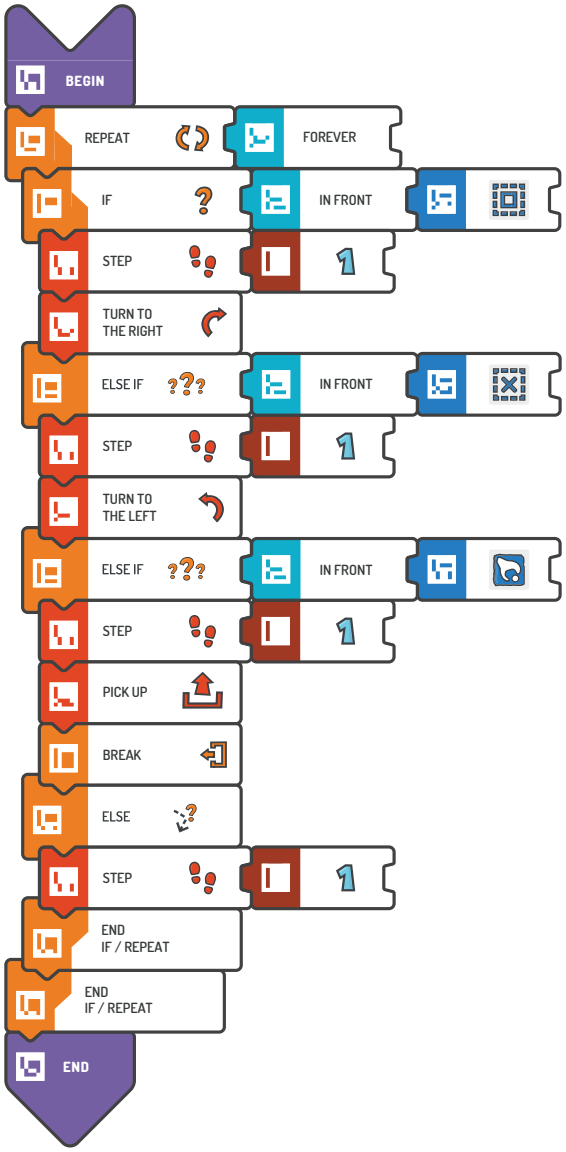
# HINT

You can use multiple **ELSE IF** tiles after the first **IF** in your code.

Challenge **(Use a conditional statement in this quest)**:




Example solution:



Challenge **(Use a conditional statement in this quest)**:



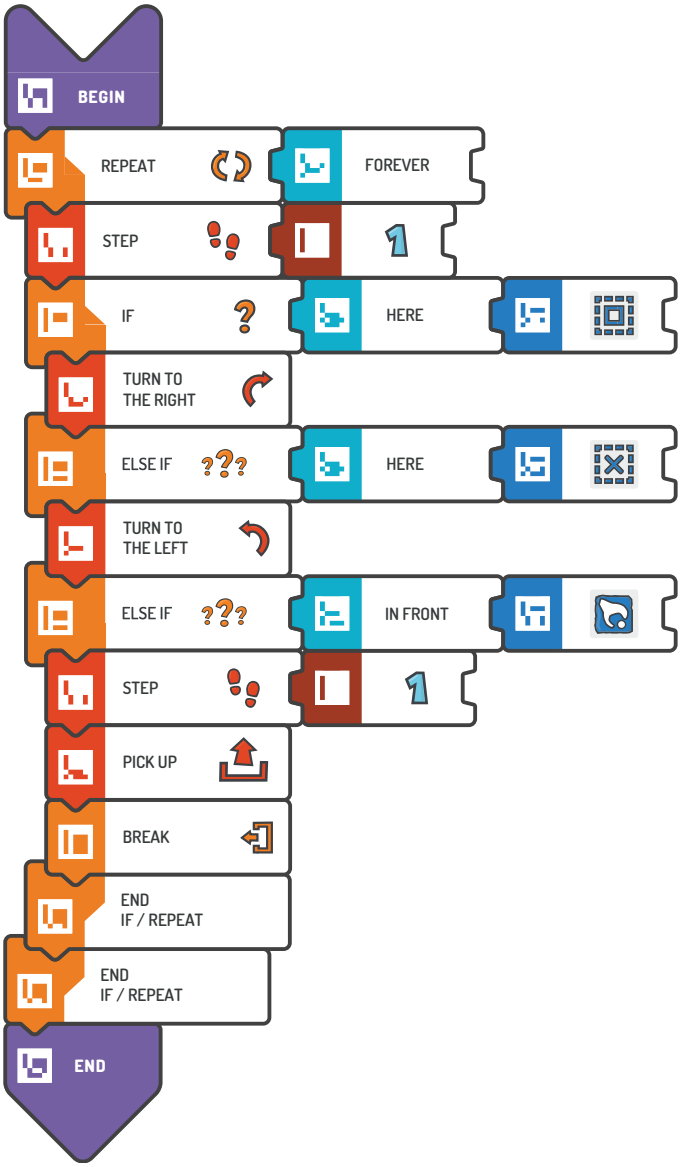

Example solution (in text format):

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < MARK - SQUARE) {
            STEP (1)
            TURN TO THE RIGHT
        } ELSE IF (IN FRONT < MARK - CROSS) {
            STEP (1)
            TURN TO THE LEFT
        } ELSE IF (IN FRONT < COLLECTABLE OBJECT) {
            STEP (1)
            PICK UP
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

Challenge **(Use conditional statements in this quest)**:




Example solution:



Challenge **(Use a conditional statement in this quest)**:



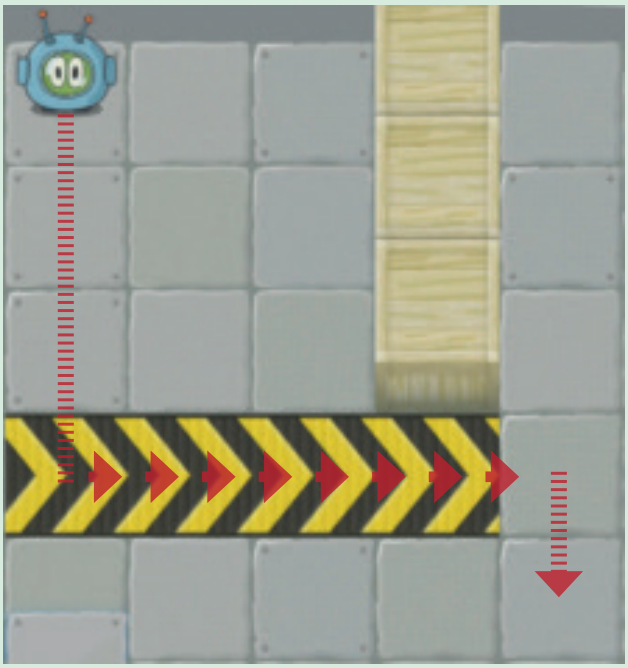

Example solution (in text format):

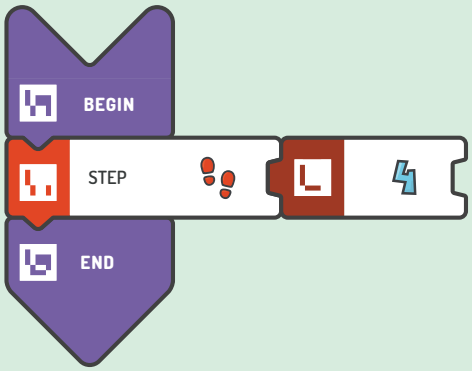
```
BEGIN
    REPEAT (FOREVER) {
        STEP (1)
        IF (HERE < MARK - SQUARE) {
            TURN TO THE RIGHT
        } ELSE IF
        (HERE < MARK - CROSS) {
            TURN TO THE LEFT
        } ELSE IF
        (IN FRONT < COLLECTABLE OBJECT) {
            KROK(1)
            PICK UP
            BREAK
        }
    }
END
```

# INTRODUCTION: conveyor belts

In this workshop you will find **conveyor belts**. Using them, Scottie can move across the board without the need to build complex programs.

Entering a conveyor belt **pauses execution of the program which will resume only after Scottie arrives at his destination.**

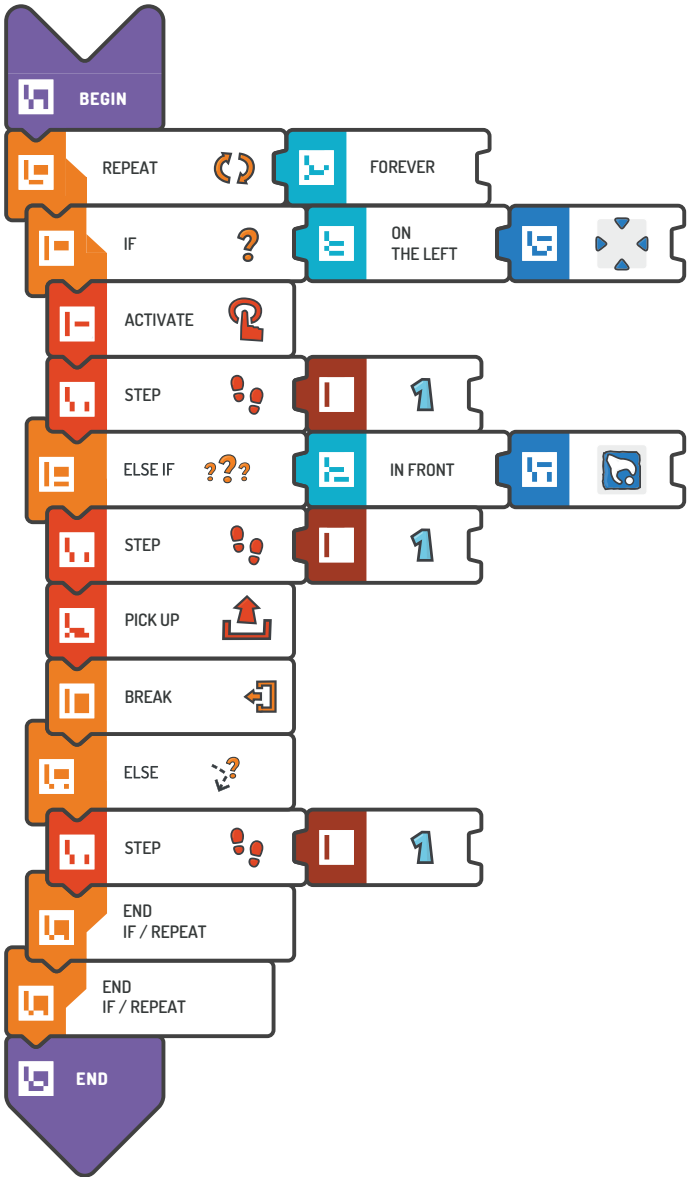




Challenge **(Use the conveyor belt to change your position. Turn it on by using the ACTIVATE tile)**:




Example solution:



Challenge **(Use the conveyor belt to change your position. Turn it on by using the ACTIVATE tile)**:




Example solution (in text format):

```
BEGIN
    REPEAT (FOREVER) {
        IF (ON THE LEFT < ACTION SQUARE) {
            ACTIVATE
            STEP (1)
        } ELSE IF (IN FRONT < COLLECTABLE OBJECT) {
            STEP (1)
            PICK UP
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

# INTRODUCTION:
# switches

Some conveyor belts are turned off. Tu turn them on, locate a switch in a workshop, approach it and use the **ACTIVATE** command.



**Remember!**
**In order for the ACTIVATE command to work, SCOTTIE mustn't be facing away from the switch.**

Challenge **(Use a conditional statement in this quest)**:




Example solution:



Challenge **(Use a conditional statement in this quest)**:



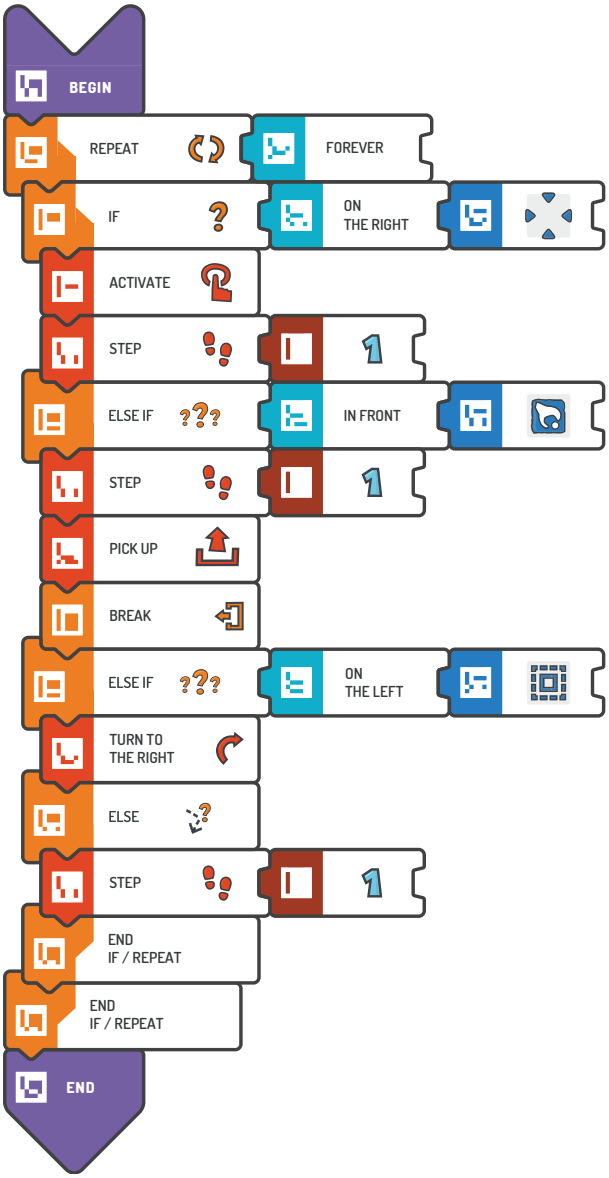

Example solution (in text format):

```
BEGIN
    REPEAT (FOREVER) {
        IF (ON THE RIGHT < ACTION SQUARE) {
            ACTIVATE
            STEP (1)
        } ELSE IF (IN FRONT < COLLECTABLE OBJECT) {
            STEP (1)
            PICK UP
        } ELSE IF (IN FRONT < OBSTACLE) {
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

Challenge **(Use a conditional statement in this quest)**:




Example solution:



Challenge **(Use a conditional statement in this quest)**:




Example solution (in text format):

```
BEGIN
    REPEAT (FOREVER) {
        IF (ON THE RIGHT < ACTION SQUARE) {
            ACTIVATE
            STEP (1)
        } ELSE IF (IN FRONT < COLLECTABLE OBJECT) {
            STEP (1)
            PICK UP
            BREAK
        } ELSE IF (ON THE LEFT < MARK - SQUARE) {
            TURN TO THE RIGHT
        } ELSE {
            STEP (1)
        }
    }
END
```