Web Simulation of a Thymio Robot

Quentin Flückiger (`flucq1@bfh.ch`)

November 7, 2019

Declaration of Authorship

Management Summary

Contents

1 Introduction

2 Environment

3 Thymio

3.1 What is Thymio

Thymio is an educational robot that aims at improving early education (starting in primary school) in STEM (Science, Technology, Engineering and Mathematics), computational thinking, base computer science and researching the acknowledgement by kids of robots in their learning environment. The project also had technical aims, such as how to provide hardware modularity, fast reaction time amid perception and action, clear internal communication bus in a user-friendly way and streamline development for group robot, this includes direct changes to the robots' programs and parallel debugging wirelessly, transparently and cheaply.

The Thymio project is based on a collaboration between the MOBOTS group from the Swiss Federal Institute of Technology in Lausanne (EPFL) and the Lausanne Arts School (ECAL). MOBOTS being the Miniature Mobile Robots Group, they are mainly focused around system design for small robots of the kind. It started with a strange-looking pile of components, that were assembled on any kind of support and hold the name of "Monsieur Patate" (Sir Potato), most likely due to its appearance, that saw life during the first workshop between the two contributors. After what the first "Thymio" was developed, it was a four-block robot that could be self-assembled, but not self-programmed as it was coming with pre-programmed behaviours. It was used as a user study to gather feedback from clients to know what features needed to be implemented on the Thymio II.

The result is a robot with a complex and complete set of sensors and actuators. The National Centre for Competence in Research (NCCR) Robotics research program supported the development of the robot whereas Mobsya, a non-profit organization that creates a robot, software, and educational activities to broaden young people's mind about technology and science, oversees the production, distribution, and communication of said robot. Every step of the Thymio project is open-source and has a non-profit aim to enhance the quality of it with the user's project and research, and reduce the cost and augment the lifetime for educational platforms and materials.

3.2 How does it works

As seen in the figure above there exist two Thymio models, Thymio and Wireless Thymio. The difference between them lies in the ability of the second one to be programmed wirelessly, as its name suggests. To begin the creation of a program for the robot there exist two possibilities. The first one, and the most common one for the public is done by using the software Aseba and a connected Thymio. In this case, the robot needs to be

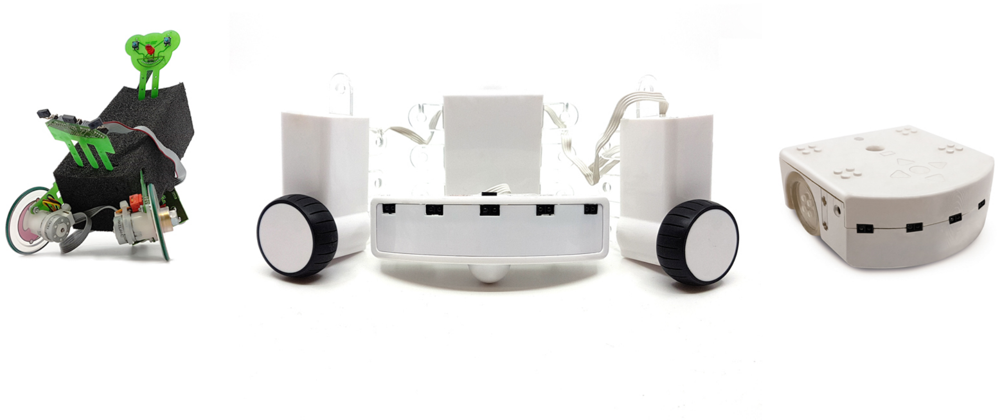


Figure 1: From left to right, ”Monsieur Patate”, Thymio, Thymio II

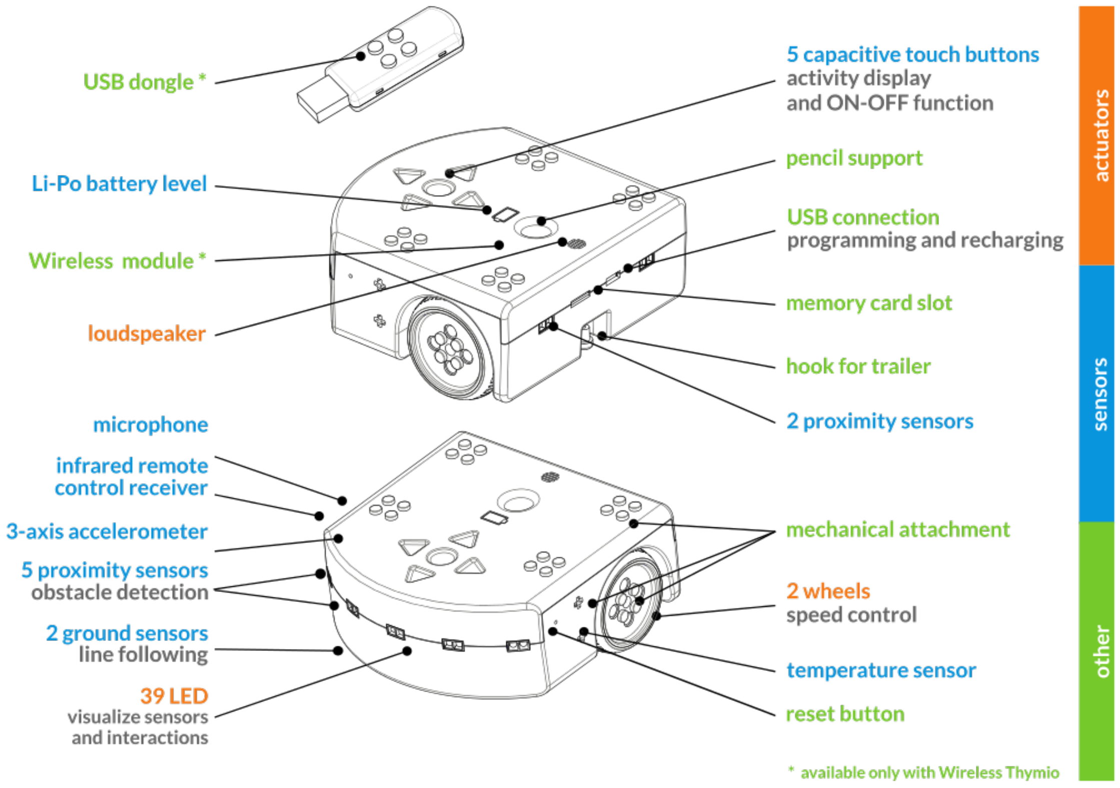

USB dongle *
Li-Po battery level
Wireless module *
loudspeaker
microphone
infrared remote control receiver
3-axis accelerometer
5 proximity sensors obstacle detection
2 ground sensors line following
39 LED visualize sensors and interactions
5 capacitive touch buttons activity display and ON-OFF function
pencil support
USB connection programming and recharging
memory card slot
hook for trailer
2 proximity sensors
mechanical attachment
2 wheels speed control
temperature sensor
reset button
* available only with Wireless Thymio
actuators
sensors
other

Figure 2: Thymio II sensor and actuator

plugged in via USB cable or USB dongle (possible only if it is the Wireless Thymio) and powered on. Then the software can be used to connect to said robot and start to program in one of the four different programming languages, that are: VPL, Blockly, Aseba, and Scratch. Once the program is ready and sent to the robot it will be available to play.

The second option is to use the work-in-progress Thymio Suite version. This software doesn't require a Thymio robot to be connected physically (or wirelessly) at all times as it has its own simulator built-in. The four said languages are still available, and one need to be chosen. After what comes the choice of connecting a physical Thymio or starting a simulation to emulate the programmed behaviour.

3.3 The different programming languages

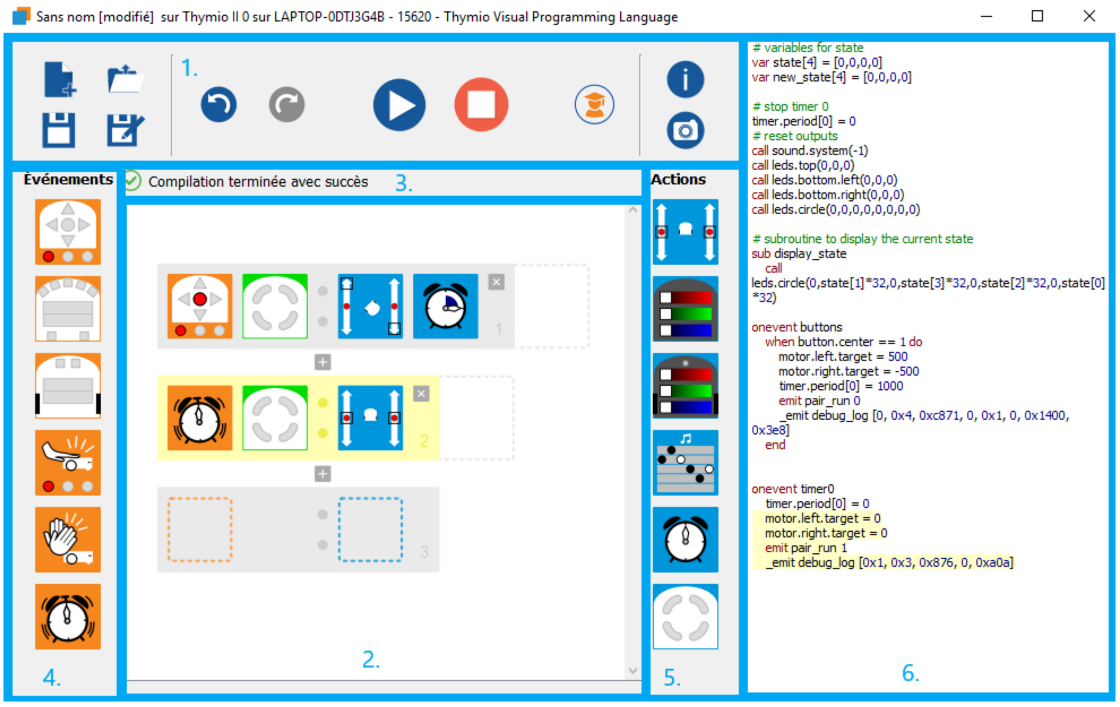
3.3.1 VPL

One of the four different possibilities to program the Thymio is by using the visual programming language, or VPL, developed by the creator of Aseba. A visual programming language is an abstraction of the more common way to program. It is based on the manipulation of program elements graphically that can be manipulated following some spatial grammar to create a program. VPLs are based on a set of entities and relations, whereas most of the time entities are represented by boxes, or other graphical objects, and relations by simple arrows. They can be categorized into icon-based, form-based and diagram-based languages depending on the extent of visual expression inside of it. The use of visual programming languages can be found in multiple areas, such as the game engine "Unreal Engine 4" where their system of Blueprints is created upon a node-based VPL, or "Microsoft SQL Server Integration Services". This abstraction allows easier access for neophytes, for example using graphic elements such as blocks, forms, diagrams, and others reduce drastically, if not eliminate, the syntactic errors made by the user.

In the case of the VPL developed by Aseba's team, and the one we are mostly interested in, we have a programming language based on two types of blocks: Event blocks and Action blocks. From those two are built the seventeen, respectively eleven event blocks and six action blocks, entities. One of the main goals of VPL for Thymio was to let people who cannot yet read the ability to start programming and discover this world.

To begin creating a program follow the first steps described in the "How does it works" section. Once the VPL option has been chosen and the Thymio Visual Programming Language window appears we are ready to go. The window is split into six different regions with each of their purposes.

1. A tool bar
2. A programming window
3. Console messages
4. The event blocks
5. The action blocks


Sans nom [modifié] sur Thymio II 0 sur LAPTOP-0DTJ3G4B - 15620 - Thymio Visual Programming Language
1.
Événements
Compilation terminée avec succès
3.
Actions
4.
2.
5.
6.

```
# variables for state
var state[4] = [0,0,0,0]
var new_state[4] = [0,0,0,0]

# stop timer 0
timer.period[0] = 0
# reset outputs
call sound.system(-1)
call leds.top(0,0,0)
call leds.bottom.left(0,0,0)
call leds.bottom.right(0,0,0)
call leds.circle(0,0,0,0,0,0,0,0)

# subroutine to display the current state
sub display_state
  call
leds.circle(0,state[1]*32,0,state[3]*32,0,state[2]*32,0,state[0]
*32)

onevent buttons
  when button.center == 1 do
    motor.left.target = 500
    motor.right.target = -500
    timer.period[0] = 1000
    emit pair_run 0
    _emit debug_log [0, 0x4, 0xc871, 0, 0x1, 0, 0x1400,
0x3e8]
  end


onevent timer0
  timer.period[0] = 0
  motor.left.target = 0
  motor.right.target = 0
  emit pair_run 1
  _emit debug_log [0x1, 0x3, 0x876, 0, 0xa0a]
```

Figure 3: Thymio VPL Event and Action blocks

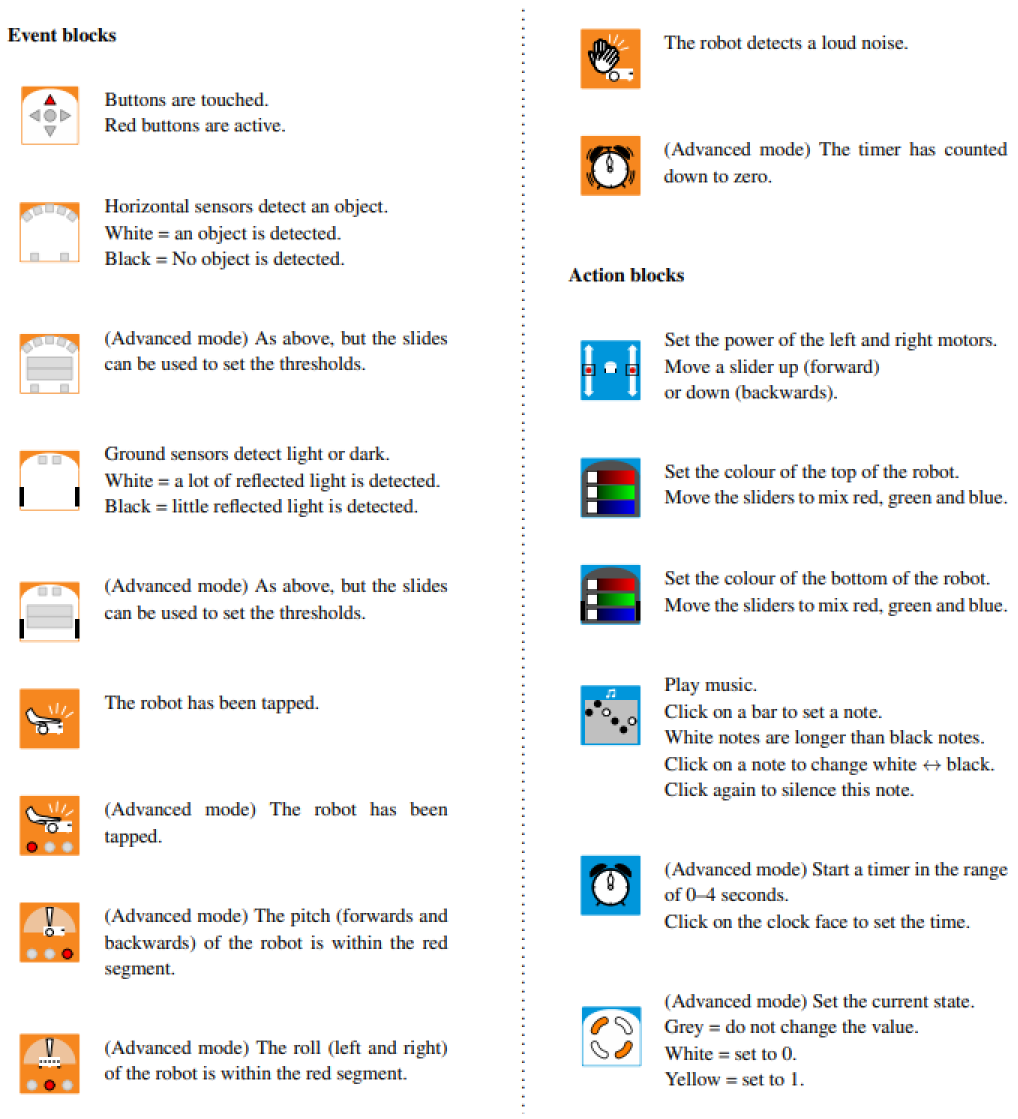
6. The program translated into AESL

At first, the programming window will be empty of blocks, containing just a placeholder with empty slots. This placeholder is the base of every Thymio VPL program, it contains exactly one event block and one or more action block. This means that whenever the event of the event block happens then the set of actions added to this placeholder will occur at the same time. For example, with the following pair:

Both wheels are powered to the maximum when the middle button is pressed. But more than one action can be attributed to one event, to do so simply drag another action block onto the previous pair, notice that the same block cannot be used twice for the same event. Here we turned the lights on top and set them to a complete green:

The maximum amount of action blocks we can add to an event is four, but we can add as many event blocks to our program as we want. Let us add two more event blocks to allow the robot to turn:

Now we have a basic behavior, go straight with green lights when the middle button is pushed, turn left on itself with orange lights when the left button is pushed, and at last turn right on itself with yellow lights when the right button is pushed.



Event blocks

Buttons are touched.
Red buttons are active.

Horizontal sensors detect an object.
White = an object is detected.
Black = No object is detected.

(Advanced mode) As above, but the slides can be used to set the thresholds.

Ground sensors detect light or dark.
White = a lot of reflected light is detected.
Black = little reflected light is detected.

(Advanced mode) As above, but the slides can be used to set the thresholds.

The robot has been tapped.

(Advanced mode) The robot has been tapped.

(Advanced mode) The pitch (forwards and backwards) of the robot is within the red segment.

(Advanced mode) The roll (left and right) of the robot is within the red segment.

The robot detects a loud noise.

(Advanced mode) The timer has counted down to zero.

Action blocks

Set the power of the left and right motors.
Move a slider up (forward)
or down (backwards).

Set the colour of the top of the robot.
Move the sliders to mix red, green and blue.

Set the colour of the bottom of the robot.
Move the sliders to mix red, green and blue.

Play music.
Click on a bar to set a note.
White notes are longer than black notes.
Click on a note to change white ↔ black.
Click again to silence this note.

(Advanced mode) Start a timer in the range of 0–4 seconds.
Click on the clock face to set the time.

(Advanced mode) Set the current state.
Grey = do not change the value.
White = set to 0.
Yellow = set to 1.

Figure 4: Thymio VPL Window

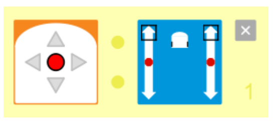


Figure 5: Event and one Action relation

By clicking the button with a student as an icon we enable the advanced mode that gives us more possibilities for multiple blocks. It raises the amount of action block from four to six as well.

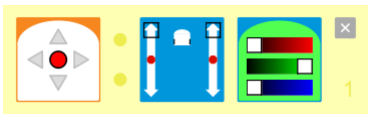


Figure 6: Event and multiple Actions relation

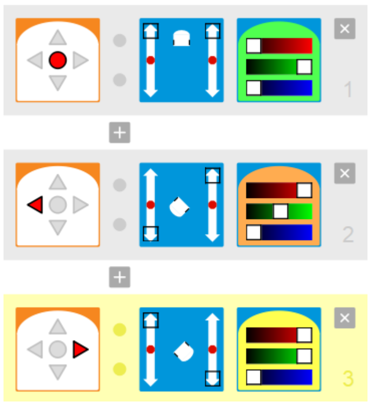


Figure 7: Event and Actions relations

Let us refactor a bit the program from before, we will change the program by making the robot look left then right and starting over again using timers. To help us develop a more interesting program we have now access to a condition, a four led light on top of the robot, using this and the timer we can behave depending on the state of the robot. For example, hereafter the middle button was pressed a timer will start and after a short amount of time, it will light one particular led. Afterward, the event "timer elapsed" will be triggered but which pair should the program execute, turning right or turning left? Hence comes the use of the condition as we will execute the part of the program that corresponds to the state of the condition light. In this example, it will go back and forth between the two pairs:

3.3.2 Blockly 4 Thymio

The second possibility is to use Blockly4Thymio which is an environment based on Blockly. Blockly was released in May 2012 and was initially a replacement for Open-Blocks for the MIT App Inventor. It is an open-source client-side library that allows its users to easily add a block-based visual programming language to an application or website. Blockly is not in itself a programming language but rather used to create one. Its design makes it flexible and it can support a large set of features. As it is a visual programming language, we find the same advantages as the first possibility, such for example applying programming principles with no regard towards syntactic error. Blockly is among the growing and most used visual programming environments because

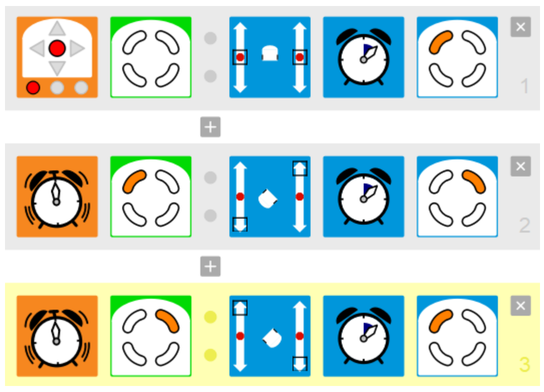


Figure 8: Advanced program

of a few important features. First, it can export the code generated with the blocks to one of the five following programming languages, as a built-in feature, JavaScript, Lua, Dart, Python, and PHP, and can be enhanced for any textual programming languages. The block pool can be expanded from its base pool or even reduced depending on the needs. The blocks are not restrained to only basic tasks and can implement sophisticated programming tasks. And it has been translated in over forty languages, and as well right-to-left versions.

Blockly includes a set of pre-defined blocks that can be used to develop with more ease the wanted application. They are arranged into eight families:

1. Logic: Blocks with Boolean definition, equality check, and conditions.

2. Loop: Blocks for loops.

3. Math: Blocks for numbers, arithmetic operation, a few basic math functions (for example cos, sin, square root) and some mathematical constant (Pi).

4. Text: Blocks to create text and text operations.

5. Lists: Blocks to create lists and standard list operation (length, get the value).

6. Color: Blocks with a color definition.

7. Variables: Blocks to create variables, and to set/get their values.

8. Functions: Blocks to create functions, with return value or not, and to call existing function.

Each block holds a pre-assigned shape, thus restraining its usage to certain situations as a ”hidden” way to control the syntax. Their shapes are defined by the different connections with other blocks, both external and internal, while external blocks describe what happens after or before, the internals describe what happens during or what are the arguments, logic. Following is a basic variable block with three external connectors, and a math block with the value of one, with one connector, that is assigned to the "Count" variable (the blocks need to be assembled).

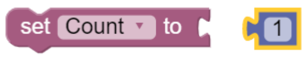


Figure 9: Variable block

Using the same logic as above we created a "Limit" variable with the value of 5 to demonstrate the next example. The block used is from the logic family and test whether the "Count" variable is smaller or equal to the "Limit" as internal blocks. It can then be added to a loop, a function or other statements that needs logic.

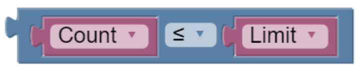
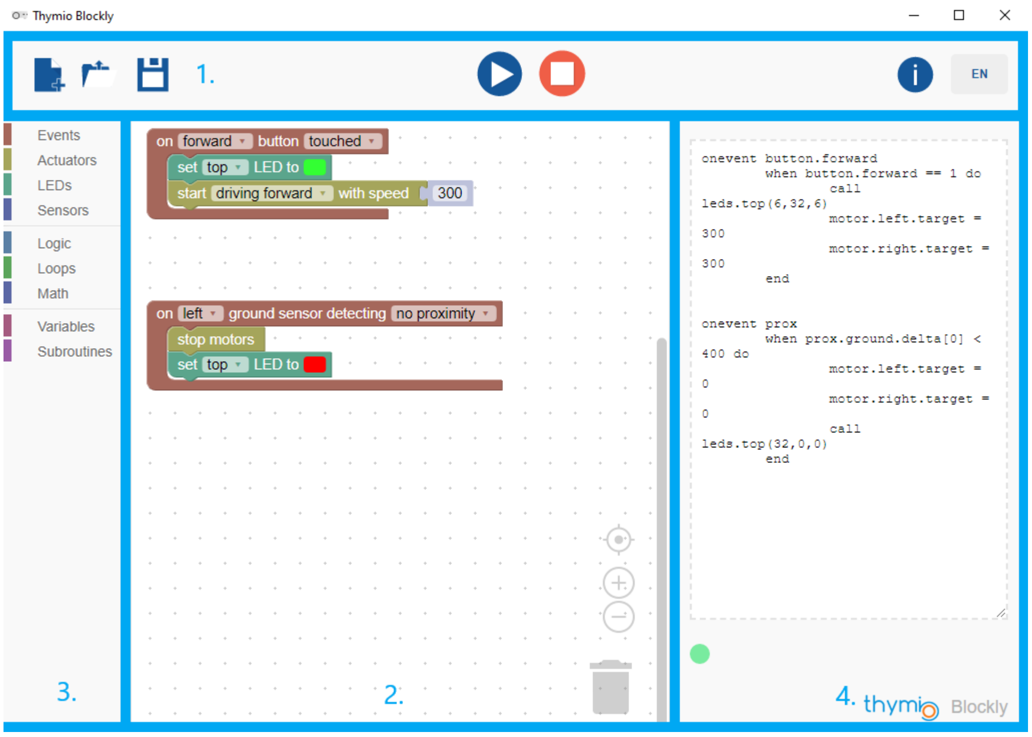


Figure 10: Logic block

Added to the base that Blockly is for Blockly4Thymio, is a compilator that interpret and adapt the Blockly code directly into Aseba language, and an Aseba Framework. Let us once again follow the steps described in the "How does it works" section in order to start blockly-ing a little program with Blockly4Thymio. Note that it is possible to open the Thymio Blockly environment without going through the Thymio suite, and without any Thymio II connected (physically or simulated). To do so open the location of Thymio, the downloaded not the installed, and select thymio_blockly, and then index. The environment window that opens after choosing the Blockly option is split into four parts.

1. A tool bar
2. A programming window
3. The category of blocks
4. The program translated into AESL

The following figure demonstrates a simple program, once run the program listens to two different events. When the center button is pressed and when the front middle proximity sensor detects a wall. The first one will activate the two motors at the same speed, as to drive forward, and light the top LED to green. Whereas the second will stop the motors and turn the LED to red.



```
onevent button.forward
        when button.forward == 1 do
                call
leds.top(6,32,6)
                motor.left.target =
300
                motor.right.target =
300
        end


onevent prox
        when prox.ground.delta[0] <
400 do
                motor.left.target =
0
                motor.right.target =
0
                call
leds.top(32,0,0)
        end
```

Figure 11: Thymio Blockly window

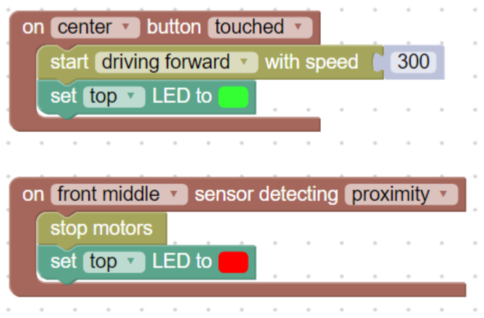
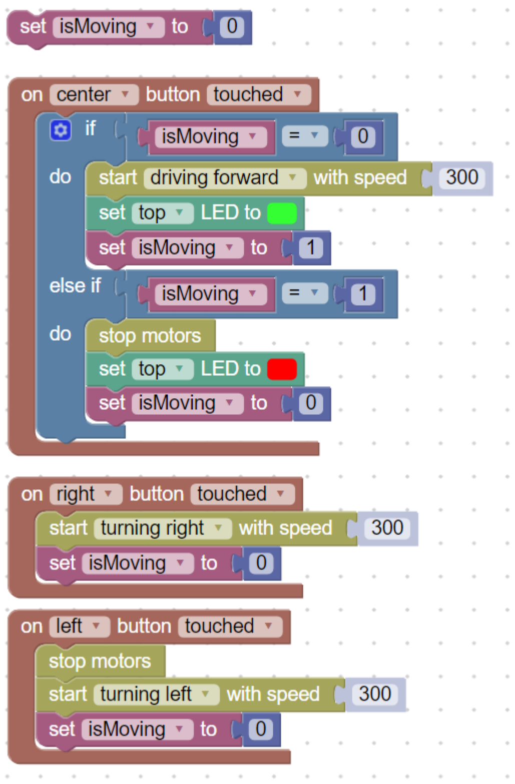

on center button touched
start driving forward with speed 300
set top LED to
on front middle sensor detecting proximity
stop motors
set top LED to

Figure 12: Basic program

Here we set a variable to act as a control if Thymio is moving or not. We then use this information into a test when we click the middle button, and we either move forward or stop according to the result. We added two other events for the right and left buttons that are responsible to turn the robot.



```
set isMoving to 0

on center button touched
  if isMoving = 0
  do start driving forward with speed 300
     set top LED to 🟩
     set isMoving to 1
  else if isMoving = 1
  do stop motors
     set top LED to 🟥
     set isMoving to 0

on right button touched
  start turning right with speed 300
  set isMoving to 0

on left button touched
  stop motors
  start turning left with speed 300
  set isMoving to 0
```

Figure 13: More complex program

3.3.3 Aseba

3.3.4 Scratch

3.4 What already exist

WeBots / Aseba logiciel

4 Three JS

5 Architecture

Domain class diagrams, Domain model, Package diagram, Sequence diagram, System sequence diagram

Write about the mvc, the interpreter pattern (?if used), problems because the three languages write files differently. All with the .aesl extension but not the same content format.

6 Usefull latex commands (to be deleted later on)

For that we read the article book of **Jerald:2015:VBH:2792790** which is very interesting, much more than the article **Diniz:2017:UGO:3100317.3100324**

7 Conclusion and future work

8 Appendices

8.1 Requirements Documentation

8.1.1 Vision

8.1.2 Objectives

8.1.3 System Boundaries

8.1.4 Requirements

8.1.5 Risk Analysis

Hack via .aesl as we execute the code if some code with malicious is feeded we still execute it. What happens if part of the project aren't implemented ? How to bypass blockade? Security?

8.1.6 User Stories

Customer User Stories

8.1.7 Use Cases Model

Use case diagram

8.2 Product Backlog

8.3 Sprint Backlog

8.3.1 First Sprint

2019-09-16 until 2019-11-15
Domain class diagrams, Domain model, Package diagram, Sequence diagram, System sequence diagram

8.3.2 Second Sprint

2019-11-15 until 2019-12-16

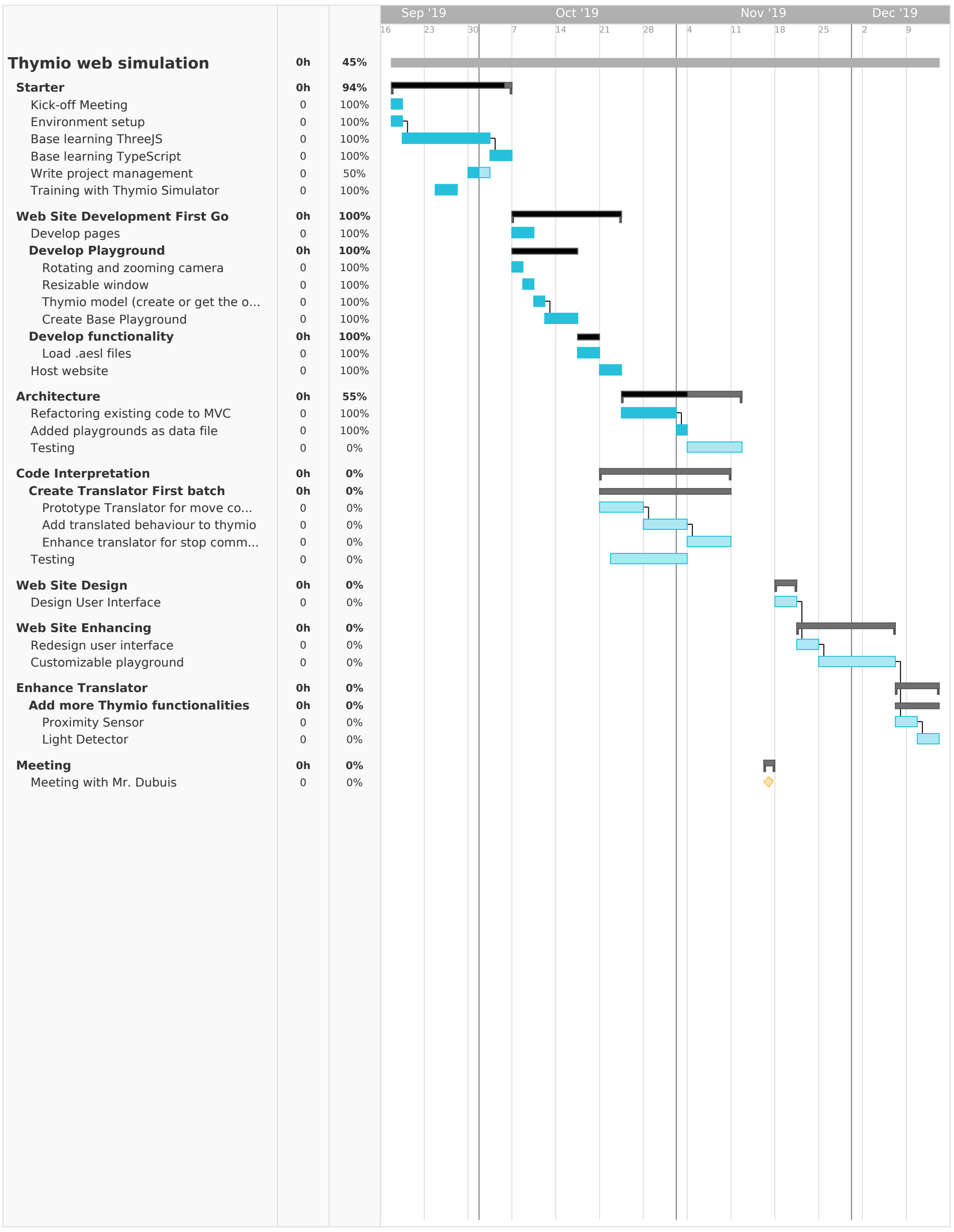
8.3.3 Third Sprint

2019-12-16 until 2020-01-06

8.3.4 Fourth Sprint

2020-01-06 until 2020-02-?

8.4 Gantt Diagram



Thymio web simulation	**0h**	**45%**
Starter	**0h**	**94%**
Kick-off Meeting	0	100%
Environment setup	0	100%
Base learning ThreeJS	0	100%
Base learning TypeScript	0	100%
Write project management	0	50%
Training with Thymio Simulator	0	100%
Web Site Development First Go	**0h**	**100%**
Develop pages	0	100%
Develop Playground	**0h**	**100%**
Rotating and zooming camera	0	100%
Resizable window	0	100%
Thymio model (create or get the o...	0	100%
Create Base Playground	0	100%
Develop functionality	**0h**	**100%**
Load .aesl files	0	100%
Host website	0	100%
Architecture	**0h**	**55%**
Refactoring existing code to MVC	0	100%
Added playgrounds as data file	0	100%
Testing	0	0%
Code Interpretation	**0h**	**0%**
Create Translator First batch	**0h**	**0%**
Prototype Translator for move co...	0	0%
Add translated behaviour to thymio	0	0%
Enhance translator for stop comm...	0	0%
Testing	0	0%
Web Site Design	**0h**	**0%**
Design User Interface	0	0%
Web Site Enhancing	**0h**	**0%**
Redesign user interface	0	0%
Customizable playground	0	0%
Enhance Translator	**0h**	**0%**
Add more Thymio functionalities	**0h**	**0%**
Proximity Sensor	0	0%
Light Detector	0	0%
Meeting	**0h**	**0%**
Meeting with Mr. Dubuis	0	0%

ID	Story Name	Story / Task Description
1	Create Documentation	Develop and write the documentation
2	Set up the Environment	Setting up and configuring the development environment
3	Basic Learning	Learning and training of the different technology used later on
4	Develop Playgrounds	Create playgrounds and function to generate meshes
5	Architecture Implementation	Refactor the existing code into the designed architecture
6	Web Deployement	Deploy the application on a webserver
7	Basic UI	Implement a basic UI
8	Behaviour Pipeline	Create pipeline to take .aesl file and translate/compile it into behav
9	Phyisics Implementation	Implementation of Collisions for ThreeJS Meshes
10	Enhanced UI	Enhancement of the current UI
11	Customizable Playgrounds	Implementation of a playground creator for users
12	Update Documentation	Update existing documentation and elaborate use cases and diagra
13	Enhanced Behaviour Pipeline	Implement more sensors, action and event for Thymio II
14	Update Documentation	Update existing documentation
15	Finish Documentation	Finish documentation

8.5 Version control

8.6 Configuration

User information On demand.

Access the Windows Virtual Machine

	Linux WM -ssh	Windows VM - rdp
Windows	Putty	Remote Desktop Connection
Linux	terminal	Remmina
MacOS	terminal	Microsoft Remote Desktop

See link bellow for more information and links. (It requires to be inside the bfh networkd to access it) `https://intranet.bfh.ch/TI/fr/Studium/Bachelor/Informatik/Tools/VMsHowto/Pages/default.aspx?k=vm`

First Configuration of IIS manager

We followed the steps in this tutorial video in order to configure IIS. `https://www.youtube.com/watch?v=rPRLe7QeVHM`

Then we had to open the port in order to access it from within the LAN.

1. Open the windows Firewall, click on Inbound Rules and New Rule. This will open the New Inbound Rule Wizard.

2. Select the desired type, Port, click next.

3. Choose TCP and specify the port used, here 80, click next.

ID	Story Name	Story / Task Description
1	Create Documentation	Develop and write the documentation
1.1	Template and Content	Choose a latex template and modify the content structure
1.2	About Thymio	What is thymio and how does it work.
1.3	Four supported languages	Descibe and initiate to VPL, Blockly, Aseba and Scratch
1.4	Backlogs	Create the Sprint and Project backlog
1.5	Architecture	Create DCD, DM, PD, SD, SSD and proposition of architecture
1.6	User Stories	Formulate the User Stories
1.7	Risk Analysis	Create risk analysis
2	Set up the Environment	Setting up and configuring the development environment
2.1	GitHub	Create the project in GitHub and the Git environment
2.2	Tools	Install Thymio Suite, NodeJS and download ThreeJS
3	Basic Learning	Learning and training of the different technology used later on
3.1	ThreeJS	Read documentation and examples, and practice
3.2	JavaScript	Update and deepen knowledge
3.3	Thymio languages	Learning and using VPL, Blockly, Aseba and Scratch
4	Develop Playgrounds	Create playgrounds and function to generate meshes
4.1	Two Default Playgrounds	Generating two defalut playground to be choosen for the simulator
4.2	Thymio Model	Create or load Thymio model
4.3	Mesh Generation	Create function to generate meshes for the playgrounds
5	Architecture Implementation	Refactor the existing code into the designed architecture
5.1	Refactor Code	Refactor existing code into MVC Pattern
5.3	Unit Testing	Write the JavaScript tests
5.4	JSDoc	Write the JavaScriptDoc
6	Web Deployement	Deploy the application on a webserver
6.1	Virtual Machine Setup	Set up the Virtual machine
6.2	WebServer	Create WebServer and publish it on bfh network
7	Basic UI	Implement a basic UI
7	Pages UI	Create three pages UI, one for each of the following index, simulati

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]	Actual Eff
			Total			

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]
1	Create Documentation		High	8	4
			Total		

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]
1	Create Documentation		High	8	4
			Total		

4. Select Allow connection, click next.

5. Select all three profile options, click next.

6. Add a Name and a description to this rule, click finish.

With this specified the website is now accessible from within the LAN at the following adress : http://147.87.116.44/Code/HTML

Additional setup

It was needed to create a web.config file and add a few file extension so that the .mtl and .obj would still be able to load. Otherwise we encountered an error of the type "Failed to load resource: the server responded with a status of 404 (Not Found)." The text that needed to be added to the web.config file is the following :

```
<?xml version="1.0" encoding="UTF-8"?>
  <configuration>
    <system.webServer>
          <staticContent>
            <remove fileExtension=".mtl" />
            <mimeMap fileExtension=".mtl" mimeType="text/plain" />
            <mimeMap fileExtension=".obj" mimeType="application/octet-stream" /
          </staticContent>
    </system.webServer>
</configuration>
```

8.7 Meetings

Date	Content
17.09.2019	**Kick Off meeting** - Documentation/Management - Technology to use : ThreeJS and Typescript - Setting up the goals
24.09.2019	**Second meeting** - Documentation language : English - Thymio model - Base talk about riks management
08.10.2019	**Third meeting** Workplace - Discussion on the choice of Windows as the Virtual Machine - Create a configuration file with the information of the VM - And an architecture proposal
15.10.2019	**Fourth meeting** - Which shapes and meshes should the user be able to create for his own custom playground - Problems with webserver, has to be accessible from outside the vm, so maybe switching from windosw to linux - Talk about the problem of thymio suite, that is the software allows the user to create programs only if a physical or a simulated one is plugged in
25.10.2019	**Fifth meeting** - Discussed using a Finite state machine to handle the events, but it may be too rigid so a non-deterministic finite state machine was the possible solution we came with - First little talk about the meeting with the expert, report

List of Figures

List of Tables

9 Problems encountered

- javascript not refreshing properly due to cache -> disable cache
- 3d Model not loaded on the webserver -> first tried to change the directory, then mixed two solution. Had to create a web.config file and add file extension for .mtl and .obj. https://stackoverflow.com/questions/41245938/web-server-cannot-find-mtl-file https://stackoverflow.com/questions/16097580/three-js-loading-obj-error-in-azure-but-not-locally
- shadow not rendering on plane of both playgrounds
- javascript file not found on server, net::ERR_ABORTED 404 (Not Found) => first solution (working partially) was to add a IIS_IUSRS.
- Thymio Blockly has trouble loading saved files. Using the software I wasn't able to load any .aesl file previously created with it, but I could load them if I used the index.html one.