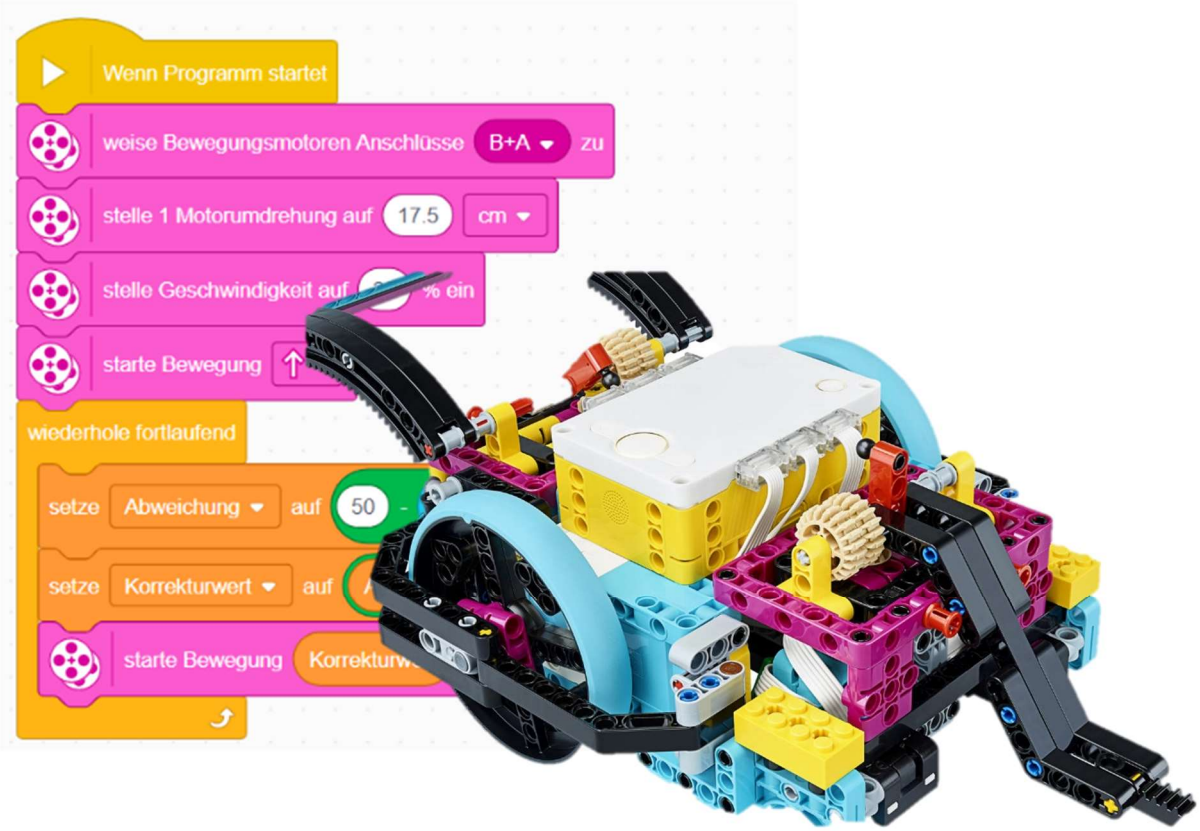
SWITZERLAND

Robotics for Beginners

Learning Dossier



by Vera Hausherr
Version 1.2 /05.02.2023

Copyright

Trademarks

Technical Note

This Learning Dossier was written on the basis of version 3 of the SPIKE App. There may be minor differences between this version and previous (or future, not yet released) versions.

Part 1: What Is Robotics?

1: Machines, Computers, and Robots

Learning objective:

- ☐ I k now the difference between machines, computers, and robots.

Fill in the table below with these devices. Can you think of other examples?








Machine	Computer	Robot

Remember:
A **machine** can work, but not think.
A **computer** can think but not work.
A **robot** can think and work.

2: Which Parts does a Robot Need?

Learning Objectives:

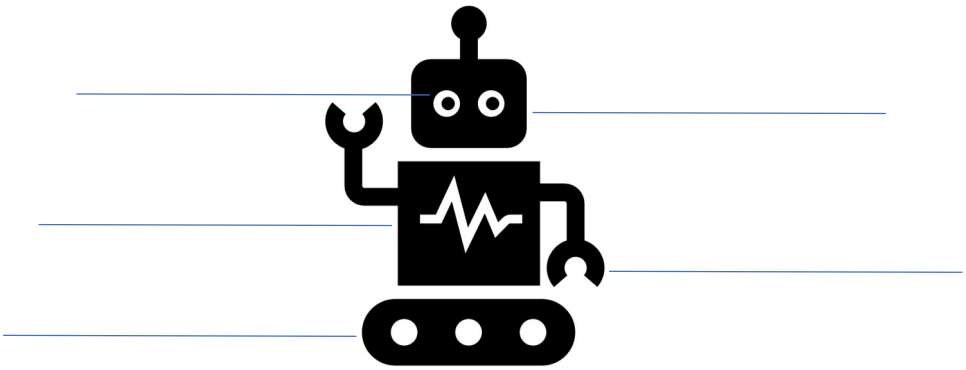
- ☐ I know which parts a robot needs.
- ☐ KI know why sensors are important.
- ☐ I know the parts of the SPIKE Prime robotics kit.

In the precious task you saw that there are many different kinds of robots. But all of these have some things in common:

2.1 Overview

Task:

Write the following words in the correct place: *Locomotor system, "brain", sensors, power supply, tools.*



2.2 What do you need these parts for?

Task:
Write the words form task 2.1 into this table.

Perception of the environment (e.g. recognizing a color, or measuring a distance)	
Moving around (e.g., walking or driving)	
Controlling the robot (deciding what it needs to do next)	
Work (e.g., collect an object)	
Getting energy for moving and working	

2.3 Which parts are there in the SPIKE Prime robotics kit?

Task:
Complete the table below.
You will need these words: Movements, "brain", buttons, colors, dark, driving, far, lifting, light matrix, switch

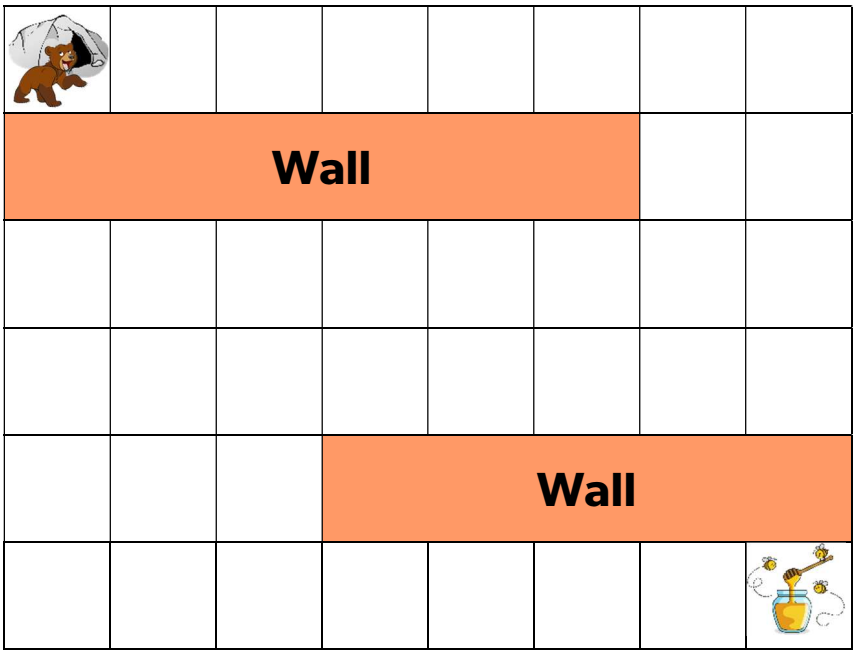
	**Hub with rechargeable battery and gyro sensor**	The Hub contains the _____ oft he robot. On the white surface you find the _____ and the _____. And the gyro sensor will detect if the robot has been turned or tilted.
	**Motors**	You need motors for all _____. The smaller ones are great for _____, while the larger ones are better for _____.
	**Color sensors**	You can use color sensors to scan for _____, and to determine how _____ or _____ something is.
	**Distance sensor**	You can use the distance sensor to determine how _____ away an object, an obstacle or a wall are.
	**Power sensor**	A power sensor has a button that you can use like a _____. You can also measure how hard the button is being pressed.

3: How Does the Robot Know What It Has To Do?

Learning objectives:

- ☐ I know the terms "programming" and "code".
- ☐ I can give a robot simple instructions using normal language.

The "bear robot" must get from its cave to the honeypot. It cannot climb over the walls. And it can only move left-to-right, right-to-left, top-to-bottom, and bottom-to-top (not diagonally).



Tasks:

1. Draw the best path for the bear robot into the map.
2. Explain to your partner how the bear robot finds its way.

Remember:
You need to give your bear robot very clear instructions so it can find the way.

That is called _____.

The list of instructions are called _____.

Task: Complete this code so the bear robot will find the way from the cave to the honeypot. (Possible words: "*right*", "*left*", "*up*", and "*down*», as well as numbers)

- Leave the cave and go _____
- Move straight for _____ squares.
- Turn _____.
- Move straight for _____ squares.
- Turn _____.
- Move straight for _____ squares.
- Turn _____.
- Move straight for _____ squares.
- Turn _____.
- Move straight for _____ squares.

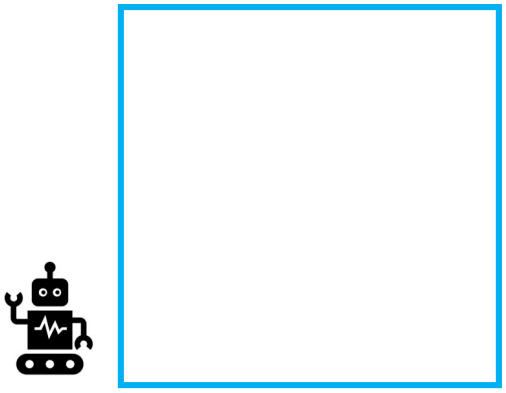
Teil 2: Introduction to Coding

4: Simplifying the code: Loops

Learning objective

- ☐ I know what a loop is and what you need it for.

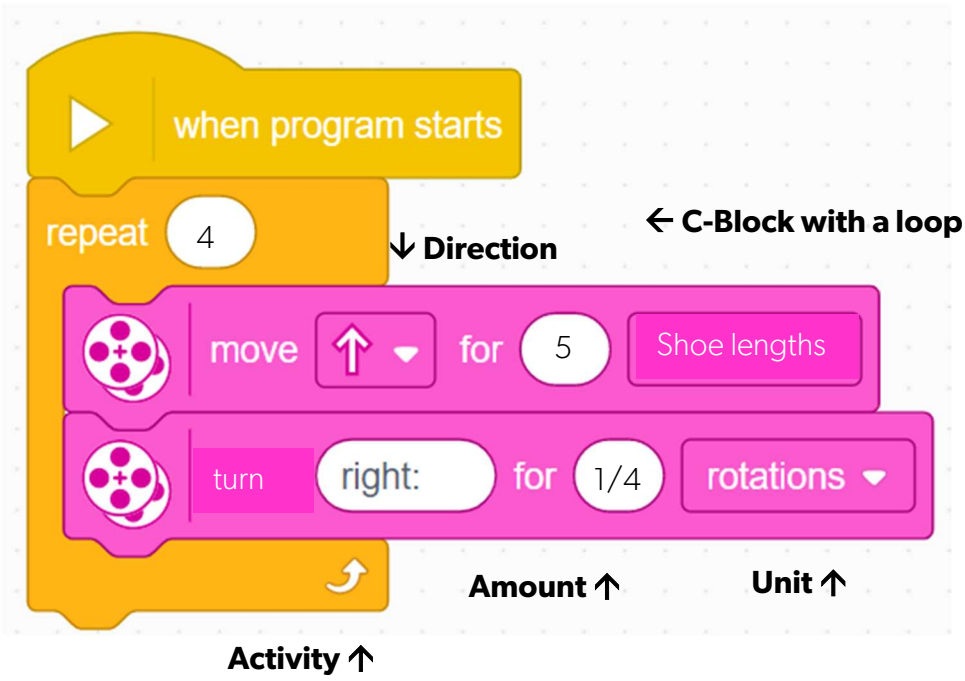
Tell your robot to walk a path that looks like a perfect square.



Tasks:

1. Your partner is the robot. Give her/him instructions how to walk. The side of the square must be exactly for shoe-lengths long.
2. Try to write a very short program. It must be exactly 3 lines of code.

3. Now we write the code very similar to what it will look like for our Lego robot. This is not yet real code, but it is designed to give you a general idea.



Think: Why is it useful to use such repeat blocks?

Remember:

A repeat block is called a "loop".

And here's a golden programmer's rule for you: **Never write something twice if you can write it once and use a loop!**

4: Can we make our robot a little smarter? Conditions

Learning objectives

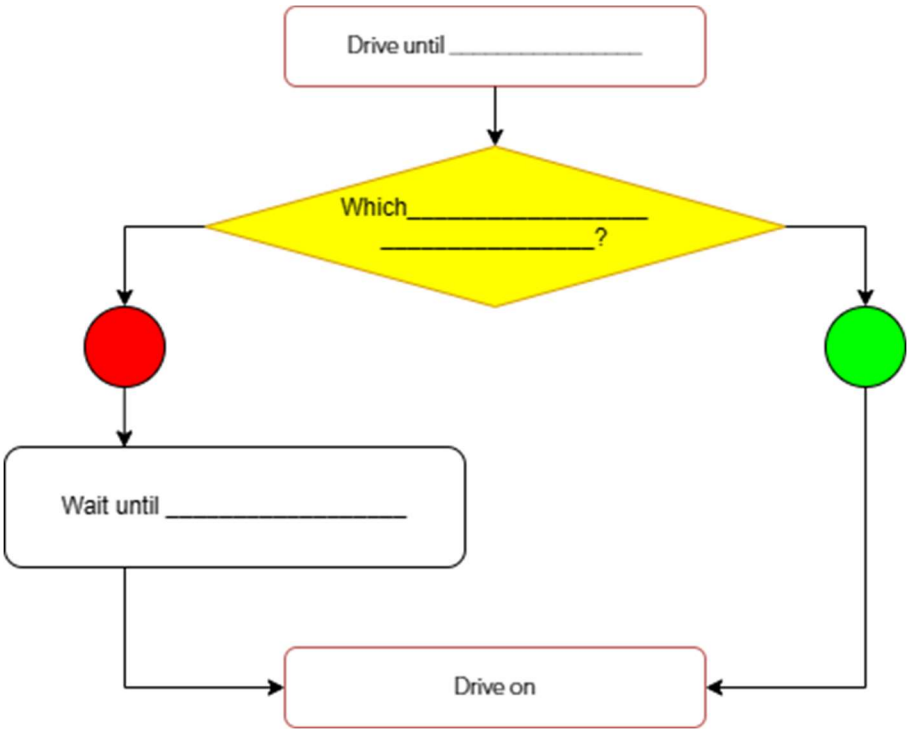
- ☐ I know that a robot always must make decisions.
- ☐ I can explain a simple condition.

Your robot can do some really nice things with such simple code as we used for the bear-robot or the square. But the robot didn't really need to think for itself- But we want to have a smart robot that can think for itself and make decisions on what to do.

Example: Your self-driving car robot must drive. When it reaches a yellow line it must check which color the light is. If the light is red the robot must stop and wait until the light is green. When the light is green it needn't stop.

Task:

Complete the code, which is represented by a flow chart here.



These situations are called **Decisions** or **Conditions**. They are the most important thing in coding!

Teil 3: A SPIKE Prime Robot

5: First Steps with a SPIKE Prime

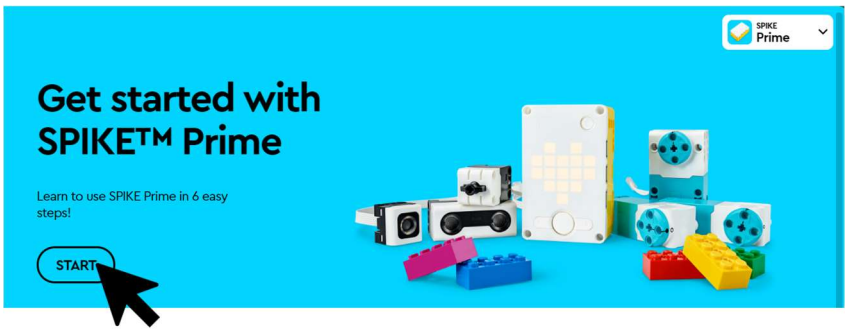
Learning objectives

- ☐ I know the parts of a SPIKE Prime.
- ☐ I know the SPIKE App
- ☐ I can write my first code!

Now we can start building and programming your robot!

Open the **SPIKE App** on your tablet or computer. Then click on

We start with **"Get Started with SPIKE™ Prime"**. Click the **Start** button.



This opens a tutorial with 6 tasks with which you can get to know the Hub, the motors and the sensors, and write your first little programs.

Task:

Complete all 6 tasks. If you are really quick you can also experiment with these extra tasks.

Extra task 1 (The Light Matrix): Can you also make the hub show an arrow? How can you turn on and off all pixels at once? Can you change the brightness oft he pixels?

Extra task 2 (The Motor): Can you make the motor to spin the other way around? (hint: put a small lego piece on the rim of your wheel to make the spinning motions better visible!)

Extra task 3 (The Color Sensor): Try the same thing with different colors. Use Lego bricks for testing as well as other objects with similar colors. Try to find out which colors the sensor recognizes easily and which ones are tricky, and how close the sensor has to be for the most reliable color recognition.

Extra task 4 (The Distance Sensor): Can you change the distance to 15 cm? Can you play a different beep?

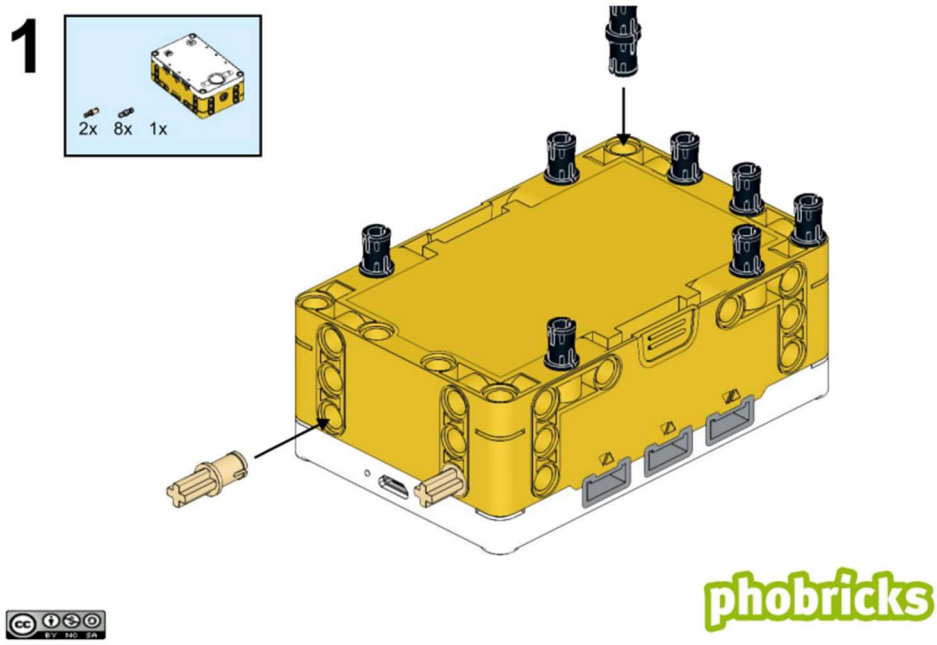
(No extra task for the force sensor because the tutorial only allows for a very limited pallet of coding blocks!)

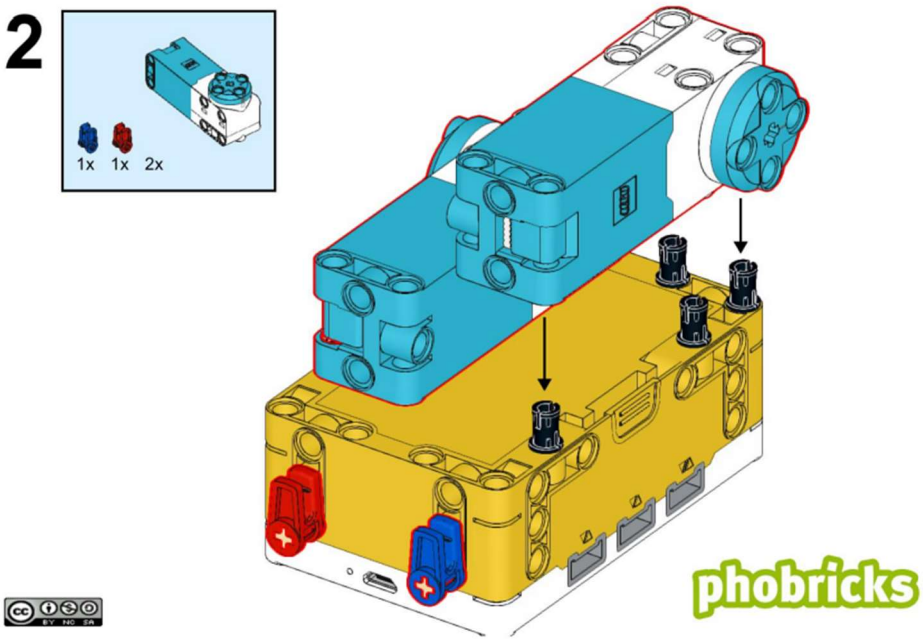
Extra task 5 (The Gyro Sensor): Can you also detect if your hub is tilted to the front? And can you play different sounds?

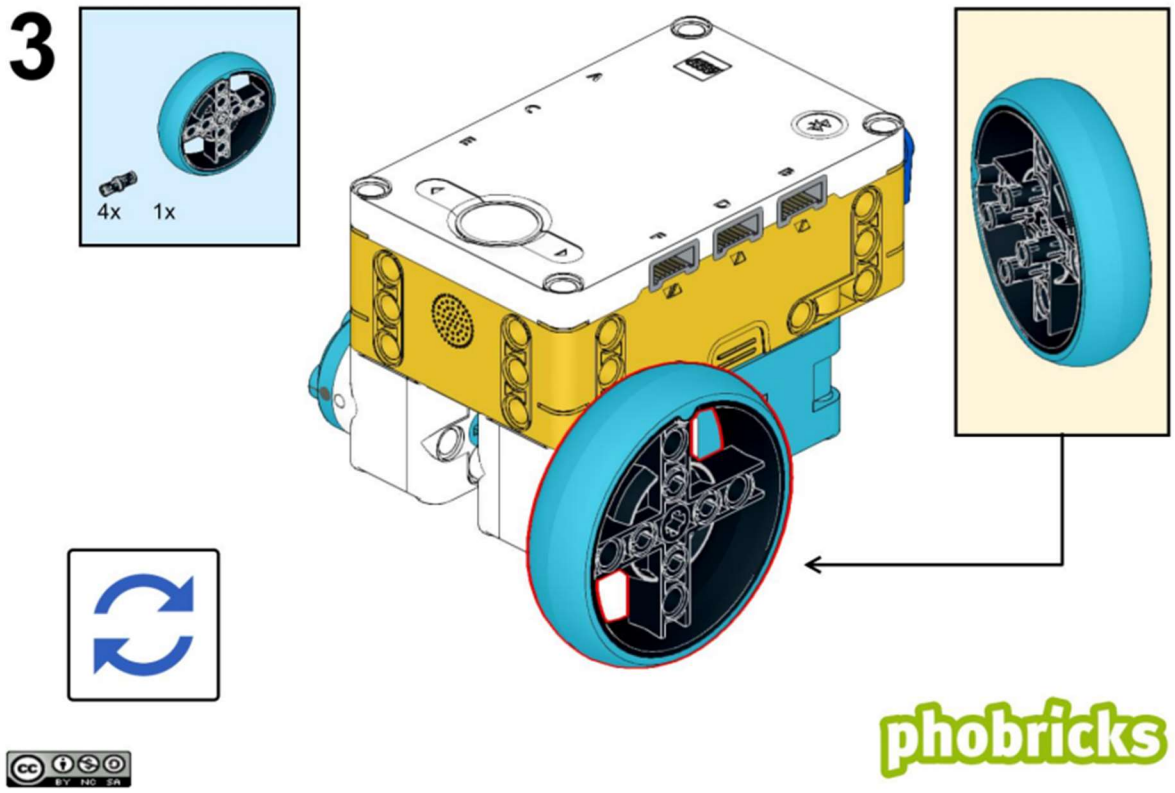
6: Build a small robot.

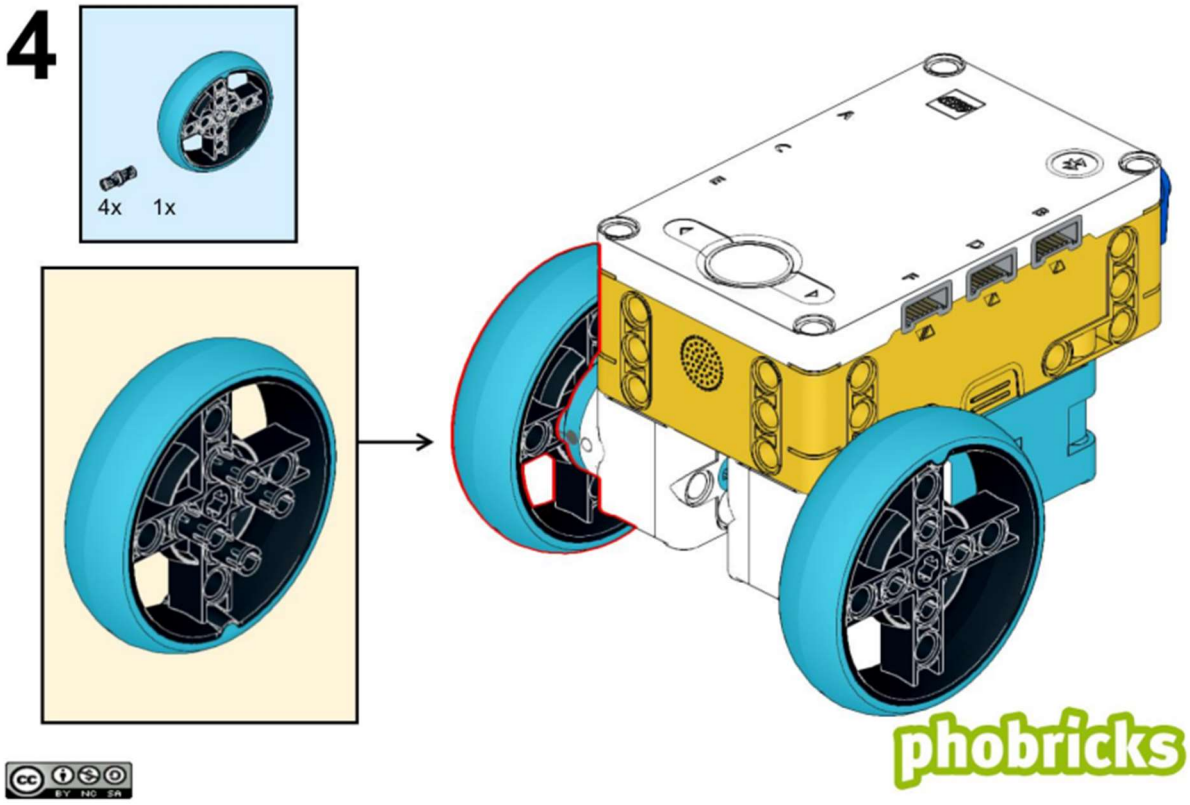
Learning objective

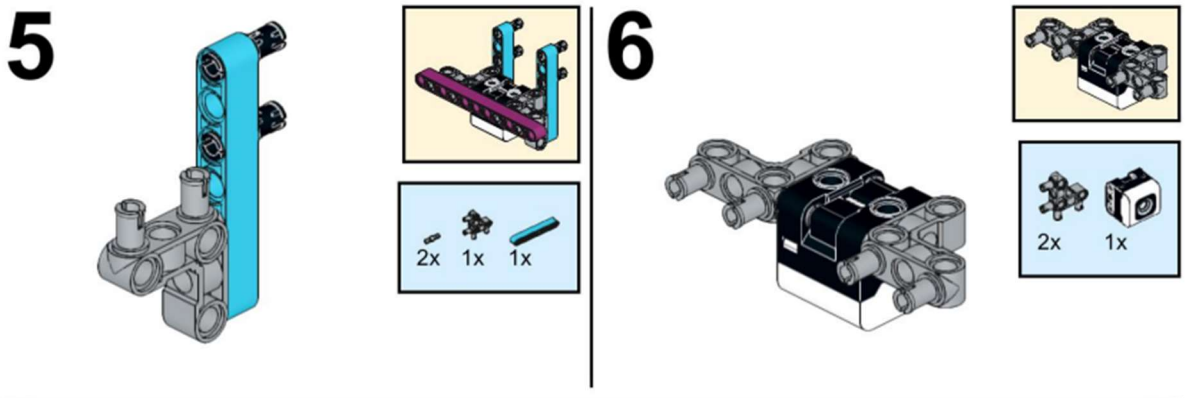
- ☐ Ich baue einen kleinen Roboter mit Farbsensor.

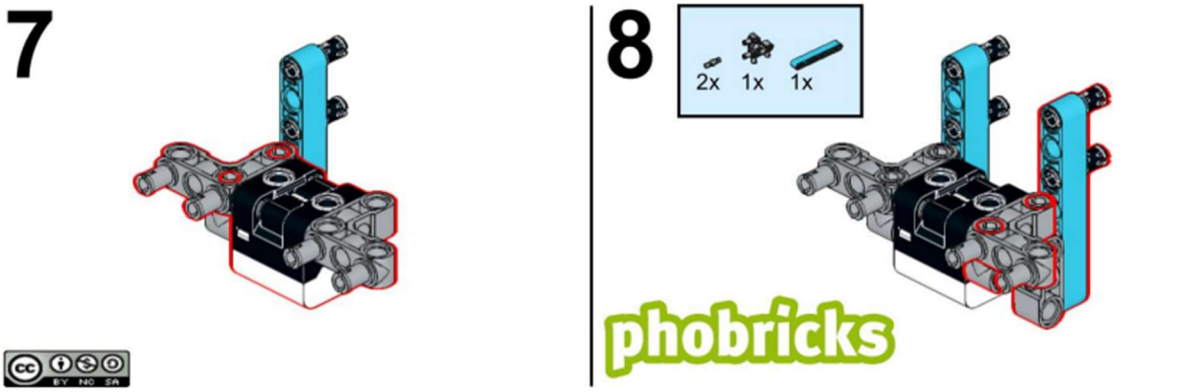


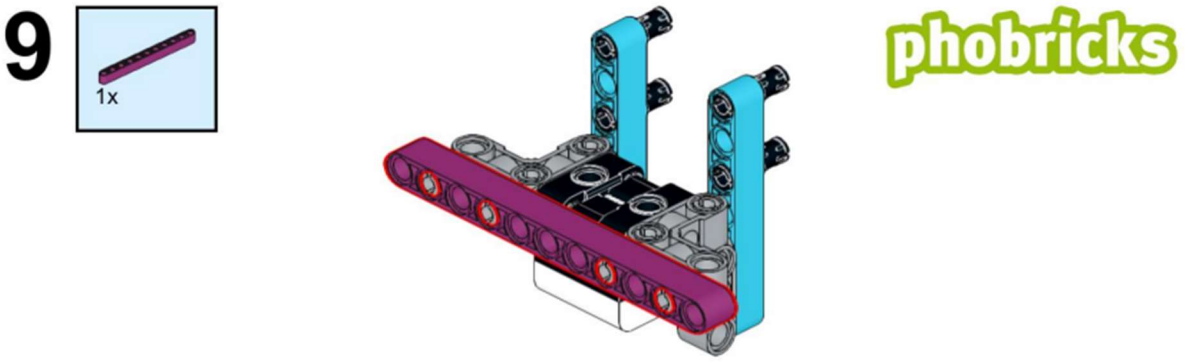


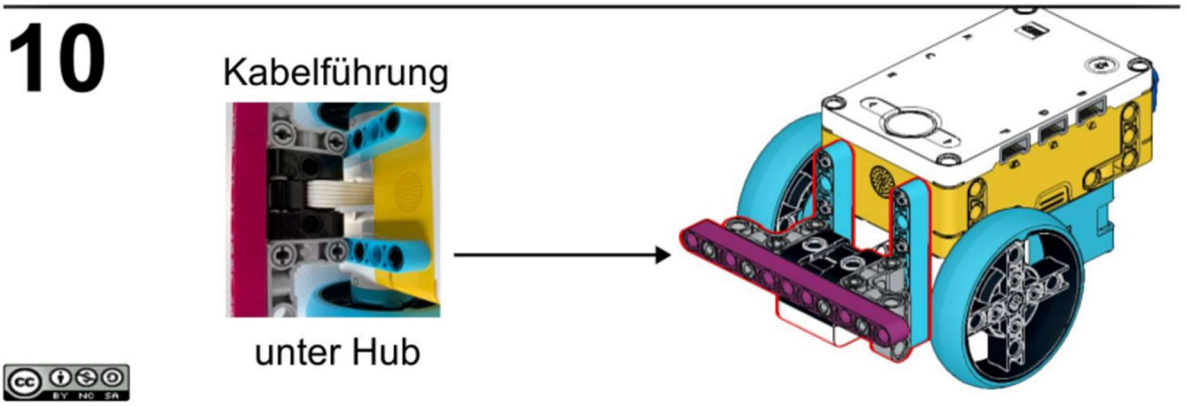


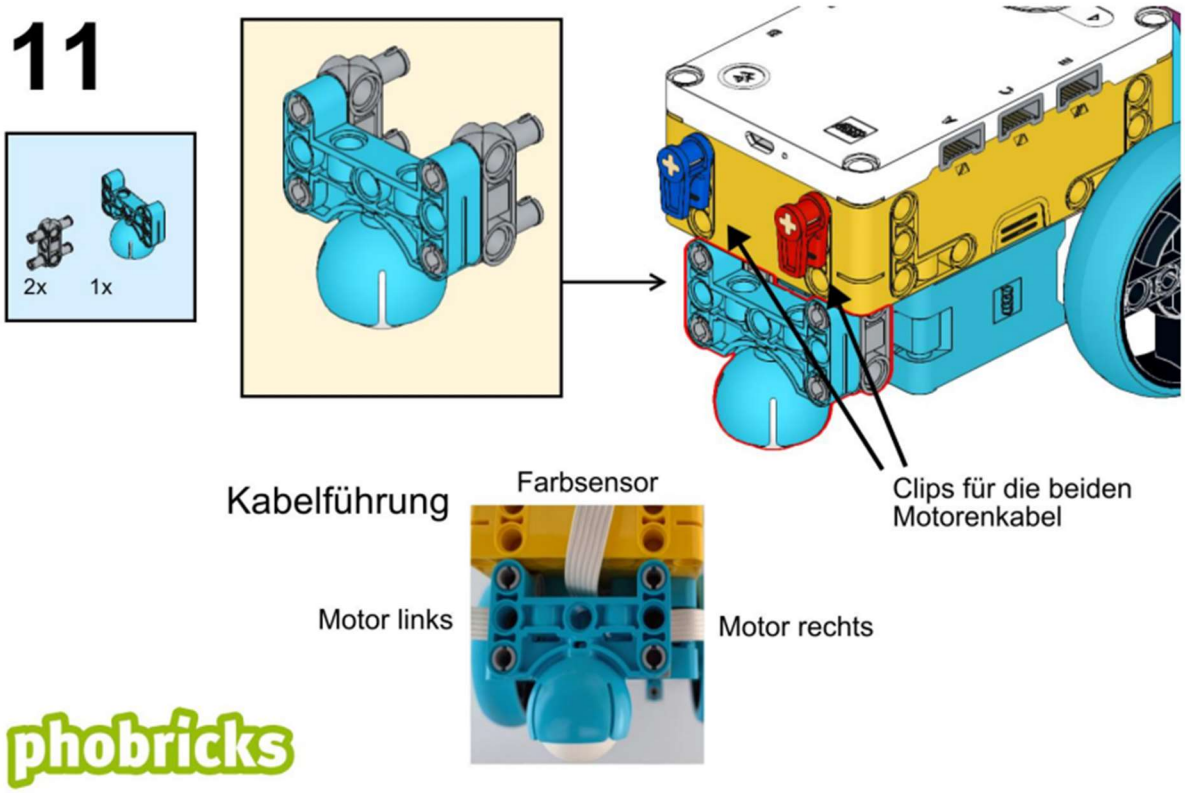




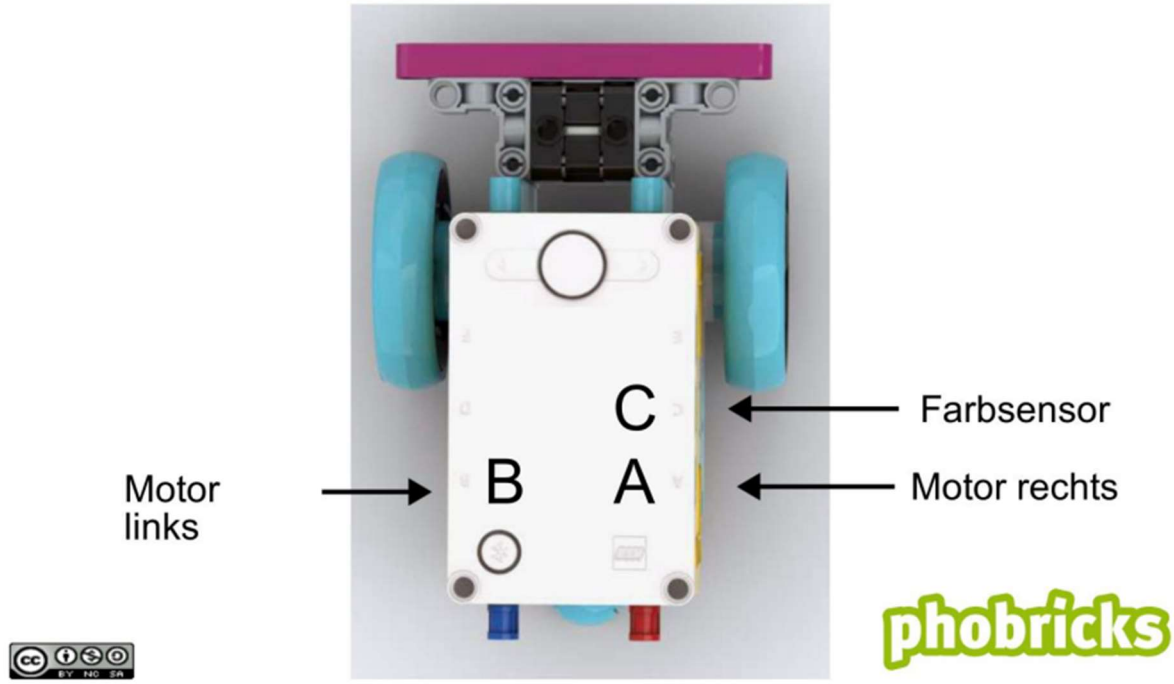








Anschlüsse



7: Making Our Robot Drive

In the tutorial you learned how to move a single motor. That is great for a tool. But if you want to drive like a car, you will need two motors.

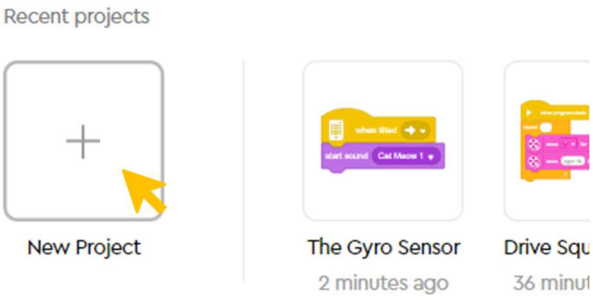
7.1 Start a new project

Learning objectives

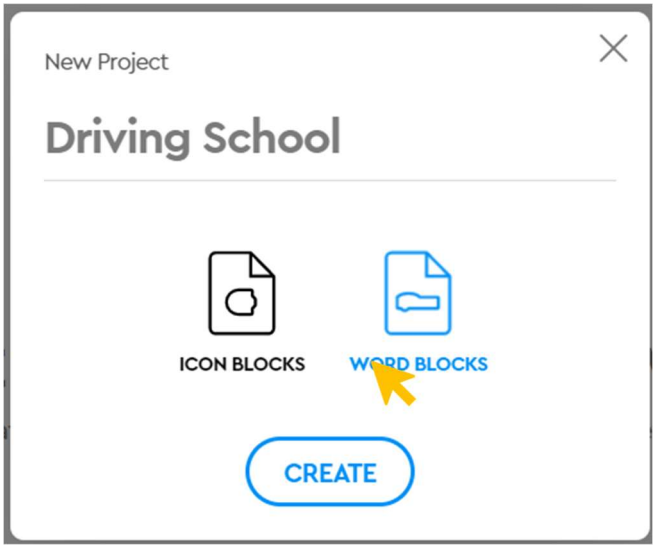
- ☐ I know how to start a new project in the SPIKE App.

Go to the **HOME page** in the SPIKE App. You find it with the little house icon from everywhere in your app.

Then click **New Project** (just under the blue Tutorial area).



Now you can see a small window with the words "Project 1" in very light gray. Write a good name for your project so you can find it more easily next time. For this example, use "Driving School". Then select **Word Blocks** and click "**Create**".



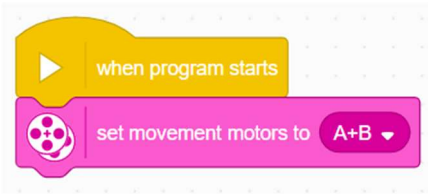
7.2 Assign movement

Learning objectives

- ☐ I know what movement motors are and how I can set them for the project.

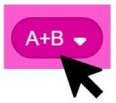
First, we need to teach our robot which motors it uses for driving, and where front and back are. These are called **the movement motors**.

Tipp:
When we set the movement motors right at the beginning of the project the code will always know which ones they are, and we needn't set them every single time we need them. This is especially important in more complex programs where we keep changing directions.



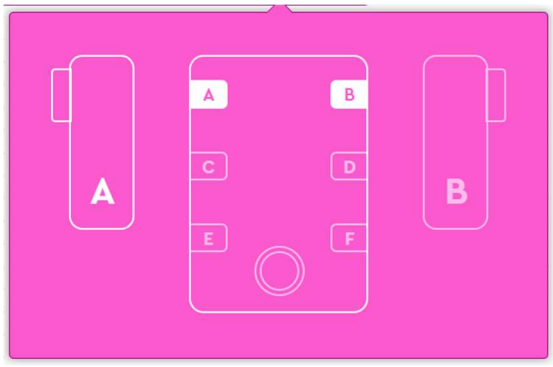
This is the standard setting, and it is perfect if your **left** motor is plugged in port **A** and the **right** one in port **B**. Obviously, if your motors are in different ports, you need to change the letters.

If after the start of the program your robot drives in the wrong direction (backward instead of ahead) you will need to switch the assignment. Of course, you could just unplug the motors and replug them so left is A and right is B. But depending on the size of your robot, that may not be an option. Instead, you can change the assignment in the code. Here's what you need to do:

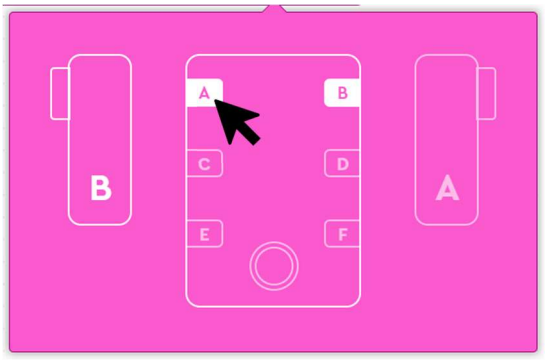
Click the settings field 

.

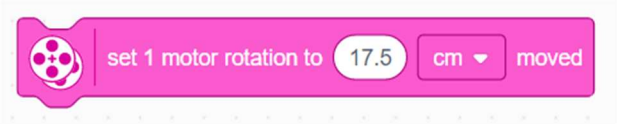
Now you see this:



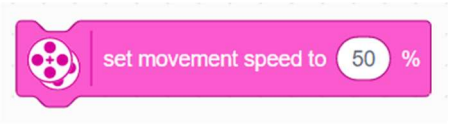
This graphic shows you in which two ports the motors are, and which one is assigned "left" and which one "right". Now if you click on the **A** on the hub image, the assignment is reversed, and **B is left**, and **A is right**.



Now we can also set the size of our wheels, because obviously a larger wheel drives a longer distance during one motor rotation than a smaller wheel. The wheels included in the SPIKE Prime box are the default size (17.5 cm), so if you forget to set the size it's not a big deal. But for your next project you might be using different wheels so it's best practice to get used to setting the size at the beginning.

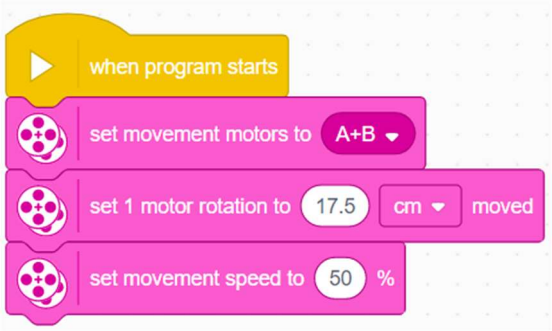


And lastly, we set the standard speed. This is used for all movements where no other speed is specified. Speeds are given in per cent: 0% is no movement, 100% is the maximum speed of the motors. If you write a number bigger than 100, the code will automatically interpret that as 100. You cannot trick the robot to drive faster, unless you use gears, which is more advanced mechanics than we are learning in this beginner's course.



Tipp:
For experimenting, it's usually best to start with a slower speed. That way it is easier to observe what is happening. Also, at smaller speeds the motor is more precise, because hat higher speeds, inertia kicks in. So best start slow, with 20% or 30%, and then keep increasing the speed, testing each time that your distances and turns are still yielding the correct results.

So at the end of the movement motors assignment, your code looks like this:



Of course, you have to make all the correct changes to reflect in which port your motors are, how big your wheels are, and how fast you want the standard speed to be.

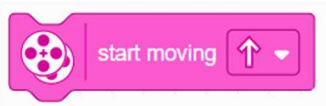
7.3 Simple driving

Learning objectives

- ☐ I can start my robot and stop it using a "Wait"-block.

Now we are finally ready to make our motor drive!

Add a "**Start Moving**"-Block under your movement motors assignment. The arrow pointing up tells the robot to go straight ahead.



"Start moving"-Block

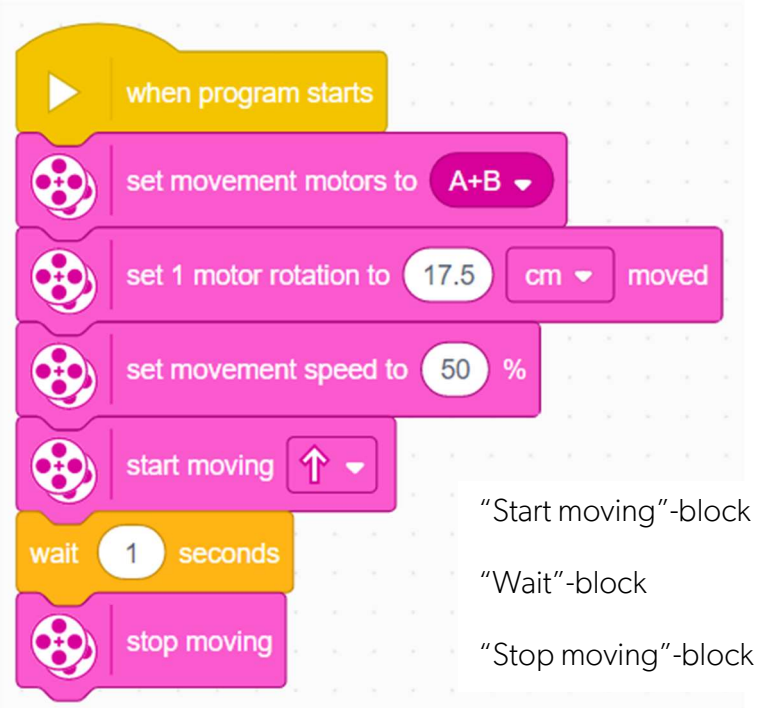
Now start your program. What is happening? How do you get your robot to stop again? After all, you don't want it to run until your battery runs empty.

You can stop the motor either from within the SPIKE App by pressing the red button, or by pressing the round button on your robot.

But we want the robot to stop on its own.

Now we want the robot to drive for exactly 1 second and then stop. So we need an orange "**Wait**"-Block and a pink "**Stop moving**"-block. Add them to your stack.

Your code now looks like this:



"Start moving"-block

"Wait"-block

"Stop moving"-block

7.4 Use a "Wait until"-block to stop at the black line

Learning objectives

- ☐ I know the "Wait until"-block and can use it to stop my robot.
- ☐ I can use the color sensor.

Now we don't want to drive for one second, but exactly as long as it takes to reach a black line.

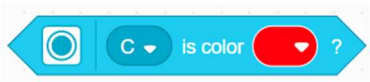
So we need a "**Wait until**"-block.

"Wait until"-block

There is a dark hexagon within this block. This is where we need to put the condition "arriving at a black line". So, to know whether we are at the black line, we need a **color sensor**. Make sure your color sensor is plugged into port **C**! Then go to the light blue coding blocks which deal with sensors.

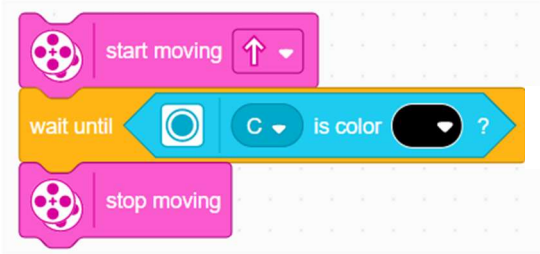
Tipp:
For all sensors, there are round shaped blocks, which tell you the sensor values at that moment, as well as pointy block, which sets a condition to look for. In a "Wait until"-block you always need the pointy sensor blocks!

We need the "**Is color?**"-block:



But wait, there's a problem here! With this setting, the sensor is not looking for black, but for red! Click on the little arrow in the red field and select back. Then add this block into the hexagon area of your "Wait until"-block.

Now this part of your code will look like this (after the movement motors assignment, which will not be shown any more to save space).



"Is color?"-block inside a "wait until"-block

Task:

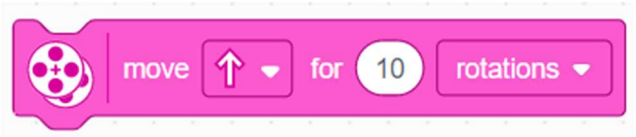
When the robot reaches the black line it must stop and also display an "X" in the light matrix and make a beeping sound. What do you need to add to your code?

7.5 The "Move … for"-block

Learning objectives

- ☐ I know the "move … for"-block.

The "Start moving"-block is useful when we don't know exactly how big the distance is until the condition in the "Wait until"-block is true. But in many cases, we have a ruler or a measuring tape and know the exact distance. In that case we can use the "**Move … for**"-block.



Tasks:

1. What do you need to do to make your robot drive backwards?
2. Make your robot drive straight ahead for exactly 29 cm. Use the long side of an A4 sheet to check the distance!
3. Make your robot drive straight ahead for exactly one rotation. How many centimeters is that?
4. Set the movement speed to 20%. Then make your robot drive along a ruler or measuring tape for exactly one second. Now change the speed to 40% and repeat. What can you observe? Now repeat again with 80%.

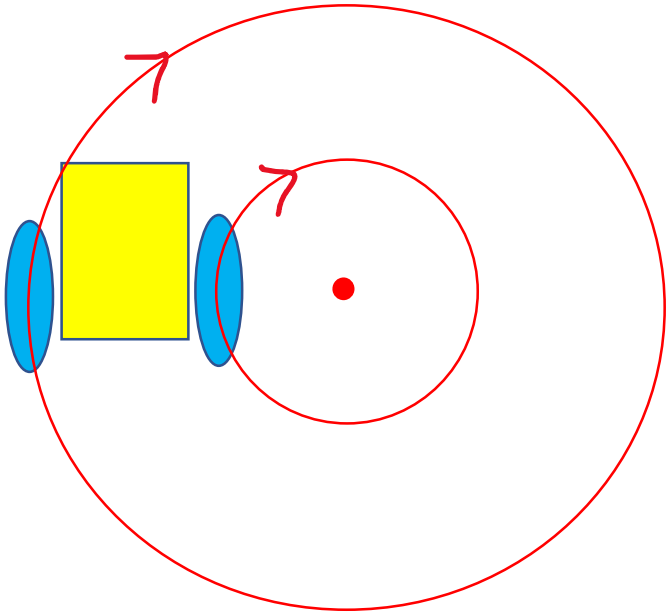
7.6 Spinning and turning

Learning objectives

- ☐ I know different methods of making my robot change directions.

So far, your robot can drive straight ahead, and you tried make it go backwards as well. But it can also spin and turn. There are several ways to do that:

1. **Arc turn**: You can drive an arc (which is a part of a circle): Technically, both motors move forwards, but with different speeds.



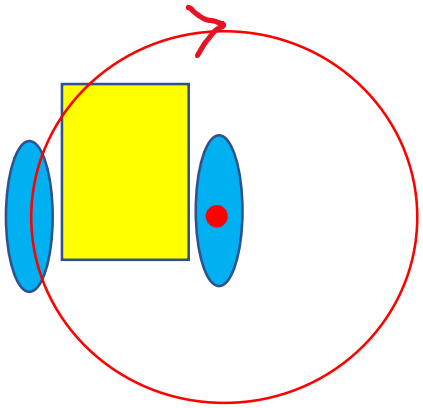
To do this, we need the "**Move right/left ... for**"-block.

Task:

Try what happens when you set the number in the right/left field bigger or smaller!

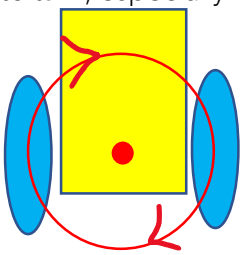
If you leave the number at 30, how many centimeters do you need to complete a full circle?

2. **Compass turn**: A special case of the arc turn is the "compass turn". It works like a compass that you use in geometry: One wheel stays in its place, the other one moves.



For this kind of turn we need the same "**Move right/left ... for**"-block as before. But now the value in the right/left field must be **50** (of you want to turn right) or **-50** (if you want to turn left).

3. **Pivot turn**: The robot spins around the spot in the middle between its two movement wheels. This is often the best way to turn, especially in a tight space.



Technically, the two motors move in the exact same speed, but one moves forwards and one backwards.
Again, we need the "**Move right/left ... for**"-block. Set the right/left value to 100.

Task:

How many centimeters do you need to complete a full circle in a pivot turn?

If you have some time to experiment, then build a new robot which is wider than the one you have been using. Now try again how many centimeters you need to complete a circle!

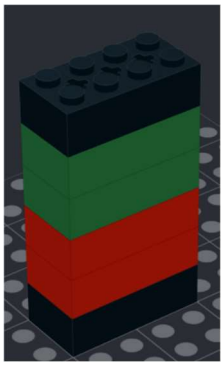
7.7 Conditions

Learning objectives

- ☐ I can modify the construction of my robot to fit a different need.
- ☐ I know how to code if-then and if-then-else conditions.

Now let's get back to our self-driving car-robot. Remember it had to drive to the yellow line and then check for the color of the traffic light? We want to do the same thing.

Unfortunately, we only have one color sensor in the standard SPIKE Prime box, so we need that to check for the traffic light. We cannot use it to look for the yellow line! Therefore, you will have to change the construction of your robot so the sensor looks ahead instead of down. Be creative, it's not hard! Next build one or two items you could use as a stop light. Make sure the red or green bricks are exactly at the same level as your color sensor! Here is an idea to get you started, but depending on how you attach the sensor your object needs to look different.



Now place the robot on your practice area without any object in front of the sensor.

Make sure your hub is turned on and connected.

Check what the color sensor sees:

You find this little info graphic in the top left corner of your project. You see to which port the motors are attached, (**A** and **B**) and in which position they are. We are interested in what the color sensor in port **C** tells us: Right now, it doesn't recognize any color. That's what this icon means: 

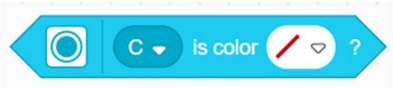
.

Now place your "traffic light" object in front of your sensor, so that you can but a finger between the sensor and the object. Now the color sensor in port **C** should show you the correct color.

We start the robot as usual with the "**Start movement**"-block followed by a "**wait until**"-block.

We then need this distinction between no color and either red or green.

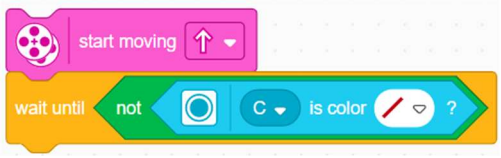
First, let's check for "no color recognized" with an "**is color?**"-block:



But we do want to have a color! We want the exact opposite of what we're having so far! So we need another block that turns this one into its opposite: The "**Not**"-block.

As you can see, this block has space for another pointy block, so we just drop the blue "**Is color?**" into the green "**Not**"-block.

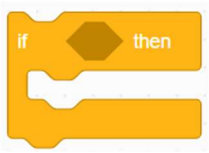
And now we drop this whole thing into the "**Wait until**"-block.



So now we're driving until the sensor detects the "Traffic light" object.

Now it needs to make the decision: Stop or keep driving?

For a decision we need another C-Block, the "**If-then**"-block.



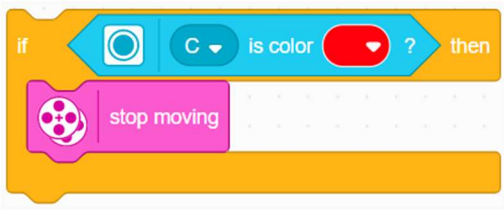
Now think: Which is more important: to detect red, or to detect green? If you don't see the green light, you will just keep driving, which is fine because that is what you should do anyway. But if you miss the red light and keep driving, a bad accident could happen. So we scan for the red light.

Remember:

Always check for the most important thing first!

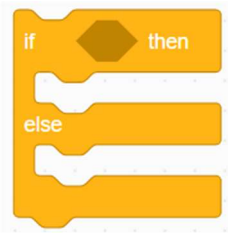
So we add the necessary color sensor value in an "**Is color?**"-block into the hexagon of the "**If-then**"-block.

And inside the C-Block we write what we want to do when that condition is true:



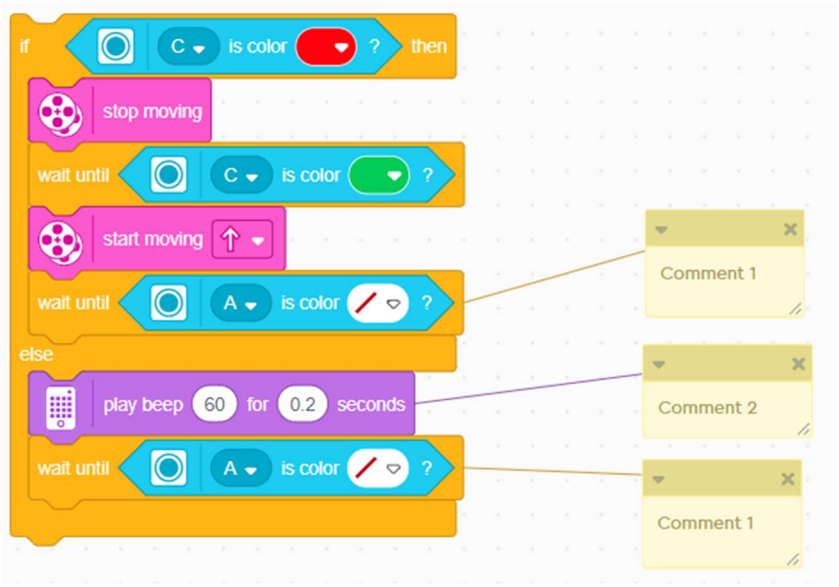
With this code, the robot will only stop if the detected color is red. If it isn't, it will just continue with whatever is below the "**If-then**"-block.

But we also want to give a special order for when the color sensor detects green. For example, we want the robot to drive a bit slower than normal, drive across the cossroads, and then pic up speed again. For this we need a different condition block, the so-called "**If-then-else**"-block or **E-block**.



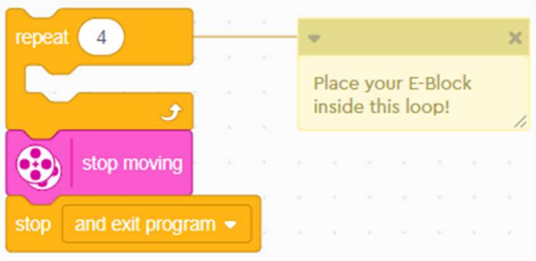
Here, in the top part we add the code for whatever happens when the light is red, and in the bottom part we add what it does if it is not red, i.e. if any other color is detected. We will see later what we can do if we have more options than just two, but right now all that matters is red or not red.

Our **E-Block** could look like this:

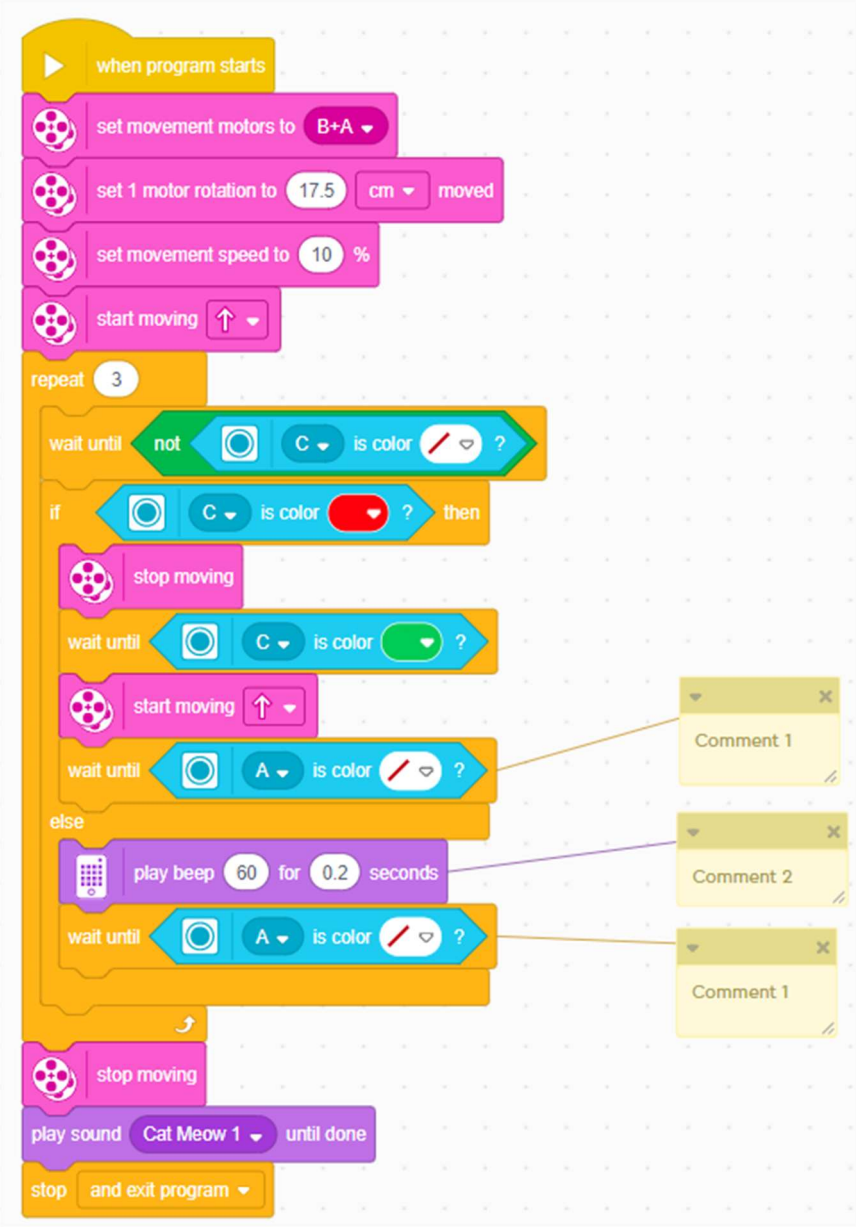


This wait block where you wait until the sensor detects no color gives you enough time to rearrange your traffic light objects. If you do not use a wait block, the loop will repeat within a fraction of a second so you cannot observe anything.

Now if we know that there are four intersections with traffic lights on the way, we could add a loop around the E-Block. And at the end of the loop it would be a good idea to stop the robot and turn off the program, or the robot will keep running forever!



This is what the complete code will look like:



Teil 4: Important algorithms

Remember:

An algorithm is a task that is repeated again and again within your code and that works autonomously, i.e. without you needing to do anything after you've finished programming.

8: "Stay in the arena"

Learning objectives

- ☐ I know what an algorithm, is.
- ☐ I can code the arena game.
- ☐ For advanced coders: I can extend the arena game by other events.

In this task you combine everything you have learned so far: Lights, sounds, movements, loops and conditions. Loops and conditions can be nested!

Task:
Create a new project called "Arena Game".
Your robot must remain within the black border lines. Each time it reaches the line, it must:

- Stop
- Beep
- Show a happy face for 0.3 seconds
- Drive backwards for 8 cm, then make a turn to the left to complete about one third of a circle.
- Drive forwards again until it hits the next border.

Repeat ten times. After ten times, the robot must stop and make a funny sound.

Tipp:
First, try solving the task on your own! Always keep trying if the robot does what you want it to do; and change your code where necessary. When you have a problem, there are some options:

- First, try coding what happens when you get to a black line, without bothering with the condition and the loop. Keep testing. Then add the condition, then the loop.
- If you find find it difficult to figure out the structure of the code, then look at page 26 in this book. There you will see the code, but without any texts. The colors and shapes of the code blocks can help you.
- If that still isn't enough, check with your teacher or instructor. They know the solution. 😉

9: Follow the line!

Learning objectives

- ☐ I know why a line follower is useful.
- ☐ I can program a simple zigzag line follower.

Sometimes you are not exactly sure which way your robot needs to take. Or the way is so complicated that it's difficult to just the normal movement blocks. But in such situations there is often a very useful help: black lines you can follow!

Task:

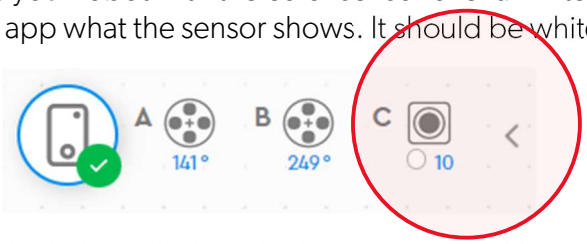
Make a long line on the floor using masking tape or chalk. It doesn't have to be straight.
Form a team with another person.
One of you is the "robot", the other the "programmer".
The "robot" starts blindfolded at one end of the line.
The "programmer" guides the "robot" by giving commands.
The task is to follow the line as closely as possible. If the "robot" loses the line, the "programmer" has to give commands to get back on track.
After that, switch roles.

Observe how the "robot" follows the line!

Now we want to program a line follower that works in a very similar way.

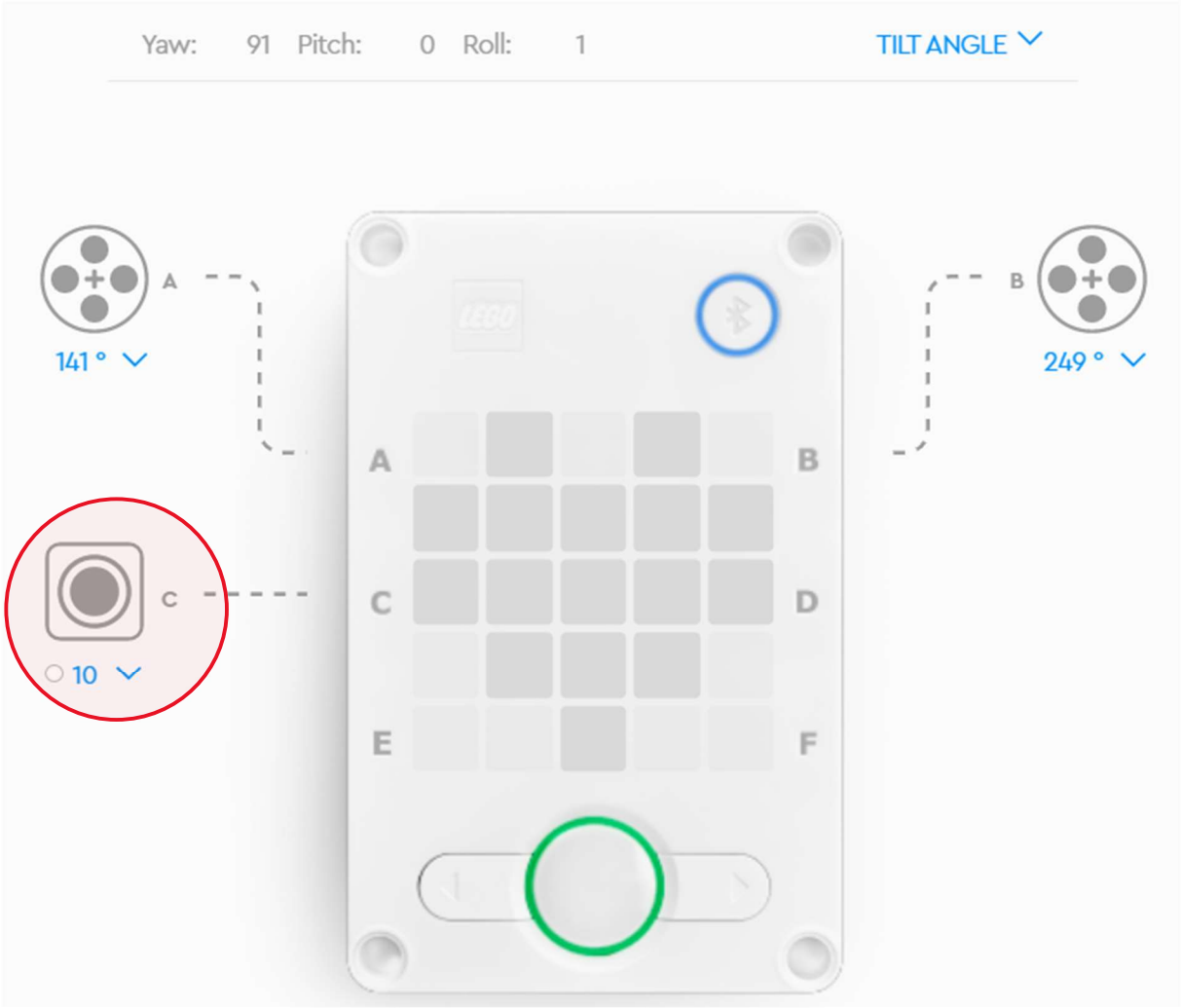
First, we need to try something. You need to reattach the color sensor in such a way that it faces the ground. It has to be at the front end of the robot.

Now place your robot with the color sensor over a white surface, e.g. a piece of paper. Now look in the app what the sensor shows. It should be white.

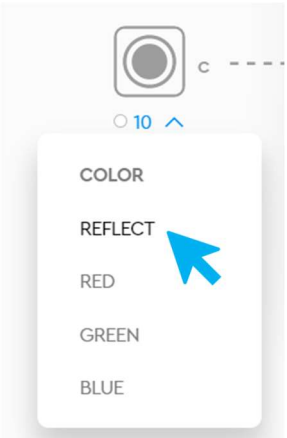


The sensor shows «White». The number 10 is the color ID for white.

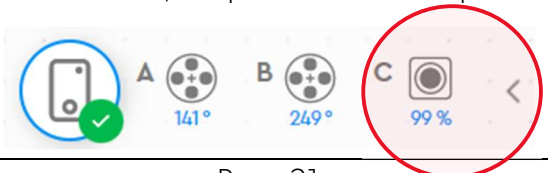
Now click the hub symbol inside the green circle. You will see an enlarged view of your hub with the connected motors and sensors:



If you click the arrow below the color sensor icon you get a context menu. There, select "Reflect:"



When you close the detailed Hub view, the port view in the top left corner will look like this:



The color sensor is now showing a very high percentage value, close to 100%. If the number is significantly lower, that may be due to very bad lighting or your surface isn't totally white. Or your sensor is attached at a too high level. Don't worry about that, it's not a problem.

Now place your sensor over a black surface. The value will be very small, usually below 20%.

Now place your robot in such a way that the color sensor is exactly over the right-hand edge of the black line (in driving direction). Check the sensor value again. It should be somewhere between 40% and 60%. This value will be our **Threshold Value**.

✗ sensor completely on the black line. Sensor value too low

✗ sensor too much within the white area. Sensor value too high.

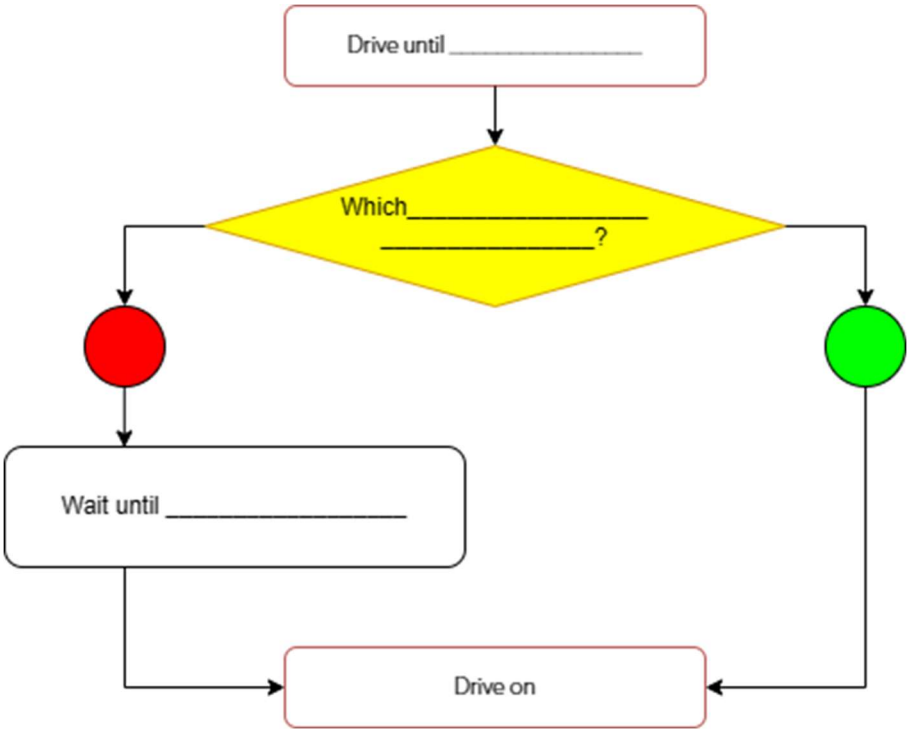
Just right!

Now remember your experiment with the line on the floor: When the "robot" lost the line and moved too much to the right, it had to correct towards the left. When the "robot moved too much to the left, it had to correct towards the right.

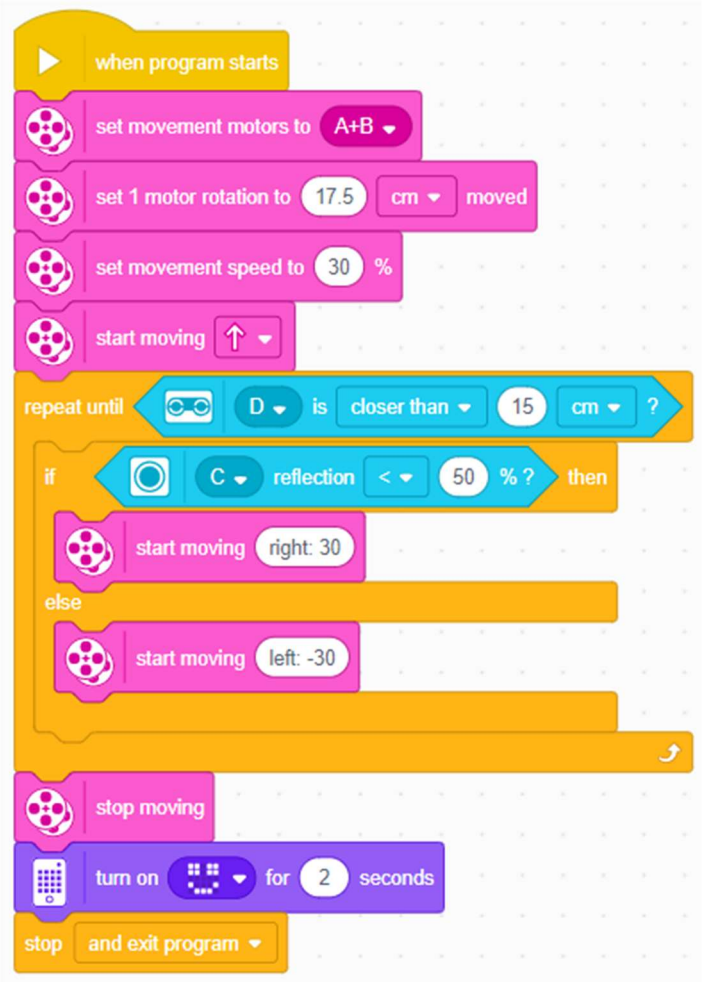
Now we will do the same thing with our real robot. And our color sensor helps with it: When the robot moves too much to the right, the sensor value will get higher. When the color sensor moves too much to the left, the sensor value will be too low. But we want to move as straight as possible directly over the edge of the line, with the sensor value always near our threshold value.

And of course, we want to repeat this until we've reached the end of our line.

For this exercise we want to stop when the robot is less than 15 cm (about the length of a match) from an obstacle. So you need to add a **distance sensor** to the front of your robot.



Our code will look like this:



«**If-then-else**»-block:

Condition: Light reflection is smaller than the threshold value.

Then (If Condition is True): It is too dark → Correct toward the right.

Else (If Condition is False): Too light → Correct toward the left.

Observe: What does your robot drive like?

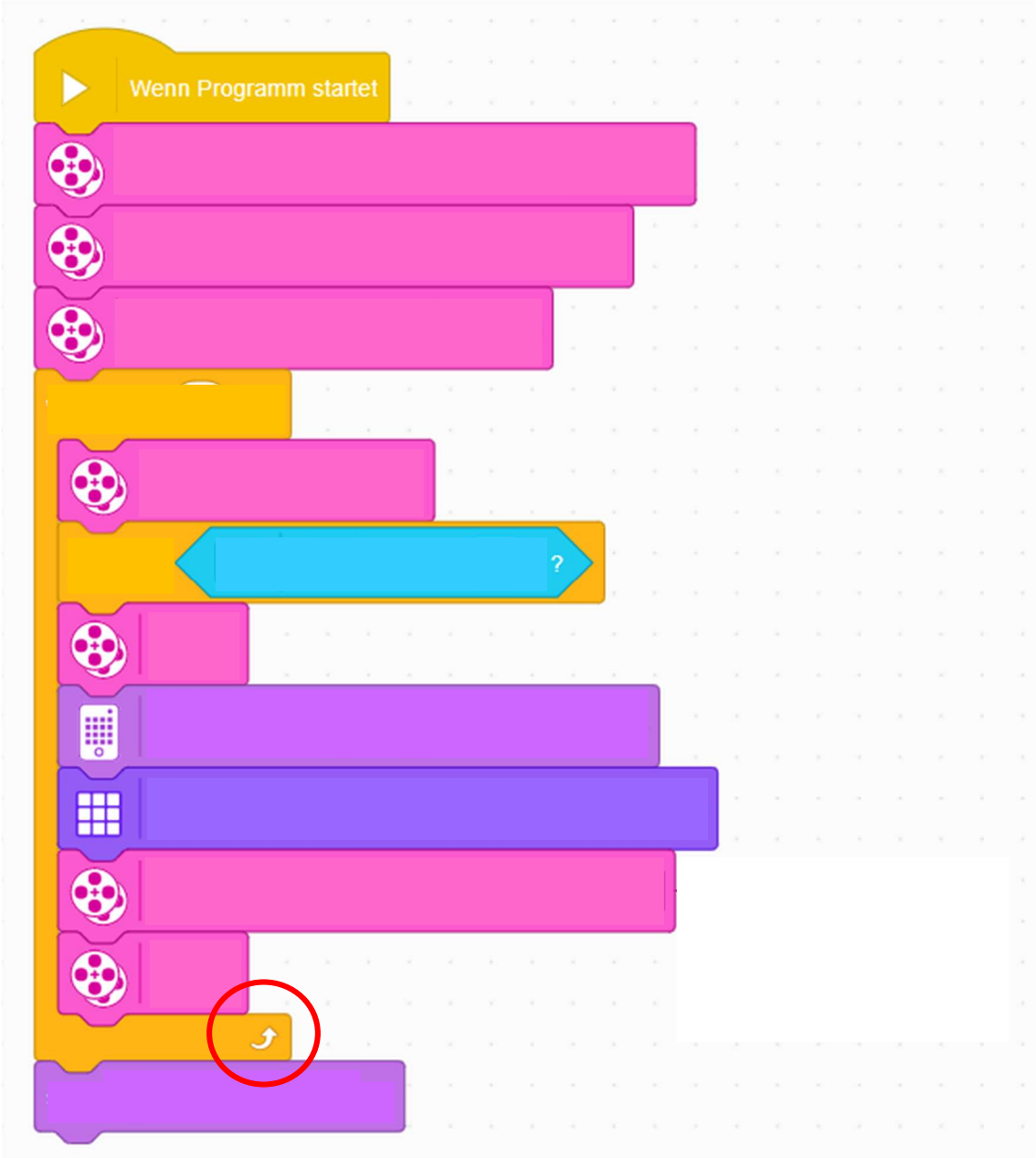
Task:

Experiment with the correction value in the "If-then-else"-block. What happens when you set that number bigger or smaller? Try to find the value that gives the best results. Also, experiment with the speed that works best for you.

Hilfe zum Lösen der Aufgabe «Bleib in der Arena»

Denke daran: Am Anfang des Codes musst du die Antriebsmotoren festlegen. Im Warten-bis-Block legst du fest, was er tun soll, wenn er an eine schwarze Linie kommt, die er mit seinem Farbsensor erkennt.

Beachte: Wenn du den grossen Bogen rückwärts fahren willst, musst du vor der Masseinheit (cm oder Umdrehungen) eine negative Zahl (Zahl mit einem Minus davor) eingeben, dann fährt der Roboter in die umgekehrte Richtung.



Lösung der «Aufgabe bleib in der Arena»