

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

Moti Ben-Ari and other contributors
see authors.txt for details

Version 1.3~pre1 for **Aseba 1.3.1**

© 2013–14 by **Moti Ben-Ari** and other contributors.

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.



Contents

1	Your First Robotics Project	4
2	Changing Colors	11
3	Let's Get Moving	13
4	A Pet Robot	17
5	The Robot Finds Its Way by Itself	23
6	Bells and Whistles	27
7	A Time to Like	30
8	States: Don't Always Do the Same Thing (Advanced)	32
9	Counting (Advanced)	38
10	What Next?	42

Preface

What is a robot?

You are riding your bicycle and suddenly you see that the street starts to go uphill. You pedal faster to supply more power to the wheels so that the bicycle won't slow down. When you reach the top of the hill and start to go downhill, you squeeze the brake lever. This causes a rubber pad to be pressed against the wheel and the bicycle slows down. When you ride a bicycle, your eyes are *sensors* that sense what is going on in the world. When these sensors—your eyes—detect an *event* such as a curve in the street, you perform an *action*, such as moving the handlebar left or right.

In a car, there are sensors that *measure* what is going on in the world. The speedometer measures how fast the car is going; if you see it measuring a speed higher than the limit, you might tell the driver that he is going too fast. In response, he can perform an action, such as stepping on the brake pedal to slow the car down. The fuel meter measures how much fuel remains in the car; if you see that it is too low, you can tell the driver to find a gas station. In response, she can perform an action: raise the turn-signal lever to indicate a right turn and turn the steering wheel in order to drive into the station.

The rider of the bicycle and the driver of the car receive data from the sensors, decide what actions to take and cause the actions to be performed. A *robot* is a system where this process—receive data, decide upon an action, perform the action—is carried out by a computerized system, usually without the participation of a human.

The Thymio-II robot and the Aseba VPL environment

The Thymio II is a small robot intended for educational purposes (Figure 1.1). The robot includes sensors that can measure light, sound and distance, and can detect when buttons are touched and when the robot's body is tapped. The most important action that it can perform is to move using two wheels, each powered by its own motor. Other actions include generating sound and turning lights on and off.

In the rest of this document, the Thymio II robot will be called simply Thymio. It will always refer to the version II of the robot.

Aseba is a programming environment for small mobile robots such as the Thymio. VPL is a component of Aseba for *visual programming* that was designed to program Thymio in an easy way through event and action blocks. This tutorial assumes that you have Aseba installed on your computer; if it is not the case, go to <https://aseba.wikidot.com/en:downloadinstall>, select your operating system, download and install.

Chapter 1

Your First Robotics Project

Getting to know your Thymio

Figure 1.1 shows the front and top of Thymio II. On the top you can see the center circular button and four directional buttons. Behind the buttons, the green light shows how much charge remains in the battery. At the back are the top lights which have been set to red. There are similar lights on the bottom which have been set to green (Figure 3.2). The small black rectangles are sensors which you will learn about in Chapter 4. You can ignore the small red lights for now.



Figure 1.1: The top and front of the Thymio robot

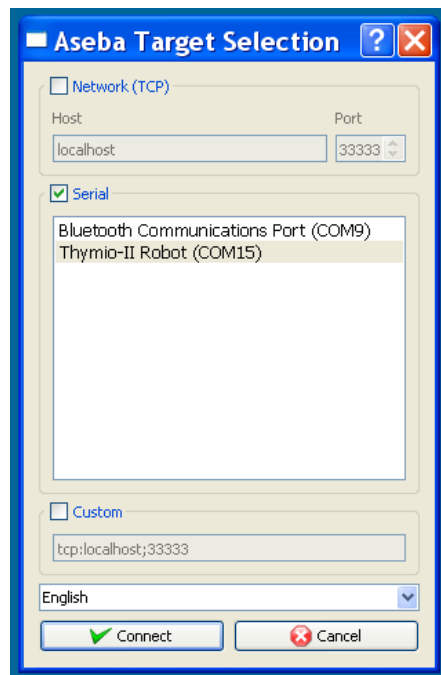



Figure 1.2: Connect to Thymio, through serial port (USB)

Connect the robot and run VPL

Connect your Thymio robot to your computer with a USB cable; the robot will play a sequence of tones. If the robot is turned off, turn it on by touching the center button for five seconds. Run VPL by double-clicking on the icon .



Important information

When a small icon appears in the text, a larger image is displayed in the margin.

VPL may connect automatically to your robot. If not, the window shown in Figure 1.2 will be displayed. Check the box next to **Serial**, click on **Thymio Robot** below it, select a language, and then click **Connect**. Depending on the configuration of your computer and the operating system that you are using, there may be several entries in this table and the data following **Thymio** may be different from what is shown in the Figure.



Trick

It is also possible to access VPL from Aseba Studio, the text-based programming environment, through the VPL plugin found in the *Tool* area at the bottom left of the screen.

The VPL user interface

The user interface of VPL is shown below. There are six areas in the interface:

1. A toolbar with icons for opening, saving, running a program, etc.
2. A program area where programs for controlling the robot are constructed.
3. An indication whether the program you are creating is well-formed or not.
4. A column with available event blocks to construct your program.
5. A column with available action blocks to construct your program.
6. The text translation of the program (see at the bottom of this page).

The event and action blocks will be described in the course of this document.

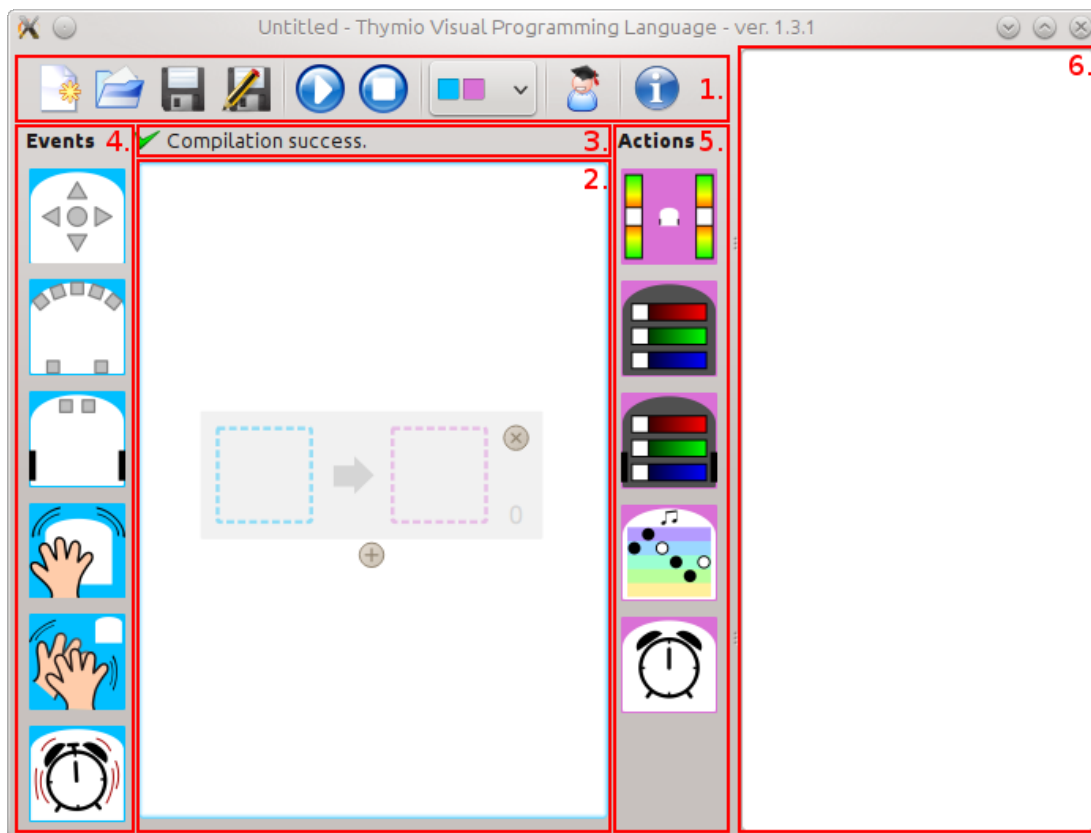



Figure 1.3: The VPL window



To go further

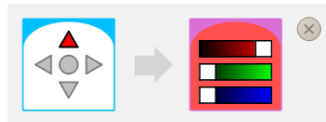
When you construct a program using VPL, the text program that will be loaded into the robot appears in the right part of the window. If you are curious and wish to understand this language, you can read the [text mode tutorial](http://aseba.wikidot.com/en:thymiotutoriel) (<http://aseba.wikidot.com/en:thymiotutoriel>).

Write a program

When you start VPL, a blank program area is displayed; if, after having built a piece of program, you wish to clear the content of the program area, click .



A program in VPL consists of one or more *event-action pairs*, each constructed from an event block and an action block. For example, the pair:



causes the top light of the robot to display red when one touches the front button on the robot.

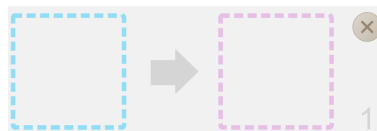


Important information

The meaning of an event-action pair is:

When the event occurs, the robot runs the action.

Let us construct an event-action pair. In the program area you see an outline of a pair:



The left, light blue, square is the space for the event; the right, pink, square is the space for the action. To bring a block to the program area from the sides (areas 4 and 5 of Figure 1.3), you can click on it or drag it into the corresponding square by holding the left mouse button pressed and releasing it when the block is in its place.

Start by bringing the button event block  into the blue square. Now, bring the top color action block  into the pink square. You have constructed an event-action pair!

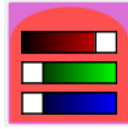


Next we have to modify the event and the action to do what we want. For the event, click on the front button (the top triangle); it will turn red:




This specifies that **an event will occur when the front button of Thymio is touched**.

The color action block contains three bars with the primary colors red, green, blue; at the left end of each bar is a white square. The colored bar with its white square is called a *slider*. Drag a white square to the right and then back to the left, and you will see that the background color of the block changes. All colors can be made by mixing these three primary colors: red, green and blue. Now move the red slider until the square is at the far right, and move the green and blue sliders until they are at the far left. The color will be all red with no blue nor green:




Save the program

Before running the program, save it on your computer. Click on the icon  in the toolbar. You will be asked to give the program a name; choose a name that will help you remember what the program does, perhaps, **display-red**. Choose the location where you want to save your program, on the desktop for instance, and click on Save.



Run the program

To run the program, click on  in the toolbar. Touch the front button on the robot; the light on top of the robot should change to red.



★ **Congratulations!**

You have created and executed your first program. Its behavior is:

When you touch the forward button of the Thymio, it becomes red.

Turn the robot off

When you have finished working with the Thymio robot, you can turn it off by touching and holding the center button for four seconds until you hear a sequence of descending tones. The battery in the robot will continue charging as long as it is connected to a working computer. A red light next to the USB cable connector means that the robot is charging; it turns blue when the charging is completed (Figure 1.4). You can disconnect the cable when you do not use the robot.

Trick

You can recharge the robot faster by using a mobile phone charger with a micro-USB connector.

Should the USB cable disconnect during programming, VPL will wait for the connection to be made again. Check both ends of the cable, reconnect and see if VPL is working. If you have a problem, you can always close VPL, reconnect the robot and open VPL again.

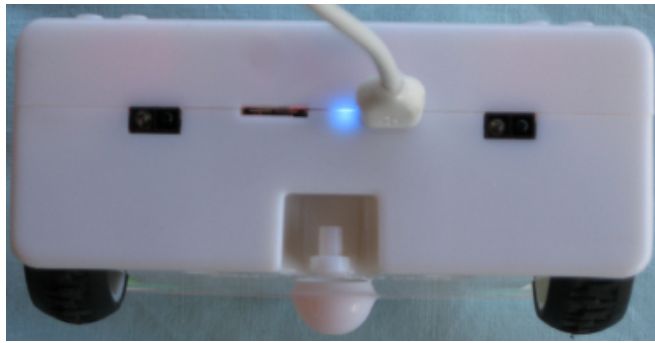





Figure 1.4: The back of the Thymio showing the USB cable and the charging light

Modify a program

- To delete an event-action pair, click  at the top-right of the pair.
- To add an event-action pair, click  available below every pair.
- To move an event-action pair to another position in the program, drag it and drop it at the desired location.






Open an existing program

Suppose that you have saved your program and turned off the robot and the computer, but later you wish to continue to work on the program. Connect the robot and run VPL as described previously. Click on the icon  and select the program you want to open, for example, **display-red**. The event-action pairs of the program will be displayed in the program area, and you can continue modifying it.





Additional operations in the VPL user interface

In the toolbar, you will find further features:

- **Save as** : Click on this icon to save the current program with a *different name*. Use it when you have a working program and you want to try something new without changing the existing program.
- **Stop** : This stops the program that is running and sets the speed of the motors to zero. Use it when the program asks the robot to move but does not include an event-action pair that can stop the motor.
- **Change color scheme** : You can select a different pair of colors for the background of the event and action blocks.



- **Advanced mode** : The advanced mode enables the use of state variables as described in Chapter 8.
- **Help** : Displays the VPL documentation in your browser (an Internet connection is required). This documentation is located at <https://aseba.wikidot.com/en:thymiovpl>.





Chapter 2

Changing Colors

Display colors


Create a program that causes two different colors to be displayed on the top of the Thymio robot when the front and back buttons are touched, and two other colors to be displayed on the bottom of the robot when the left and right buttons are touched.

Program file **colors.aesl**

We need four event-action pairs. There are four events—touching the four buttons—and a color action is associated with each event. Note the difference between the action blocks  and . The first block changes the color displayed on the top of the robot, while the second changes the color on the bottom of the robot. The block for the bottom has two black marks that represent the wheels.

This program is shown in Figure 2.1(a).

What colors are displayed? In the first three actions, the slider for one color is moved to the right edge and displayed, while the sliders for the other two colors are moved to the left edge and are not mixed in. Therefore, these actions display pure red, blue and green, respectively. The action associated with the left button mixes red and green giving yellow. You can see that the background of the color action changes depending on the sliders' positions; the background shows you which color your Thymio will display.

Run the program (icon ) and check that touching buttons changes the robot's color. Figure 1.1 shows the Thymio displaying red on the top and Figure 3.2 shows it displaying green on the bottom.

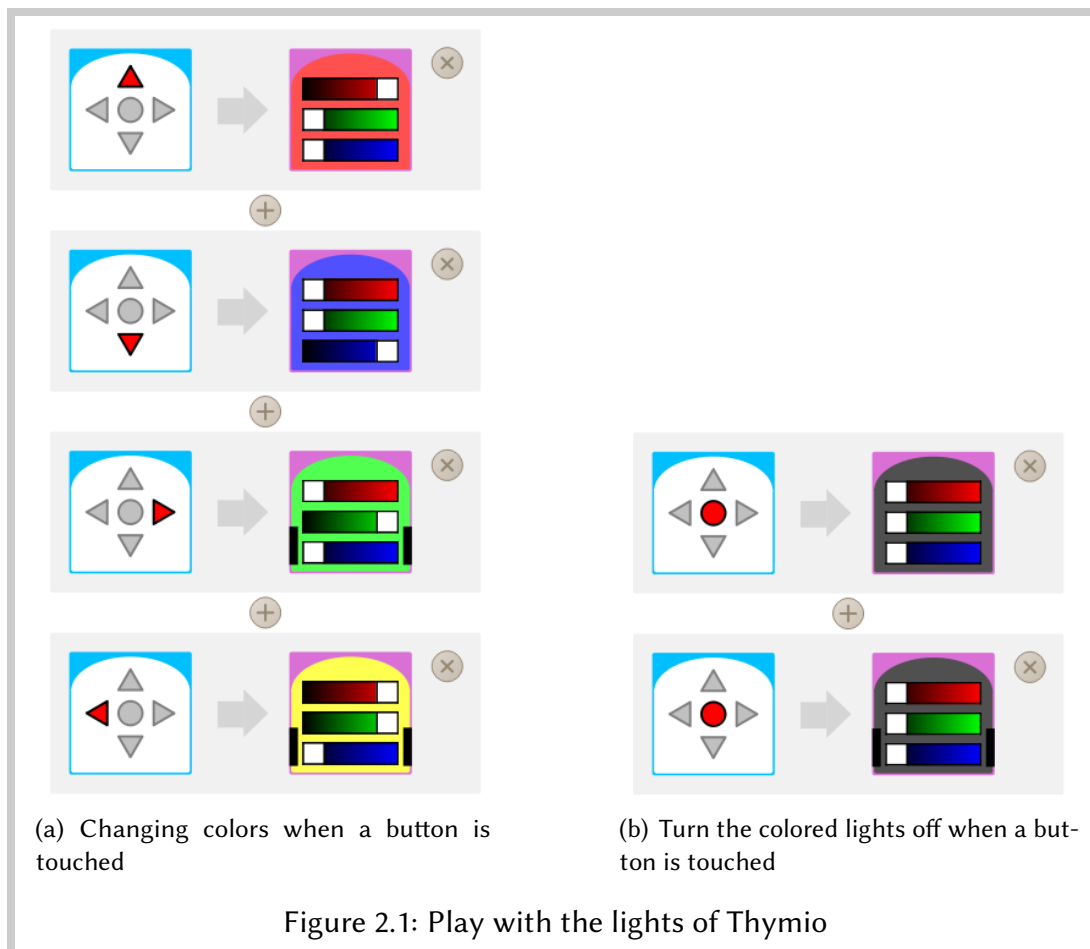


Exercise 2.1


Experiment with the sliders to see which colors can be displayed.

Turn the lights off

Let us modify the program so that the lights are turned off when the center button is touched. We need two event-action pairs, one to turn off the top light and the other to turn off the bottom light. By moving all three sliders in the color action block to the left, as in Figure 2.1(b), no colors are displayed and the light is turned off.



The *event is the same* in both pairs—touching the center button—but the *actions are different*—turning off the top or bottom light.

Do not forget to click on the icon  to run the program. In the future, we will not remind you to click this icon to run a program.

Multiple event-action pairs


- When a program is run, all the event-action pairs in the program are run.
- It is possible for several pairs to have the same event as long as they have different associated action blocks.
- If the event and the action block are similar in several pairs, VPL will display an error message (area 3 in Figure 1.3). You will not be able to run the program as long as there are errors.

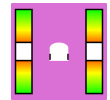
Chapter 3

Let's Get Moving

Move forwards and backwards

The Thymio robot has two motors, one connected to each wheel. The motors can be run forwards and backwards, causing the robot to move forwards and backwards, and to make turns. Let us start with a simple project to learn about the motors.

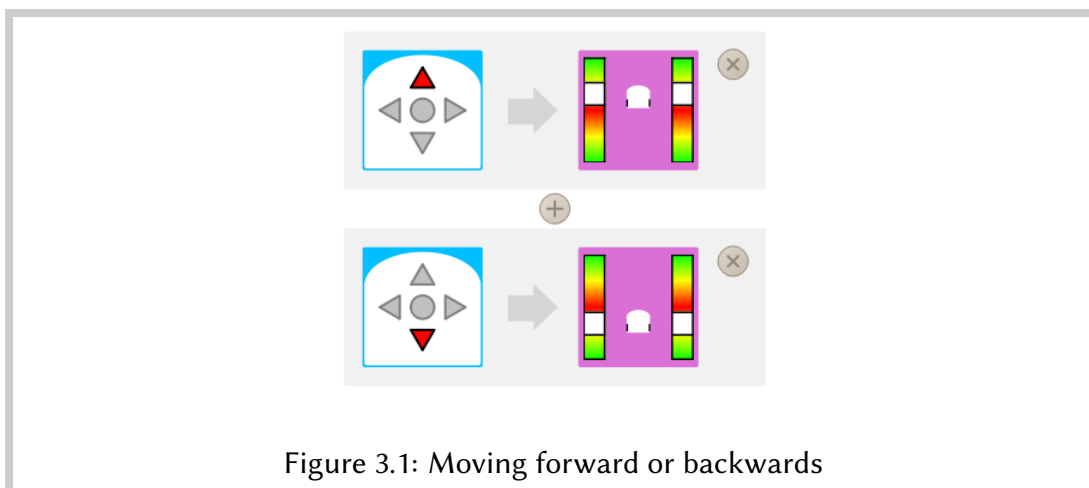
The motor action block  displays a small image of the robot in the center together with two sliders. The sliders control the speed of the motors, one slider for the left motor and one for the right motor. When the white square is centered in the slider, the corresponding motor is off. You can drag a square up to increase the forward speed and down to increase the backwards speed. Let us write a program to move the robot forward when the front button is touched and backwards when the back button is touched.



Program file **moving.aesl**

We need two event-action pairs (Figure 3.1). Drag and drop the event and action blocks and set the sliders equally for the left and right motors, half-way up for forward and half-way down for backwards.

Run the program and touch the buttons to make the robot go forwards and backwards.



Stop the robot

Help! I can't stop the robot's motors!

Click on the icon  to stop the robot.

Let us fix this problem by adding an event-action pair:



That will stop the motors when the center button is touched. When you drag the motor action block into the program area, it is already set with the sliders in the middle to turn off the motors.

Don't fall off the table

If your robot moves on the floor, at worst, it might hit a wall or pull its cable out, but if you place your robot on a table, it might fall off, crash and break! Let us arrange for the robot to stop when it reaches the end of a table.



Warning!


Whenever the robot moves on a table, be ready to catch it in case it does fall off.

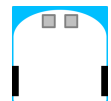
Turn the Thymio on its back. You will see at the front two small black rectangles with optical elements inside (Figure 3.2). These are the *ground sensors*. They send a pulse of infrared light and measure the amount of light that is reflected. On a light-colored table, there is a lot of reflected light, but when the front of the robot goes past the end of the table, there will be much less reflected light. When this is detected we want the robot to stop.



Trick

Use a table colored with a light color, but don't use a glass table, as it will likely not reflect the light and Thymio will believe that it is not on a table!

Drag the ground sensor event  into the program. There are two small squares at the top of the icon. Clicking the squares changes them from gray to white to red and finally back to gray. For this block, the meanings of these colors are:



- **Gray:** The sensor is not used.
- **Red:** An event occurs when there is a lot of reflected light.
- **White:** An event occurs when there is little reflected light.



Figure 3.2: The bottom of the Thymio with two ground sensors at the front



Information

The gray, red and white colors used in a block are arbitrary and others could have been chosen.

To cause the robot to stop at the border of the table when there is little reflected light, click the squares until they are white and create the following event-action pair:



Place the robot near from the edge of the table, facing the edge, and touch the front button. The robot should move forward and stop before falling off the table.



Exercise 3.1

Experiment with the speed of the robot. At maximum speed, is the robot still able to stop and not fall off the table? If not, at what speed does the robot start to fall off? Can you stop the robot from falling off when it is going backwards?



Trick

When I ran the program, the robot *did* fall off. The reason was that my nice desk has a rounded edge; by the time that the robot detected a low light level, it was no longer stable and tipped over. The solution was to place a strip of black tape close to the edge of the desk.

Chapter 4


A Pet Robot

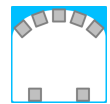
The robots we build in this chapter are called *autonomous robots*. They display independent behavior that is normally associated with living things like cats and dogs. The behavior is achieved by *feedback*: the robot will sense that something occurs in the world and modify its behavior accordingly.

The robot obeys you

First, we will program the robot to obey. Normally, the robot stays in place without moving; when it detects your hand in front of it, it moves towards your hand.

There are five horizontal distance sensors on the front of the Thymio robot and two on the rear of the robot. They are similar to the ones under Thymio that we have used in Chapter 3. Bring your hand slowly towards the sensors; when it gets close, a red light will appear around the sensors that detect your hands (Figure 4.1).

The block  is used to sense if something is close to the sensor or not. In either case it causes an event to occur. The small gray squares (five on the front and two on the rear) are used to specify when an event occurs. Clicking on a square changes it from gray to white to red and back to gray. For this block, the meaning of these colors is:



- **Gray:** The value of the sensor does not influence the program.
- **Red:** An event occurs when the sensor detects an object close to it.
- **White:** An event occurs when the sensor detects that there is *no* object nearby.



Figure 4.1: The front of the Thymio. Two sensors detect the fingers.

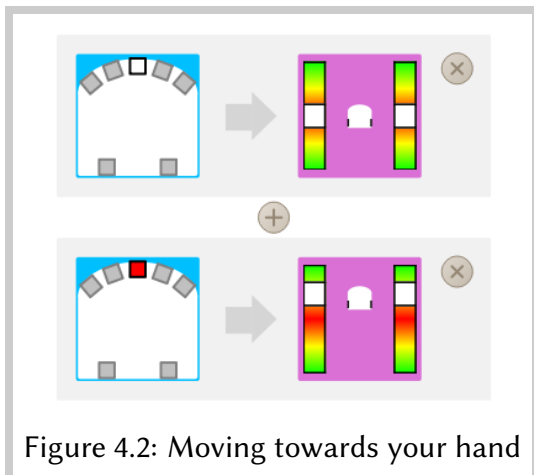


Figure 4.2: Moving towards your hand

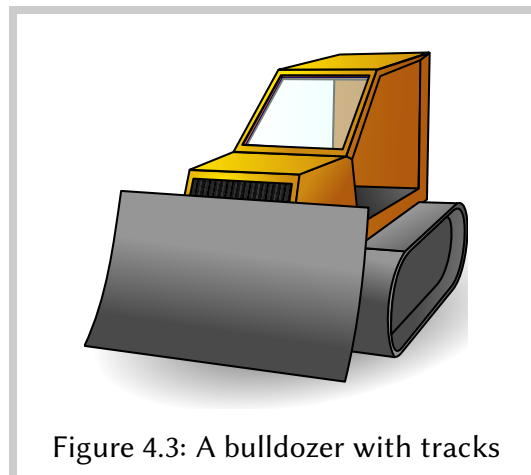


Figure 4.3: A bulldozer with tracks

Ground sensors and horizontal sensors

Be careful not to confuse the behavior of the horizontal sensors with the behavior of the ground sensors.


- For the horizontal sensors, the white square specifies that an event will occur if there is *nothing nearby*, while the red square specifies that an event will occur if there is *something nearby*.
- For the ground sensors, the white square specifies that an event will occur if *only a little light is reflected from the surface*, while the red square specifies that an event will occur *if a lot of light is reflected from the surface*. The physical principle of these two types of sensors is similar, but because of their different placement their behavior is different.

To implement the behavior, we need the two event-action pairs shown in Figure 4.2. In the first pair, the center front sensor is white and the associated action is that the motors are off. Therefore, when the robot does not see anything, it will not move, and it will stop if it had been moving. In the second pair, the center front sensor is red and the sliders of the motor block are dragged to the top. Therefore, when you bring your hand near the front of the robot, an event occurs that causes both motors to run quite fast and the robot to move forward.

Steering the Thymio robot

The Thymio robot does not have a steering wheel like a car or a handlebar like a bicycle. So how can it turn? The robot uses *differential drive*, which is familiar from tracked vehicles like the bulldozer (Figure 4.3). Instead of turning a handlebar a desired direction, the left and right tracks or wheels are driven by individual motors at *different* speeds. If the right track moves faster than the left one, the vehicle turns left, and if the left track moves faster than the right one, the vehicle turns right.


In VPL you can implement differential drive on the Thymio robot by setting the left

and right sliders of a motor action block, and therefore the wheel speeds, to different values. The greater the difference between the speeds, the tighter the turn. To achieve a large difference of speeds, you can drive one track forward and one track backwards. In fact, if one track moves forward at a certain speed, while the other track moves backwards at the *same* speed, the Thymio turns in place. For instance, in the motor action block , the left slider has been set for fast speed backwards, while the right slider has been set for fast speed forwards. The result is that the robot will turn to the left, as indicated by the small image of the robot.



Experiment with an event-action pair such as:



Set the left and right sliders, run the program and touch the center button; to stop the robot click on . Now you can change the sliders and try again.



Trick

The icon of Thymio in the center of the motor action block shows an animation of the movement of the robot when you move the sliders.

The robot likes you

A real pet follows you around. To make the robot follow your hand, add two additional event-action pairs: if the robot detects an object in front of its left-most sensor, it turns to the left, while if it detects an object in front of its right-most sensor, it turns to the right.

Program file **likes.aesl**

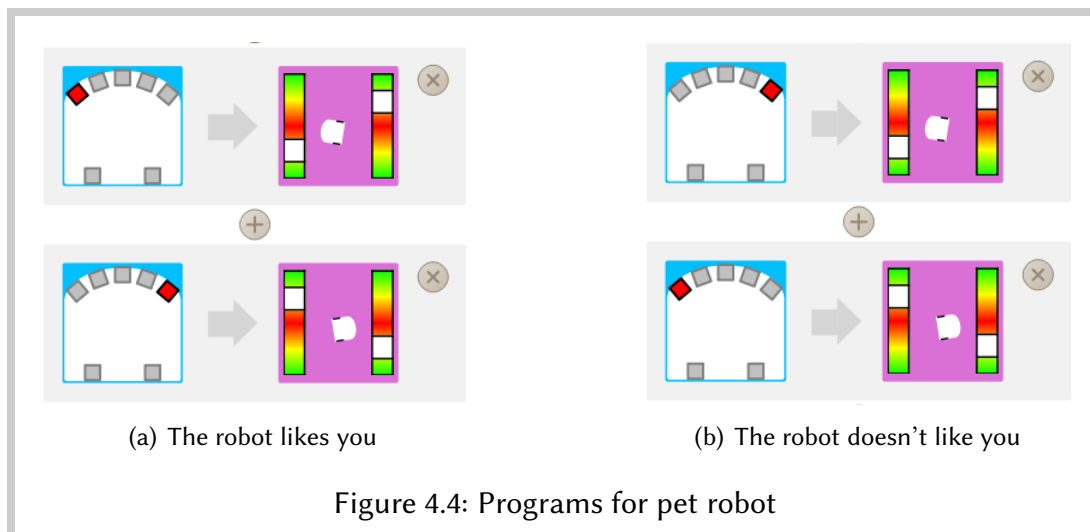
The program for the robot that likes you consists of two event-action pairs, as shown in Figure 4.4(a). Experiment with the sliders on each motor action block.



Exercise 4.1

Modify the behaviour of the pet robot so that it starts moving forward when the program is run and stops when it detects the edge of a table (or a strip of black tape).

As explained in Chapter 3, a lot of light will be reflected from a white surface, while very little light will be reflected from a black surface. You will have to experiment with the horizontal sensor block to determine when to click on a white square and when on a red square, depending on the floor or table where you place the robot.



Exercise 4.2

What happens if you change the order of the event-action pairs that you used in the previous exercise?

The robot doesn't like you

Sometimes your pet may be in a bad mood and turn away from your hand. Write a program that causes this behavior in the robot.

Program file **does-not-like.aesl**

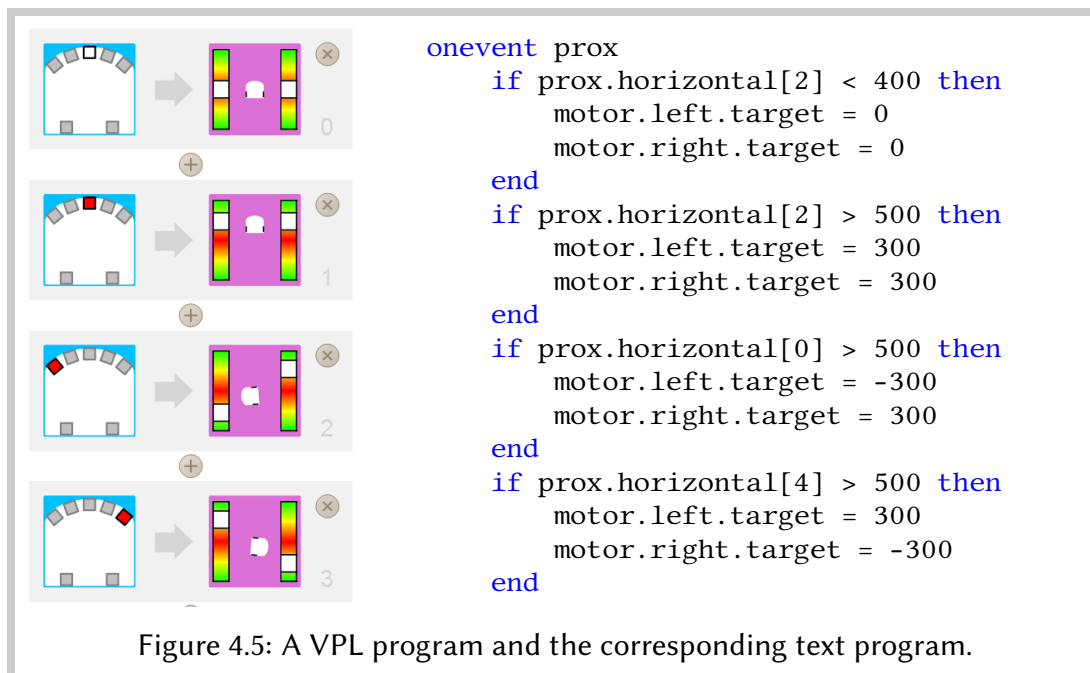
Open the program for the pet that likes you and exchange the association of the events with the actions. Detection of an obstacle by the left sensor causes the robot to turn right, while detection of an obstacle by the right sensor causes the robot to turn left, as shown in Figure 4.4(b).



Exercise 4.3

Experiment with the sensors. The front horizontal sensors are numbered 0, 1, 2, 3, 4 from the left of the robot to its right. The rear sensors are numbered 5 for the left one and 6 for the right one. Instead of using sensors 0 and 4 as before:

- Use sensors 1 and 3 to turn the robot left and right, respectively.
- Use both sensors 0 and 1 to turn the robot left and both sensors 3 and 4 to turn the robot right.
- Add event-action pairs for the rear sensors 5 and 6.



Setting the sliders precisely (advanced)

It is difficult to set the sliders precisely so that, for example, both motors run at the same speed. By looking at the translation of the event-action pairs into a textual program you can improve the precision. Figure 4.5 shows the program where the pet likes you and follows you around along with the text translation at the right of the VPL window. This text is modified automatically when you edit the event-action pairs.

The line `onevent prox` means: whenever the event of sampling the horizontal distance sensors (the *proximity* sensors, abbreviated *prox*) occurs (it occurs 10 times a second), the lines that follow will be run.

When the event happens, Thymio checks the values of the sensors using a condition of type `if ... then ... end`. It starts by testing the sensor number 2 (front center) as we see from `prox.horizontal[2]`. If this value is lower than 400, then Thymio sets the speeds of the left and right motors to 0 using the statements `motor.left.target = 0` and `motor.right.target = 0`. Every `if ... then ... end` block tests a specific sensor and executes or not the associated action, as a function of the result of the test. Hence it corresponds to an event-action pair:

0. tests if nothing is in front ; if that is true, Thymio stops.
1. tests if something is in front ; if that is true, Thymio goes forward.
2. tests if there is something at left ; if that is true, Thymio turns left.
3. tests if there is something at right ; if that is true, Thymio turns right.

Finally, once the Thymio has read all these sensors, it waits for the next event `prox` and starts these tests again, indefinitely.

To write programs in text mode, use the AsebaStudio environment (Chapter 10).



Trick

By moving the sliders on the motor action blocks, you will see that the target speeds of the motors (`motor.X.target`) jump by steps of 50 in the range -500 to 500 . By moving the sliders carefully, you can set the speeds to any of these values.

Chapter 5



The Robot Finds Its Way by Itself

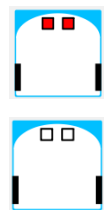
A hike in a mountain is a simple activity: get a pair of hiking shoes and follow a path. For a robot, following a path on the ground can also be very useful. Consider a warehouse with robotic carts that bring objects to a central dispatching area. There are lines painted on the floor of the warehouse and the robot receives instructions to follow certain lines until it reaches the storage bin of the desired object. To do so, it must see these lines. Let us write a program that causes the robot to follow a line on the floor.

Program file **follow-line.aesl**

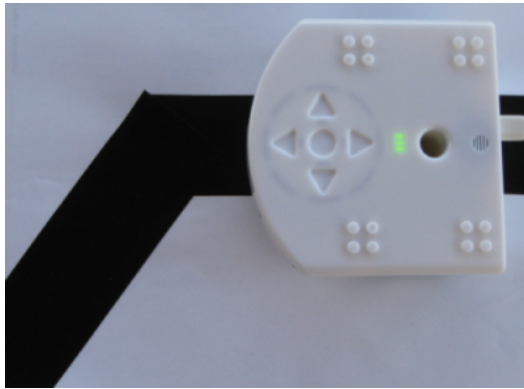
The line-following task brings out all the uncertainty of constructing robots in the real world: The robot must deal with uncertainty in its perceptions and its actions. For instance, the line might not be perfectly straight, dust may obscure part of the line, or dirt may cause one wheel to move more slowly than the other one. To follow a line, the robot must use a *controller* that decides how much power to apply to each motor depending on the data received from the sensors.

The line and the robot

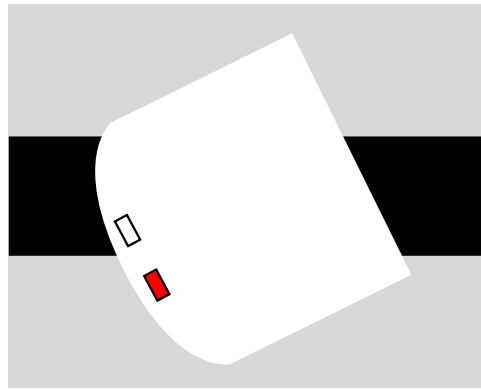
To follow a line, we use the ground sensors (Chapter 3). Remember that these work by sending infrared light (invisible to human eye) and measuring how much is reflected back. If the floor is light-colored, the sensor will detect a lot of reflected light and the event  will occur. We need a line that will cause an event to occur when there is little reflected light . This is easy to do by printing a black line, by painting it or by placing black electrician's tape on the floor (Figure 5.1(a)). The line must be wide enough so that both ground sensors will sense black when the robot is successfully following the line. A width of 5 centimeters is sufficient for the robot to follow the line even if there are small deviations.



First, we cause the Thymio to move forward whenever *both* sensors detect a dark surface (it is on the line) and stop whenever *both* sensors detect a light surface (it is not the line). The event-action pairs are shown in Figure 5.2(a).

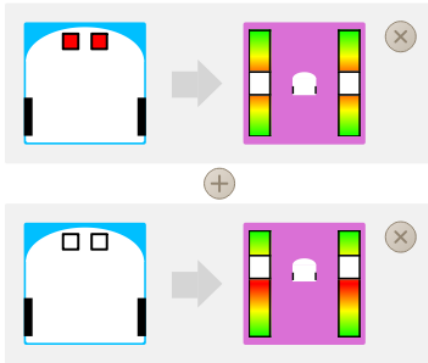


(a) Thymio following a line of tape

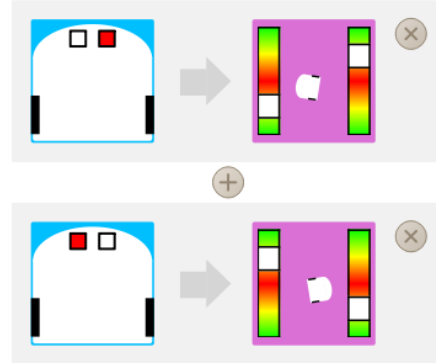


(b) The left sensor is off the tape and the right sensor is on the tape

Figure 5.1: Thymio on a black tape



(a) Start and stop the robot



(b) Correcting deviations

Figure 5.2: A program for line following



Trick

Make sure that you use a USB cable that is long enough (say, two meters), so that the Thymio can stay connected to the computer even as it moves. You can find extension cables in any computer shop.

Your first controller

The next step is to program the controller that follows the line:

- If the robot moves off the tape to the *left*, as in Figure 5.1(b), the *left* sensor will detect the floor while the *right* sensor is still detecting the tape; in that case the robot should turn slightly to the *right*.

- If the robot moves off the tape to the *right*, the *right* sensor will detect the floor while the *left* sensor is still detecting the tape; in that case the robot should turn slightly to the *left*.

Two event-action pairs are needed, as shown in Figure 5.2(b).

Setting the parameters

It is easy to see that if the robot runs off the left edge of the tape, it has to turn to the right, as in Figure 5.1(b). The true question is how tight should the turn be? If the turn is too gentle, the right sensor might *also* run off the tape before the robot turns back; if the turn is too aggressive, it might cause the robot to run off the other end of the tape. In any case, aggressive turns can be dangerous to the robot and whatever it is carrying.

In this program, you can set the speeds of the left and right motors in each motor action block. You will need to experiment with these values until the robot runs *reliably*. Here, reliably means that the robot can successfully follow the line several times. Since each time you place the robot on the line you might place it at a slightly different position and pointing in a slightly different direction, you need to run several tests to make sure that the program works.

There are many ways to configure the motor action blocks. The forward speed of the robot on the line is an important parameter. If it is too fast, the robot can run off the line before the turning actions can affect its direction. However, if the speed is too slow, no one will buy your robot to use in a warehouse.

If the Thymio runs off the line, what should it do? If it makes a sharp turn (one motor goes forward and the other backwards), the robot will return quickly to the line, but its movements will be very jerky. On the other hand, if the robot makes a gentle turn (one motor goes slightly faster than the other), the robot will move smoothly but it may lose the line. You will have to experiment to find good compromises.



Exercise 5.1

The robot stops when both ground sensors detect that they are off the tape. Modify the program so that the robot makes a gentle left turn in an attempt to find the tape again. Try it on a tape with a left turn like the one shown in Figure 5.1(a). Try increasing the forward speed of the robot. What happens when the robot gets to the end of the tape?

Exercise 5.2

Modify the program from the previous exercise so that the robot turns right when it runs off the tape. What happens?

It would be nice if we could *remember* which sensor was the last one to lose contact with the tape in order to cause the robot turn in the correct direction to find the tape again. In Chapter 8 we will learn how Thymio can remember information.

Exercise 5.3

Experiment with different arrangements of the lines of tapes:

- Gentle turns;
- Sharp turns;
- Zigzagging lines;
- Wider lines;
- Narrow lines.

Run competitions with your friends: Whose robot successfully follows the most lines? For each line, whose robot follows it in the shortest time?

Exercise 5.4

Discuss what effect the following modifications to the Thymio would have on the ability of the robot to follow a line:

- Ground sensing events occur more often or less often than 10 times a second;
- The sensors are further apart or closer together;
- There are more than two ground sensors on the bottom of the robot.

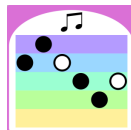
Chapter 6

Bells and Whistles

Let us take time out from complicated tasks like line following and have some fun with the robot. We show how the Thymio can play music, respond to a sound or react when it is tapped.

Playing music

The Thymio robot contains a sound synthesizer and you can program it to play simple tunes using the music action block:



Program file **bells.aesl**

You won't become a new Beethoven—you can only play a sequence of notes, on five tones and of two different lengths—but you can compose a tune that will make your robot stand out. Figure 6.1 shows two event-action pairs that respond with a tune when the front or back button is touched. There is a different tune associated with each event.

The small circles are the six notes. A black note is a short note and a white note is a long note; to change from one length to the other, click on the circle. There are five

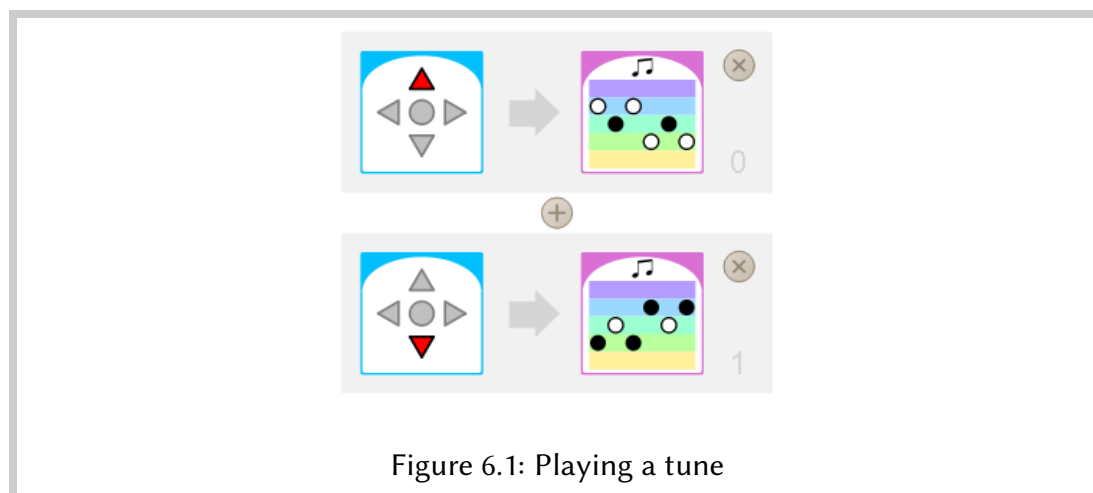



Figure 6.1: Playing a tune

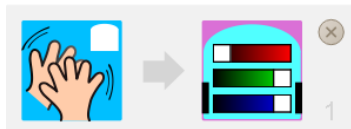
colored horizontal bars, representing five tones. To move a circle to one of bars, click on the *bar* above or below the circle. Don't try to drag and drop a note; it won't work.

Exercise 6.1

Write a program that will enable you to send a message is [Morse code](#). Letters in Morse code are encoded in sequences of long tones (*dashes*) and short tones (*dots*). For example, the letter V is encoded by three dots followed by one dash.

Controlling your robot by sound

The Thymio has a microphone. The event  will occur when the microphone senses a loud noise, for example, from clapping your hands. The following event-action pair will turn on the bottom lights when you clap your hands:



Trick


In a noisy environment you may not be able to use this event, because the sound level will always be high and cause repeated events.

Exercise 6.2

Write a program that causes the robot to move when you clap your hands and to stop when you touch a button.

Then write a program that does the opposite: starts when you touch a button and stops when you clap your hands.

Good job, robot

Pets don't always do what we ask them to do. Sometimes they need a pat on the head to encourage them. You can do the same with your robot. The Thymio contains a tap sensor that causes the event  to occur in response to a quick tap on the top of the robot. For example, the following event-action pair causes the top lights to turn on when you tap the top of the robot:



Construct a program from this event-action pair and the following pair that turns on the bottom lights when you clap your hands:



Program file **whistles.aesl**

Can you turn on just the top lights? This is difficult to do: a tap causes a sound that can be loud enough to cause the bottom lights to be turned on as well. With a little practice I was able to tap the robot gently enough so that the sound was not considered an event.



Exercise 6.3

Write a program that causes the robot to move forward until it hits a wall.

Make sure that the robot **moves slowly** so that it doesn't damage itself.

Chapter 7

A Time to Like

In Chapter 4 we programmed a pet robot which either did or did not like us. Let us consider a more advanced behavior: a shy pet who can't make up its mind whether it likes us or dislikes us. Initially, the pet will turn towards our outreached hand, but then it will turn away. After a while, it will reconsider and turn back in the direction of our hand.

Program file **shy.aesl**

The behavior of the robot is as follows. When the right button is touched the robot turns to the right. When it detects your hand, it turns to the left but after a while it regrets its decision and turns back. We know how to construct event-action pairs for the initial turn:




and for turning away when your hand is detected:



The behavior of turning back “after a while” can be broken down into two parts:

- *When* the robot starts to turn away → *start a timer* for two seconds.
- *When* the timer runs down to zero → *turn* to the right.

We need a new *action* for the first part and a new *event* for the second part.


The action is to set a *timer*, which is like an alarm clock . Normally, we set an alarm clock to an absolute time, but when I set the alarm clock in my smartphone to an absolute time like 07:00, it tells me the relative time: “Alarm set for 11 hours, and 23 minutes from now.” The timer block works the same way: you set the timer for a certain number of seconds from when the event occurs and then the action happens. The timer can be set for up to four seconds. Click anywhere within the black circle showing the face of the clock (but not on the black circle itself). There will be a short animation and then the amount of time until the alarm will be colored blue.



The event-action pair for this first part of the behavior is:



The timer is set for two seconds. When the event of detecting your hand occurs, there will be two actions: turning the robot to the left and setting the timer.

The second part of the behavior needs an event that occurs when the alarm goes off, that is, when the amount of time set on the timer runs down to zero. The event block  shows a ringing alarm clock.



Here is the event-action pair to turn the robot anew to the right when the timer runs down:



Exercise 7.1

Write a program that causes the robot to move forward at top speed for three seconds when the forward button is touched; then it runs backwards. Add an event-action pair to stop the robot by touching the center button.

Chapter 8

States: Don't Always Do the Same Thing (Advanced)

A program in VPL is a list of event-action pairs. All the events are checked periodically and the appropriate actions are taken. This limits the programs that we can create; to go further we need a way to specify that certain event-action pairs are active at a certain time, while others are not. For example, in Chapter 5, when the robot ran off the tape, we wanted it to turn left or right in order to search for the tape with the direction depending on which side it ran off.

States are supported in the *advanced* mode of VPL. Click on 🎓 before beginning to work on the following projects.



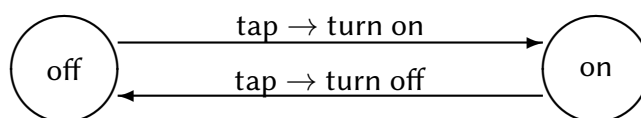
Tap, tap

In many programs, we used one button to start the robot's behavior and another to stop it. Consider, though, the power switch on my computer: The same switch is used to turn the computer on and off; the switch *remembers* whether it is in the state **on** or the state **off**. The switch includes a small green light which indicates its current state.

Write a program that turns the robot's lights on when it is tapped and turns them off when tapped again.

Program file **tap-on-off.aesl**

It is convenient to display the required behavior in a *state diagram*:



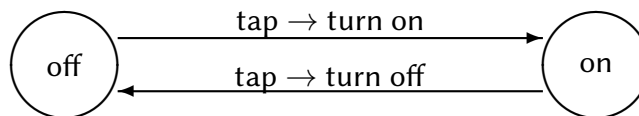
In the diagram there are two states indicated by circles labeled with the names of the states **off** and **on**. From state **off** the robot can go to state **on** and back, but only by following the instructions on the arrows. The instructions describe when a transition from one state to another can occur and what happens when it does occur:

- When the robot is in state **off** *and* the *tap* event occurs → turn the lights *on* *and* go to state **on**.

- When the robot is in state **on** *and* the *tap* event occurs → turn the lights *off* *and* go to state **off**.

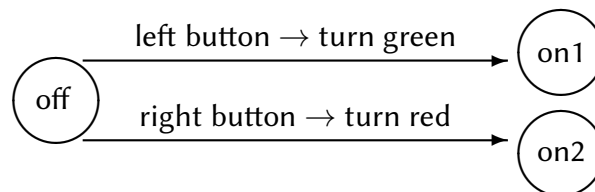
The emphasized word “**and**” before the arrow → means that there are *two conditions* that must be true in order for the transition to be taken. (a) The robot must be in a certain state and (b) the event must occur. When both conditions are true, the transition is taken, causing both the state to change and the action written after the arrow → to be performed.

It is important to realize that the two parts of the condition are independent. In the above diagram (repeated here):



the event *tap* appears twice, but the action caused by the occurrence of this event *depends on* which state the robot is in.


Similarly, in a single state, different events can cause different actions and transitions to different new states. In the following diagram:



touching the left button in the state **off** causes the green light to be turned on and a change to state **on1**, whereas touching the right button *in the same state* causes a different action, the red light is turned on, and a change to a different state, **on2**.

Implementing state diagrams with event-action pairs

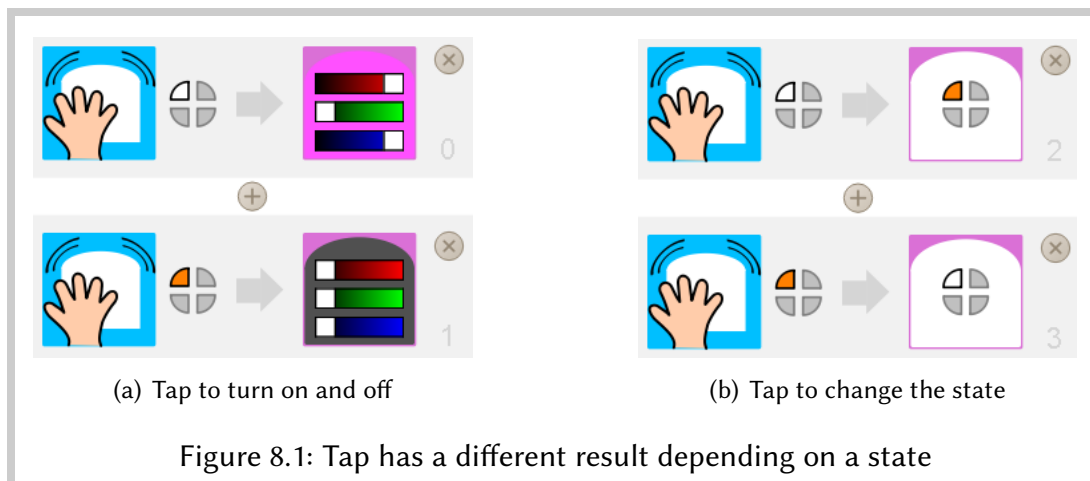
Figure 8.1 shows the implementation of the behavior described in the state machine as event-action pairs.

In the first event-action pair, the event is composed of the tap block together with an indication of the state :



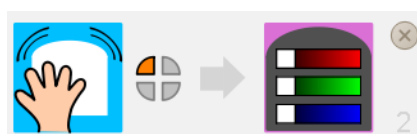
A state is indicated by four quarters of a circle, each of which can be either on (orange) or off (white). In this program, we will use the left-front quarter to indicate whether






the robot's top light is off or on. In this pair, this quarter is colored white, meaning that the robot's light is off. Therefore, the meaning of the this pair is: if the robot is tapped and the light is off, turn it on.

Similarly, the second event-action pair means: if the robot is tapped and the light is on, turn it off:

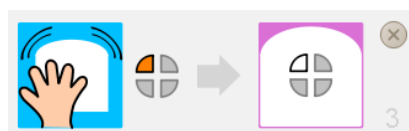


If you look again at the state diagram, you will see that only half the job is done. Indeed, when turning the light on or off, we also have to change the state of the robot from **off** to **on** or from **on** to **off**. For that we create two additional event-action pairs using the *state* action block , as shown in Figure 8.1(b).

The meaning of the first one is: *when* the robot is tapped *and* the state is **off**, change the state to **on**:



Similarly, the meaning of the second one is *when* the robot is tapped *and* the state is **on**, change the state to **off**:





Referring the complete program with four event-action pairs in Figure 8.1, we see that each event causes both an action on the light and a change of the state of the robot. Both the action and the change of state depend on the state the robot is in, called *current state*.

How many states can the robot be in?

The state is indicated by a circle divided into four quarters. When used in an event or in the state action block, each quarter can be:

- **White:** the quarter is *off*;
- **Orange:** the quarter is *on*;
- **Gray:** the quarter is ignored.

For example, in , the left-front and right-rear quarters are on, the right-front one is off and the left-rear one is not taken into account, meaning that if  is associated with an event block, the event will occur if the state is either set to:



Since each of the four quarters can be either on or off, there are $2 \times 2 \times 2 \times 2 = 16$ possible states:

(off, off, off, off), (off, off, off, on), (off, off, on, off),
...
(on, on, off, on), (on, on, on, off), (on, on, on, on).

Figure 8.2(a) enumerates graphically all these states.



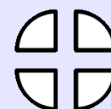
Important information

The current state of the robot is displayed in the circle LEDs on the top of the robot. Figure 8.2(b) shows the robot in the state (on, on, on, on).



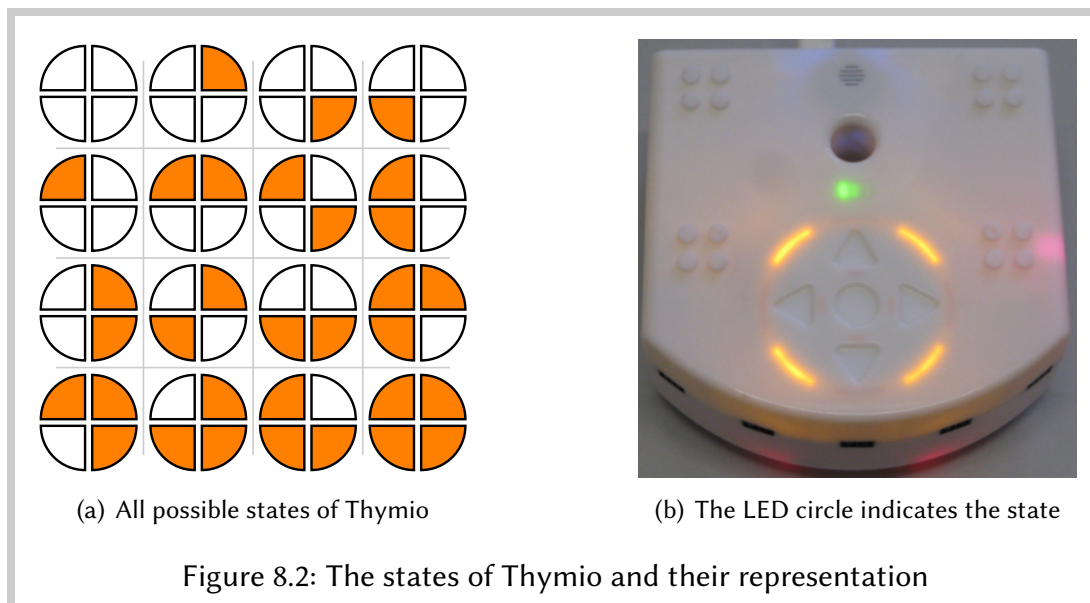
Information

When a program is run, the initial state is (off, off, off, off):



Trick

If you do not use all possible 16 states, but only 2 or 4, for example, you are free to decide which quarters you use to represent your state. In addition, if you have two different things you want to encode, and each of them has two possible values, you can use two quarters independently. That is why the ability to *ignore* a quarter is very useful!

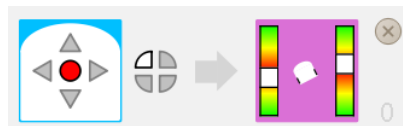


Get the mouse

Write a program that causes the robot to turn from right to left, searching for a mouse (or another object). If the robot detects a mouse with its leftmost sensor it continues the search until the mouse is detected by its rightmost sensor. Then, it positions itself facing the mouse, as shown in Figure 8.3(a).

Program file **mouse.aesl**

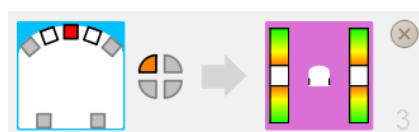
The following event-action pair causes the robot to turn left:



This only occurs when the left-front quarter is off; initially all quarters of the state are off.

The first event-action pair in Figure 8.3(b) waits until the mouse is detected at the rightmost sensor. Note that the small square next to it is colored white so that the event occurs only when the rightmost sensor alone detects the mouse. The second event-action pair in the Figure 8.3(b) changes the state.

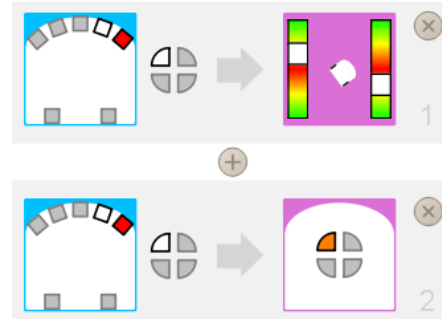
The final event-action pair in the program halts the robot when the mouse is directly facing the center sensor:



Why does the event in this pair need to depend on the state? The reason is that the central sensor will also detect the mouse in its initial scan from right to left. We want the robot to perform a full scan before returning to the position of the mouse, so it



(a) The cat has found the mouse



(b) Searching with the rightmost sensor

Figure 8.3: The robot cat is looking for the mouse

is necessary that this first detection of the mouse be ignored. This is achieved by stopping the scan only when the state is **on** and this is set only when the full scan has been completed.

💡 Trick

You will have to experiment with the distance of the mouse to the robot. If it is too close to the robot, the sensors on either side of the central sensor will also detect the mouse, while the event requires that they *not* detect it.

🔧 Exercise 8.1

Write a program that causes the robot to dance: it turns left in place for two seconds and then turns right in place for three seconds. These movements are repeated indefinitely.

🔧 Exercise 8.2 (Difficult)

Modify the line-following program from Chapter 5 so that the robot turns left when it leaves the right-hand side of the line and turns right when it leaves the left-hand side of the line.

Chapter 9

Counting (Advanced)

In this chapter we show how states of the Thymio robot can be used to count numbers and even perform simple arithmetic.

The design and implementation of the projects will not be presented in detail. We assume that you have enough experience by now to develop them yourself. The source code of working programs is included in the archive, but don't look at them unless you really have difficulties solving a problem.

These projects use the *clap* event to change states and the default behaviour of the circle LEDs to display the state:



Important information

The current state of the robot is displayed in the circle LEDs on the top of the robot. Figure 8.2(b) shows the robot in the state (**on**, **on**, **on**, **on**).

Feel free to change either of these behaviours.

Odd and even

Program

Choose one of the quarters of the state. It will be **off** (white) if the number of claps is even and **on** (orange) if the number of claps is odd. Touching the center button will reset to even (since zero is an even number).

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

Our method of counting demonstrates the concept of *modulo 2 arithmetic*. We count starting from 0 to 1 and then back to 0. The term *modulo* is like the term *remainder*: if there have been 7 claps, then dividing 7 by 2 gives 3 and remainder 1. We only keep the remainder 1.

In modulo 2 arithmetic, 0 and 1 are often called even and odd, respectively.


Another term for the same concept is *cyclic arithmetic*. Instead of counting from 0 to 1 and then from 1 to 2, we *cycle* back to the beginning: 0, 1, 0, 1,

These concepts are very familiar because they are used in clocks. Minutes and seconds are computed modulo 60 and hours are computed modulo 12 or 24. Thus, the second after 59 is not 60; instead, we cycle around and start counting from 0 again. Similarly,

the hour after 23 is not 24, but 0. If the time is 23:00 and we agree to meet after 3 hours, then the time set for the meeting is 26 modulo 24, which is 02:00 in the morning.

Counting in unary

Modify the program to count modulo 4. There are four possible remainders, 0, 1, 2, 3. Choose three quarters, one each to represent the values 1, 2 and 3; the value 0 will be represented by setting all quarters to **off**.

This method of representing numbers is called *unary representation* because different elements of a state represent different numbers. We often use unary representation to keep track of the count of some objects; for example,  represents 6.

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



Exercise 9.1

How high can we count on the Thymio using unary representation?

Counting in binary

We are very familiar with *based representation*, in particular base 10 (decimal) representation. The symbols 256 in base 10 representation don't represent three different objects. Instead, the 6 represents the number of 1's, the 5 represents the number of $10 \times 1 = 10$'s, and the 2 represents the number of $10 \times 10 \times 1 = 100$'s. Adding these factors gives the number two hundred and fifty-six. Using base 10 representation, we can write very large numbers in a compact representation. Furthermore, arithmetic on large numbers is relatively easy using the methods we learned at school.

We use base 10 representation because we have 10 fingers so it is easy to learn to use the representation. Computers, however, have two "fingers" (**off** and **on**) so base 2 arithmetic is used in computation. Base 2 arithmetic looks strange at first; while we use the familiar symbols 0 and 1 also used in base 10, the rules for counting are cyclic at 2 instead of cyclic at 10:



0, 1, 10, 11, 100, 101, 110, 111, 1000, ...

Given a base 2 number such as 1101, we compute its value from right to left just as in base 10. The rightmost digit represents the number of 1's, the next digit represents the number of $1 \times 2 = 2$'s, the third digit represents the number of $1 \times 2 \times 2 = 4$'s, and the leftmost digit represents the numbers of $1 \times 2 \times 2 \times 2 = 8$'s. Therefore, 1101 represents $1 + 0 + 4 + 8$, which is thirteen, represented in base 10 as 13.

Program

Modify the program for counting modulo 4 to use binary representation

Program file **count-to-four-binary.aesl**

We only need two quarters of the state to represent the numbers 0–3 in base 2. Let the upper right quarter represent the number of 1's, **off** (white) for none and **on** (orange) for one, and let the upper left quarter represent the number of 2's. For example,  represents the number 1 and  represents the number 2. If both quarters are white, the state represents 0, and if both quarters are orange, the state represents 3.

There are four transitions $0 \rightarrow 1$, $1 \rightarrow 2$, $2 \rightarrow 3$, $3 \rightarrow 0$, so four event-action pairs are needed, in addition to a pair to reset the program when the center button is touched.



Trick

The two bottom quarters are not used, so they are left gray and are ignored by the program.



Exercise 9.2

Extend the program so that it counts modulo 8. The lower left quarter will represent the number of 4's.

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



Exercise 9.3

How high can we count on the Thymio using binary representation?

Adding and subtracting

Writing the program to count to 8 is quite tedious because you had to program 8 event-action pairs, one for each transition from n to $n + 1$ (modulo 8). Of course, that is not how we count in a based representation; instead, we have methods for performing addition by adding the digits in each place and carrying to the next place. In base 10 representation:

$$\begin{array}{r} 387 \\ +426 \\ \hline 813 \end{array}$$

and similarly in base 2 notation:

$$\begin{array}{r} 0011 \\ +1011 \\ \hline 1110 \end{array}$$

When adding 1 to 1, instead of 2, we get 10. The 0 is written in the same column and we carry the 1 to the next column to the left. The example above shows the addition of 3 (=0011) and 11 (=1011) to obtain 14 (=1110).

Program

Write a program that starts with a representation of 0. Each clap adds 1 to the number. The addition is modulo 16, so adding 1 to 15 results in 0.

Guidance:

- The lower right quarter will be used to represent the number of 8's.
- If the upper right quarter representing the number of 1's shows 0 (white), simply change it to 1 (orange). Do this regardless of what the other quarters show.
- If the upper right quarter representing the number of 1's shows 1 (orange), change it to 0 (white) and then carry the 1. There will be three event-action pairs, depending on the location of the *next* quarter showing 0 (white).
- If all quarters show 1 (orange), the value of 15 is represented. Adding 1 to 15 modulo 16 results in 0, represented by all quarters showing 0 (white).

Program file **addition.aesl**



Exercise 9.4

Modify the program so that it starts with the value 15 and subtracts one at each clap down to zero, and then cyclically back to 15.

Program file **subtraction.aesl**



Exercise 9.5

Place a sequence of short segments of black tape on a light surface (or white tape on a dark surface). Write a program that causes the Thymio to move forward and stop when it has detected the fourth tape.

This exercise is not easy: the strips of tape have to be sufficiently wide so that the robot detects them, but not so wide that more than one event occurs per strip. You will also have to experiment with the speed of the robot.

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

Chapter 10

What Next?

This tutorial has introduced the Thymio robot and the Aseba/VPL environment. The VPL environment with its simple visual programming is intended for beginners. To develop more advanced programs for the robot, you will want to learn how to use the Aseba Studio environment (Figure 10.1).

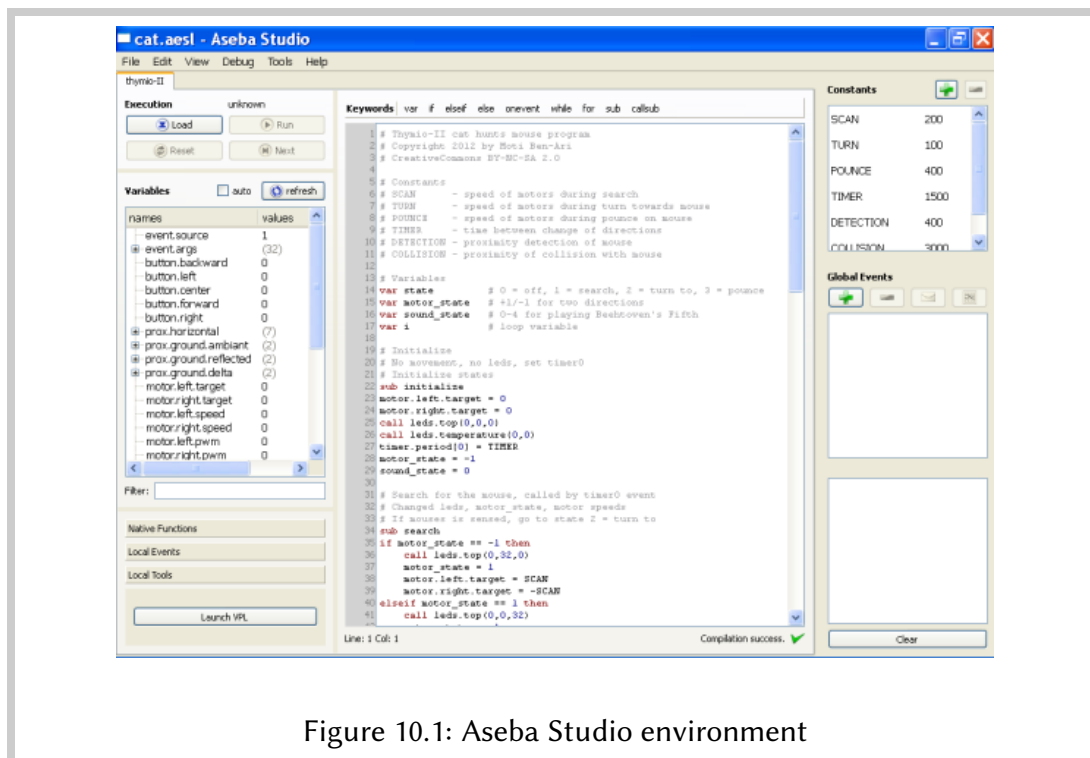


Figure 10.1: Aseba Studio environment

Programming in Aseba Studio is also based upon the concepts of events and actions. Since VPL programs are translated into textual programs, everything you learned in this tutorial is available in Studio, but many other programming features are available:

- You can control precisely when an event causes an action, depending, for example, on the amount of reflected light measured from a ground sensor or the distance from a horizontal sensor.
- You can specify that a single action consists of several different operations: controlling the motors, changing the state, setting thresholds, turning lights on and off, etc.

- You have the flexibility of a full programming language with variables, expressions, and control statements.

Aseba Studio gives you access to features of the Thymio that are not available in VPL:

- You can control all the lights such as the circle of lights surrounding the buttons.
- You have more flexibility in synthesizing sound.
- There is a temperature sensor.
- Instead of just sensing the shock of a tap, accelerometers can sense gravity and changes of speed in three dimensions.
- A remote control device can be used with the robot.

When you are working with Aseba Studio, you can open VPL by clicking on the button **Launch VPL** in the *Tools* tab at the bottom left of the window. You can import VPL programs into Aseba Studio simply by opening the file.

To use Aseba Studio, start from the *Programming Thymio II* page at:
<https://aseba.wikidot.com/en:thymioprogram>
and follow the link *Text Programming Environment*.

You can find many interesting projects at:
<https://aseba.wikidot.com/en:thymioexamples>.

★ **Have fun and learn a lot!**
Thank you for reading this tutorial!