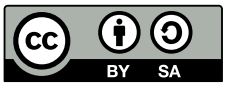
First Steps in Robotics with the Thymio Robot and the Aseba/VPL Environment

Answers to the Exercises

Version 1.5.1

Moti Ben-Ari and other contributors



1 Your First Robotics Project

No exercises.

2 Changing Colors

2.1: When mixing the three colors of the Thymio robot, each color can have an intensity between 0 (off) and 32 (on maximum). Therefore, there are $33 \times 33 \times 33 = 35,937$ different colors. If we consider only the extremes of intensity off or on maximum, the colors are: off (all colors off), white (all colors on), red, green, blue (one color on), yellow (red and green on, blue off), cyan (blue and green on, red off), magenta (red and blue on, green off).

3 Let's Get Moving

3.1: The robot should be able to stop. The maximum speed of the Thymio is 20 centimeters per second. Since the sensors are sampled 10 times a second, the robot can move at most 2 centimeters before it stops. That should be sufficiently fast to prevent the robot from falling off the table, but be prepared to catch it just in case!

There are no ground sensors at the back of the Thymio robot, so that the time that the sensors in the front detect the edge of the table, the entire body of the robot has moved beyond the edge and fallen off.

4 A Pet Robot

4.1: Replace the first two event-action pairs with the pairs shown in Figure 1. The first pair turns both motors on when all the sensors do *not* detect an obstacle; the second pair stops the motors when the ground sensors detect the edge of the table.

Program file **likes-and-stops.aesl**

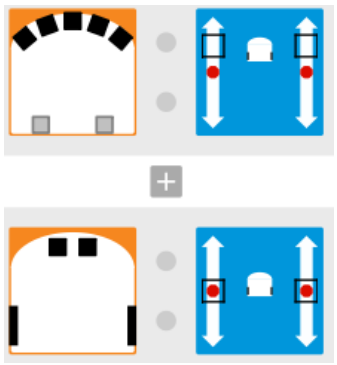


Figure 1: Stop at the edge of the table

4.2: An event-action pair is run when its event occurs. Events like touching a button occur when this external event occurs. Other events, like sampling sensors, occur at fixed intervals such as 10 times a second. When the sensors are sampled, all events for the sensors occur "at the same time." The computer in the robot cannot actually run all the event-action pairs simultaneously; instead, the event-action pairs are run one-by-one *in the order* that they appear from top to bottom in the program.

In the program in Figure 1, the horizontal sensors do not detect an obstacle and keep the motors running; *then* the ground sensors detect the tape and stop the robot.

In the program in Figure 2, the ground sensors detect the tape and try to stop the motors, but the second event-action pair keeps the motors on before they have a chance to stop.

Program file **likes-changed-order.aesl**

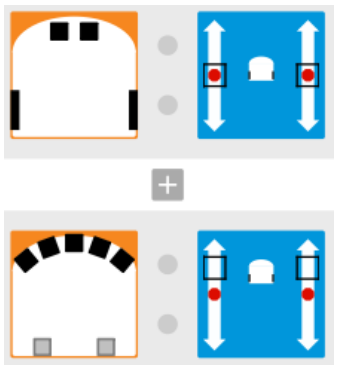


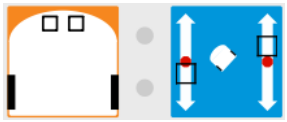
Figure 2: Changing the order of the event-action pairs

4.3:

- Use sensors 1 and 3: The robot is less sensitive to the presence of your hand on the side. You will have to move it closer to the center.
- Use both sensors 0 and 1 to turn the robot left and both sensors 3 and 4 to turn the robot right: The robot senses a wider area in the front and side. You don't have to be as careful in placing your hand.
- Add event-action pairs for the rear sensors 5 and 6: You can now cause the robot to turn by placing your hand near the back of the robot.

5 The Robot Finds Its Way by Itself

5.1: The event-action pair



will cause a gentle left turn when both ground sensors detect a lot of light. If you increase the speed, the robot might run too far off the line and not detect it when turning. If the robot gets to the end of the line it will also make a gentle left turn.

Program file **find-line.aesl**

5.2: The robot will, of course, move *away* from the line.

5.3:

- Gentle turns are easier to follow;
- Sharp turns are harder to follow;
- The robot might not return to the line before the next turn is encountered;
- Wider lines make it less likely that the robot will run off the line;
- Narrow lines make it more likely that a small deviation will cause the robot to run off the line. Therefore, there will be frequent turns and the movement will be jerky.

5.4:

- If the ground sensing events are more frequent, the robot will be more likely to respond to running off the line; if they are less frequent, it might run completely off the line before detection occurs.
- If the sensors are further apart a wider line will be needed but it is more likely that leaving the line will be detected before running off it completely; the opposite is true for sensors that are closer together.
- If there are more sensors, the robot can be more precise in its movements: gentle turns if only one sensor detects the floor and sharper turns if more than one sensor detects the floor.

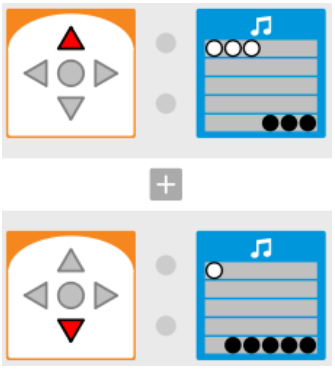


Figure 3: Morse code

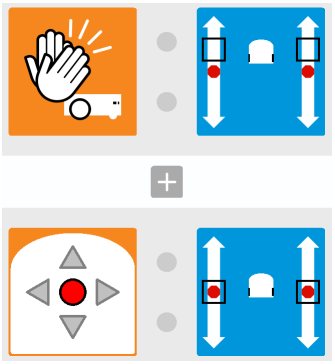


Figure 4: Clap to start the motor

6 Bells and Whistles

6.1: In Morse code, a dash is three times longer than a dot. The actions in Figure 3 use three white notes at the highest tone for the dash and one for the dot. Since there must be exactly six tones, we fill out the tune with short notes at the lowest tone which will not be heard as well as the highest tones.

Program file **bells-morse.aesl**

6.2: The program in Figure 4 turns the motors on when the clap event occurs and turns them off when the center button is touched. This may not work because the

Program file **clap-start.aesl**

For the opposite behavior, it seems obvious that the program in Figure 5 should work, but it doesn't. The reason is that the event that causes the motor to stop is not specifically clapping; *any* loud noise will cause the event to occur. The motors make so much noise that as soon as they start, the loud-noise event occurs and the motors stop.

Program file **clap-stop.aesl**

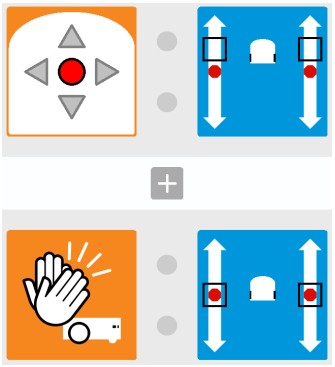


Figure 5: Clap to stop the motor

6.3: The program has two event-action pairs (Figure 6): one to move the robot when a button is touched and the other to stop it when a tap event occurs.

Program file **bump.aesl**

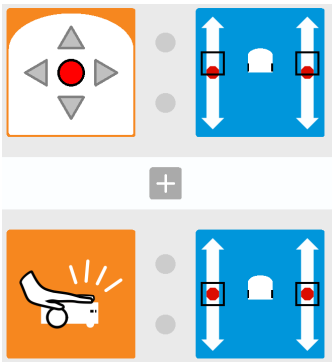


Figure 6: Stop the motor when the robot bumps into a wall

7 A Time to Like

7.1: See Figure 7. When the front button is touched the motors are turned on and a three-second timer is started. When the timer expires, the motors are reversed. Finally, when the center button is touched, the motors are stopped.

Program file **run-three-seconds.aesl**

8 States

8.1: See Figure 8 There will be two states: state *left* when the robot turns to the left and state *right* when the robot turns to the right. Initially, touching the forward button in the initial state (off, off, off, off) sets a one-second timer and state *left*. When the timer runs down

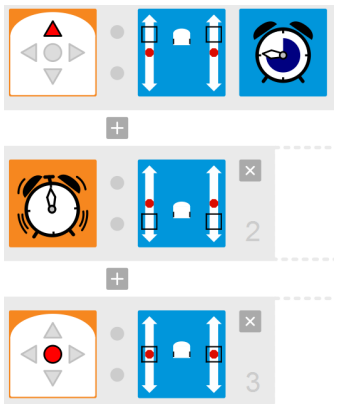


Figure 7: Run three seconds and reverse

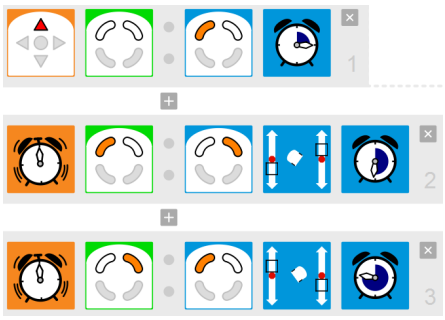


Figure 8: Dancing

in state *left*, the robot turns to the left, (re-)sets a timer for two seconds and goes to state *right*. When the timer runs down in state *right*, the robot turns to the right, (re-)sets a timer for two seconds and goes to state *left*. These two behaviors are run repeatedly, so you will have to click  to stop the robot.

Program file **dance.aesl**

8.2: I was not able to solve this problem. I had assumed that it would be possible to detect the event when *one sensor* no longer detect the line *before* the event when *both sensors* no longer detect the line. This would enable the program to identify and remember from which side the robot ran off the line. This proved impossible even at low speeds.

The best I could do is to have the program remember on which side the robot last ran off the line, even if that occurred much earlier.

Program file **follow-line-and-find.aesl**

The robot does have one cool behavior: When it gets to the end of the line, it slowly turns completely around and heads back the other way along the line!

State *left*  remembers that the robot left the left side of the line and state *right*  remembers that the robot ran off the right side of the line (Figure 9). When both sensors fail to detect the line, the robot turns in the direction indicated by the current state (Figure 10).

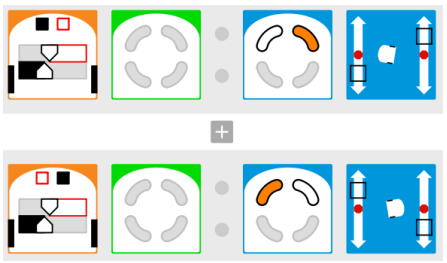


Figure 9: Change state to remember direction

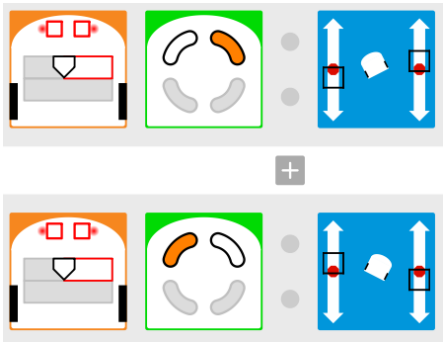


Figure 10: Search to the left or the right

9 Counting

9.1: You can count to four because there are four quarters of the state each of which can be set **on** to count an object.

9.2: The lower left quarter of the state icon will be used to represent the number of 4's in a binary number. If all quarters are off, the number represented is 0; if all quarters are on, the number represented is 4+2+1=7. Eight event-action pairs are needed, one for each transition between n and $n + 1$ (modulo 8).

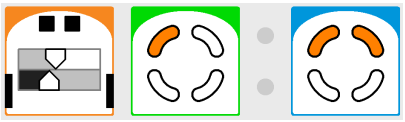
Program file **count-to-eight.aesl**

9.3: There are four quarters in the state. We can use one quarter to count each of the number of 1's, 2's, 4's and 8's. Therefore, we can count from 0 to 8+4+2+1=15.

9.4: This program is a mirror image of the program for adding. For example, if the binary number is xyz1, then subtracting one gives xyz0, whatever the values of xyz. If the number is xyz0, you will have to borrow a binary digit, and there will be different event-action pairs depending on the value of xyz. Finally, subtracting one from 0000=0 gives 1111=15 in cyclic arithmetic.

Program file **subtraction.aesl**

9.5: An event-action pair is needed to count each strip of tape. For example, the following pair



changes the count from 2 to 3 when a tape is detected.

Program file **count-tapes-four.aesl**

10 Accelerometers

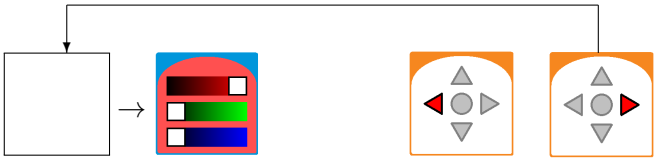
10.1: VPL does not allow identical events in more than one event-actions pair, unless the parameters are different. For this block, choosing a different angle for the red segment makes the events different.

There are seven different positions of the red segment between the vertical and horizonal on one side (altogether 13 segments). Therefore, each segment represents $90/6.5 \approx 14$ degrees. (The vertical segment is half within the left quarter circle and half within the right quarter circle, so we divide 90 by 6.5 and not 7.)

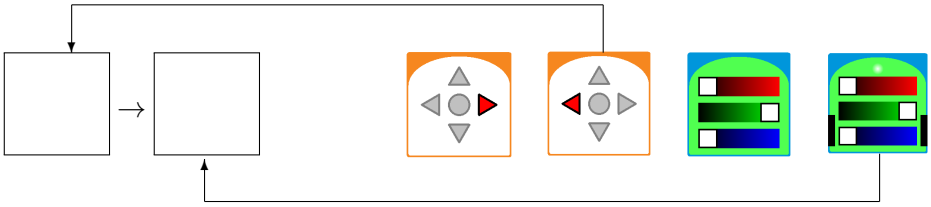
Answers to the Parsons Puzzles

1. When the right button is touched the bottom red light is turned on.

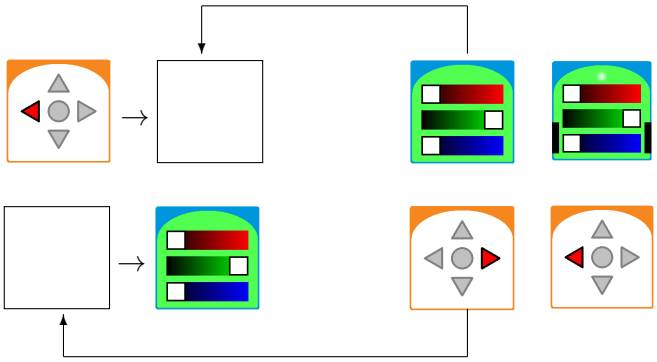
2. When the right button is touched the top red light is turned on.



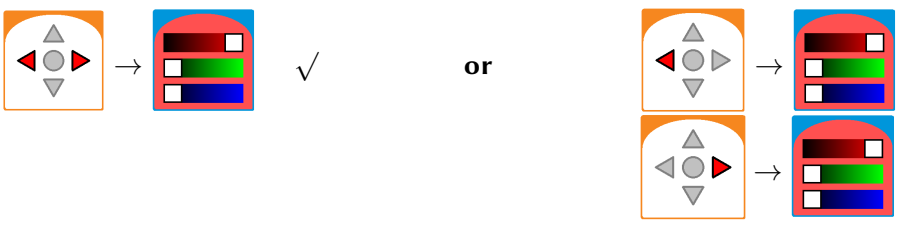
3. When the left button is touched the bottom green light is turned on.



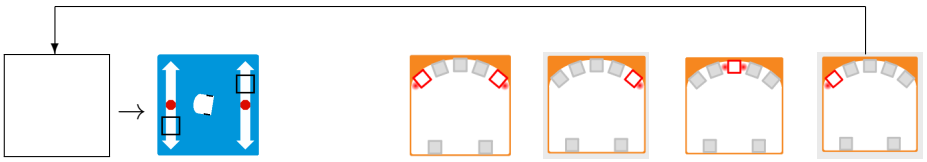
4. When the left button **or** the right button is touched, the top green light is turned on.



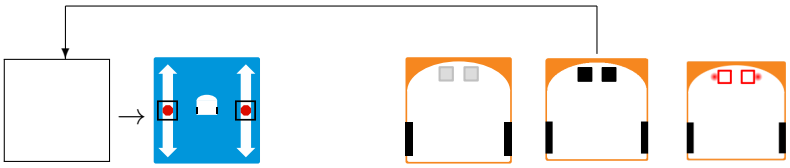
5. When **both** the left button **and** the right button are touched, the top red light is turned on. Select one of the following two programs:



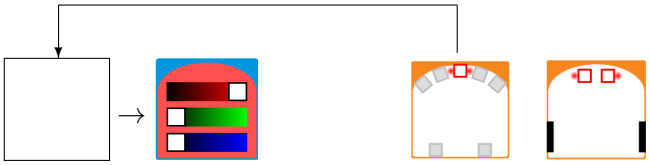
6. If an object is detected **only** by the leftmost sensor, turn left.



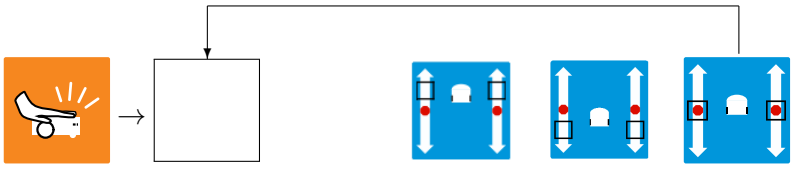
7. Stop the robot when the end of the table has been reached.



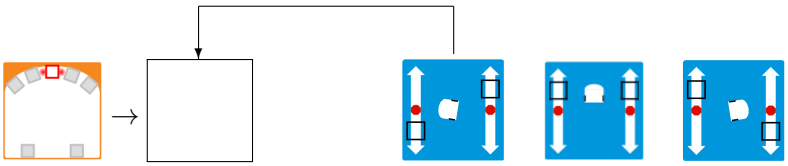
8. When the robot detects a wall, the top red light is turned on.



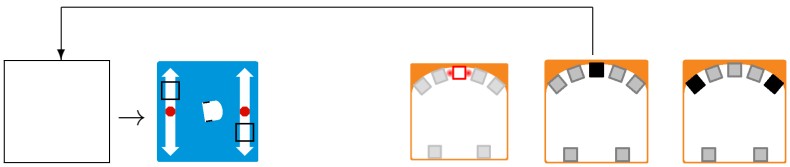
9. When the robot hits the wall, the motors are turned off.



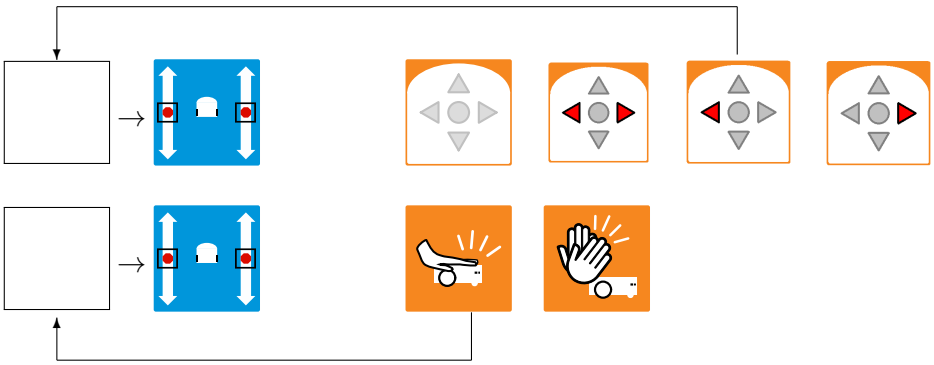
10. The robot turns to the left if there is an object in front of the center sensor.



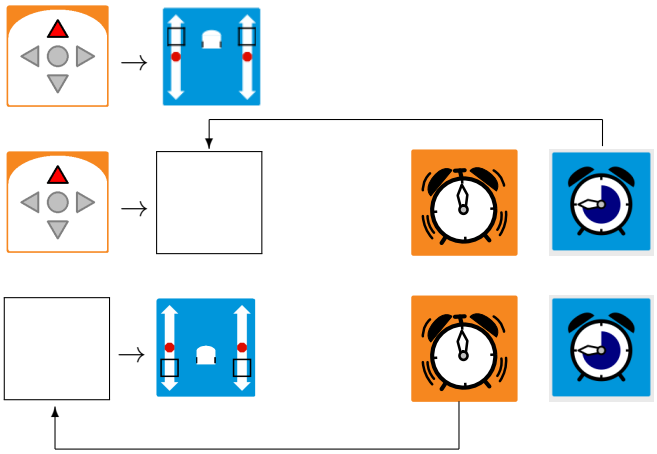
11. The robot turns to the right if there is **no** object in front of the center sensor.



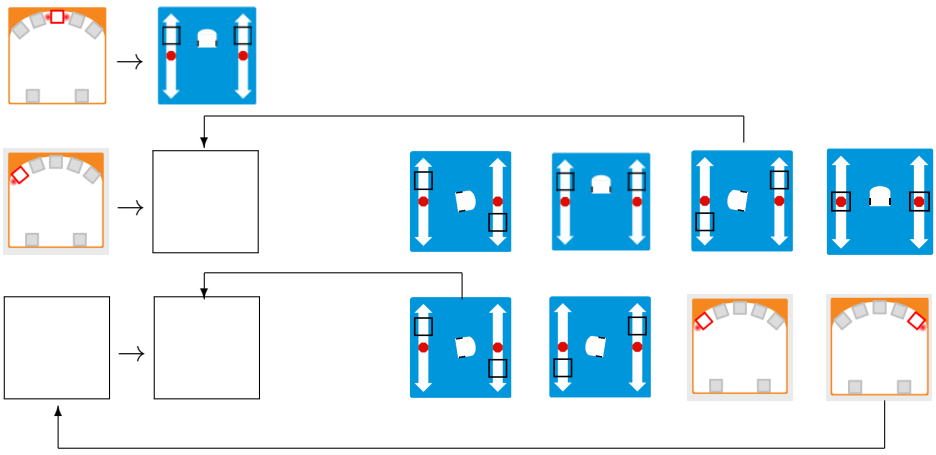
12. The motors are turned off when the left button is touched **or** if the robot is tapped.



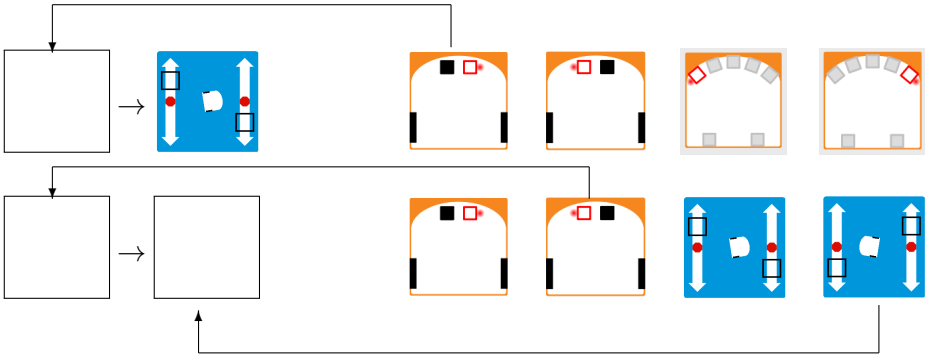
13. When the forwards button is touched, the robot moves forward for three seconds and then moves backwards.



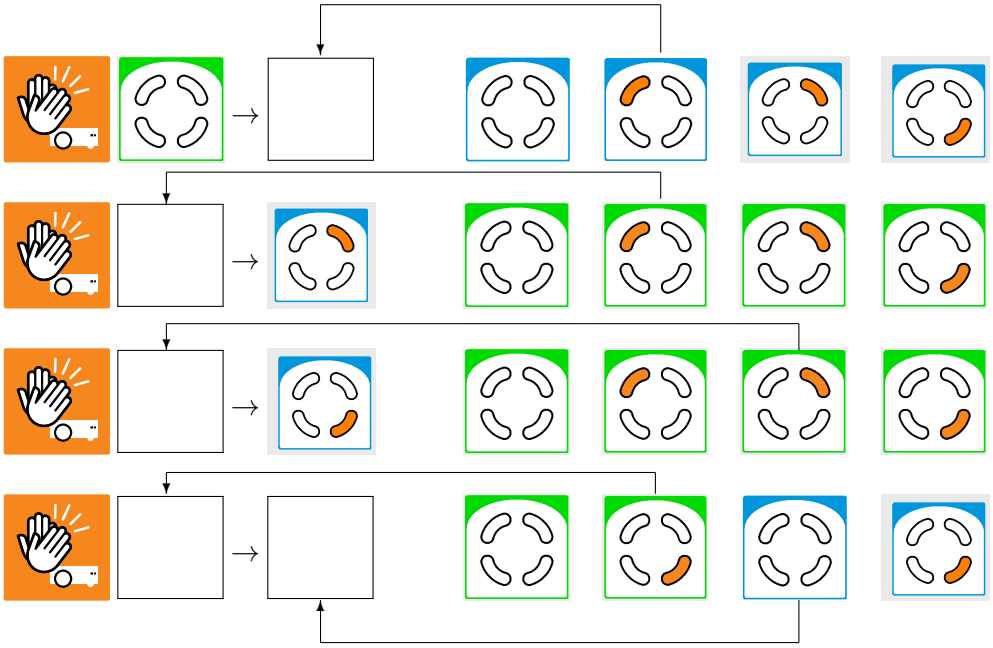
14. The robot moves towards an object that is detected by its left, right or center sensor.



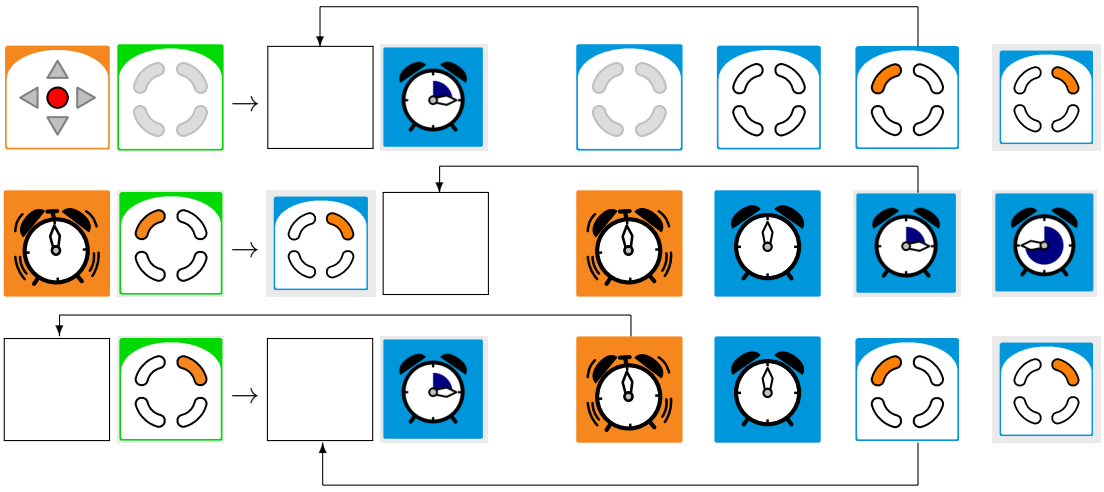
15. The robot is following a line on the floor. It turns left if it no longer detects the line in its right sensor and it turns right if it no longer detects the line in its left sensor,



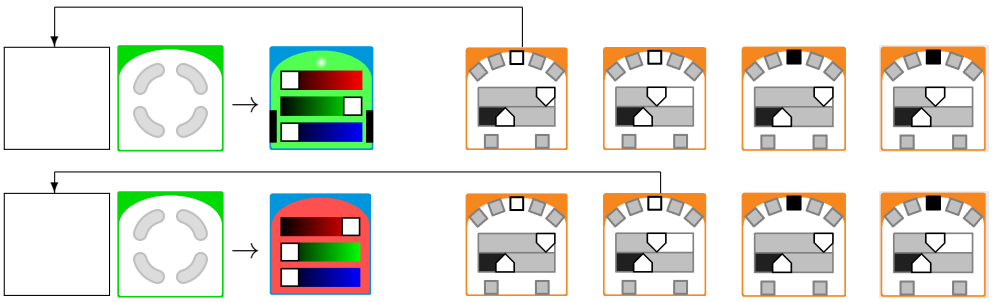
16. The robot counts 0,1,2,3,0,1,2,3, ..., whenever it detects a clap event.



17. When the center button is touched, the right front and left front circle lights turn on and off alternately at one-second intervals.



18. The bottom light of the robot turns green when it detects an object is far away from it and the top light of the robot turns red when it detects an object is close to it.



19. Tilt the robot on its left side; the top light turns blue and the bottom light is turned off. Tilt the robot on its back; the top light is turned off and the bottom light turns yellow.