



## Topics: Computer Science

**Young Programmers** ages 4 – 12

☒ means kids can do this part alone!

## SHARE YOUR RESULTS!

@EiE\_org

#STEMbeyondSchool

## Robot Routes Game

In this offline activity, learners apply computational thinking while creating algorithms to move a robot from start to finish. Young programmers have fun as they practice decomposition and debugging.

### How Much TIME

**Setup:** 15 minutes

**Activity:** 30 minutes doing, 10 minutes thinking and talking

### What You NEED (download from [eie.org/sbs-resources](http://eie.org/sbs-resources))

- Mini Map
- Robot Algorithm Sheet
- Robot Crown
- Floor Cards
- Markers, colored pencils, or crayons
- Tape or stapler
- Game piece or small figurine, approximately 1" wide, with obvious front and back
- Materials to create a grid on the floor






### Set It Up

- Create a 6 x 6 grid of 10" x 10" squares on the floor using masking tape, chalk, paper, or spot markers.
- Print Mini Map.
  - No printer? Draw a 6 x 6 grid of 1" squares on blank paper.
- Print Robot Algorithm Sheet.
  - No printer? Have learners write their own code symbols on blank paper.
- ☒ • Print and cut out Robot Crown. Tape or staple ends together to make a headband.
  - Customize your crown by coloring or decorating it.
  - No printer? Draw a robot on paper, cut it out, and tape it to a headband.
- Print Floor Cards. Place the shapes, starting position, end goal, and walls on the floor grid as they appear in the Mini Map.
  - No printer? Use household items to represent each object (e.g., flag for starting position, cup for end goal, pillows for walls).



## What to DO

1. Explain that in this activity sometimes learners will play the role of the Programmer and sometimes they will be the Robot. In part 1, they will plan their route on a Mini Map. In part 2, they will try this route out on the floor map.
2. Explain that when learners are in the role of the Programmer they will use code symbols to tell the Robot what to do.
3. Explain that when learners are in the role of the Robot they have to listen closely to what the Programmer says and follow the instructions exactly. Remind learners that a robot has a computer for a brain. Computers do exactly what they are told to do—nothing more and nothing less.
4. Introduce the code symbols that will control the Robot. Demonstrate the meaning of each symbol by moving the figurine around the Mini Map. Note that the Robot cannot move through a wall.

<b>START</b>	<b>Start:</b> Required to begin the algorithm. (Functions like an “ON” button.)
	<b>Arrow:</b> Move forward one space on the floor map. (Note that all spaces are of equal size.)
	<b>Square:</b> Rotate your body to face toward the square side of the map. (Do not move forward.)
	<b>Triangle:</b> Rotate your body to face toward the triangle side of the map. (Do not move forward.)
	<b>Circle:</b> Rotate your body to face toward the circle side of the map. (Do not move forward.)
	<b>Half-circle:</b> Rotate your body to face toward the half-circle side of the map. (Do not move forward.)

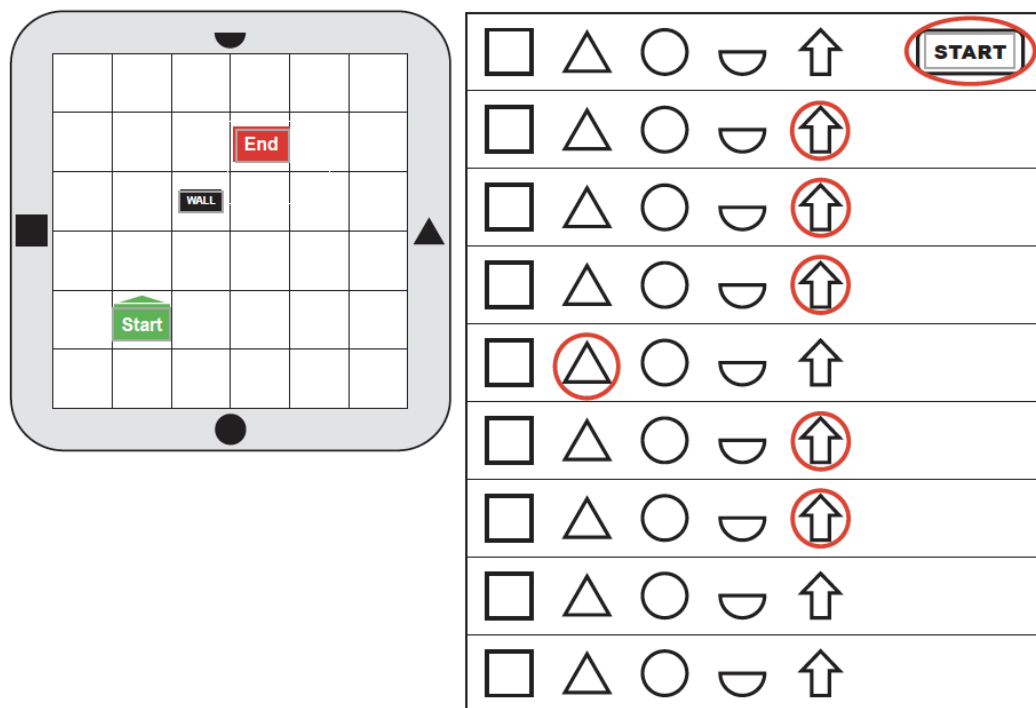
5. Review the Robot and Programmer roles. If needed, show learners an example (see following page):

- **Programmer**

- Planning: Give instructions to the Robot, one step at a time. Record each step by circling **one** code symbol on the Robot Algorithm Sheet. (Be sure to select Start as the first step!)
- On the Floor: Read the code instructions to the Robot. Be sure to read exactly what the Robot Algorithm says.

- **Robot**

- Planning: Move the robot playing piece around the Mini Map based on instructions from the Programmer.
- On the Floor: Wear the robot crown and stand on the floor map. Move around the map like a robot, doing exactly what the algorithm says to do. Stay on the map and do not move diagonally.



- ✓ 6. Invite learners to play as many rounds of the game as they would like. The online resources include two Mini Maps, and a blank grid where learners can create their own arrangements!

### Think and Talk

Use the questions and example answers below to discuss the Robot Routes Game with learners. You may also want to use this opportunity to review the shapes around the map. Ask children to name the shapes. Have them identify attributes like number of sides each shape has or the number of angles inside each shape.

- Discuss a time when you had to debug (find and fix errors).  
*Example answers: I debugged when my robot ran into a wall, I debugged because I didn't use enough arrows to go forward.*
- How do humans tell computers what to do?  
*Humans design step-by-step instructions for a computer; humans give instructions to the computer in code, a language that the computer can understand.*
- Is there one correct way to solve the problem?  
*No, there are many paths from the start to the end, so there are many different ways to give instructions to the robot.*
- What else could you tell a robot to do?  
*Example answers: Make my bed, brush my teeth for me, help someone carry their groceries home, pet my dog over and over again.*



## Computer Science Background

Scientists and engineers around the world develop robots to help solve different problems. Robots help with jobs for humans such as lifting heavy loads, handling toxic substances, and performing repetitive tasks.

A set of step-by-step instructions for accomplishing a task is sometimes referred to as an *algorithm*. **Algorithms** are usually expressed in ways that are easy for humans to understand, through things like flow charts, pictures, or descriptive words. When a programmer writes an algorithm in a language that a computer can understand, it is called a **program**.

**Decomposition** is an important skill in computer science. Decomposing involves breaking down a complex problem into more manageable parts. In the Robot Routes Game, the Programmer must decompose the large task of moving the Robot to an end location into smaller tasks such as moving forward and turning.

Another important practice in computer science is debugging. Computer scientists assume that algorithms will not be perfect the first time they test them out. For this reason, frequent opportunities for testing and debugging are always included in plans for computing projects. As they **debug**, programmers find and fix errors (also called *bugs*) in their programs. Trying things out, seeing how the computer operates on the instructions, being surprised by what happens, and then fixing instructions are all normal parts of the debugging process. Programs are rarely correct the first time.