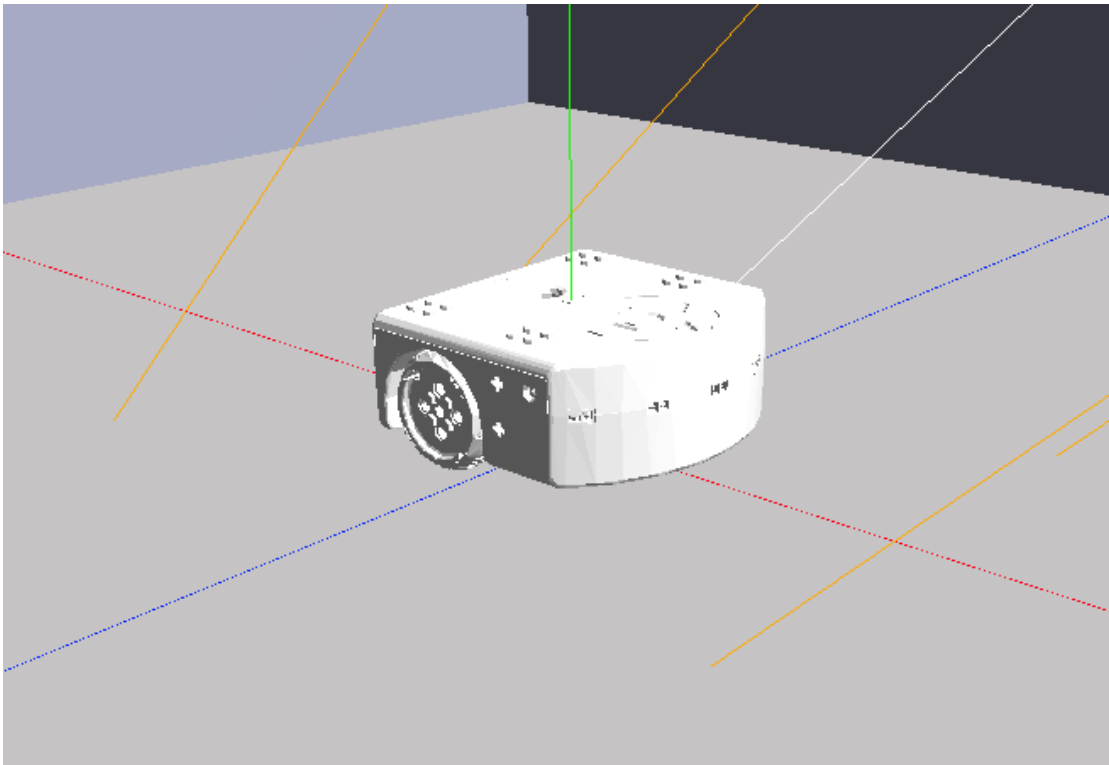


# Web Simulation of a Thymio Robot

Quentin Flückiger (flucq1@bfh.ch)

January 9, 2020





## Erklärung der Diplomandinnen und Diplomanden *Déclaration des diplômé-e-s*

### Selbständige Arbeit / *Travail autonome*

Ich bestätige mit meiner Unterschrift, dass ich meine vorliegende Bachelor-Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von mir stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet.

*Par ma signature, je confirme avoir effectué ma présente thèse de bachelor de manière autonome. Toutes les sources d'information (littérature spécialisée, discussions avec spécialistes etc.) et autres ressources qui m'ont fortement aidé-e dans mon travail sont intégralement mentionnées dans l'annexe de ma thèse. Tous les contenus non rédigés par mes soins sont dûment référencés avec indication précise de leur provenance.*

Name/Nom, Vorname/Prénom

Flückiger Quentin

Datum/Date

19.12.2019

Unterschrift/Signature

Dieses Formular ist dem Bericht zur Bachelor-Thesis beizulegen.  
*Ce formulaire doit être joint au rapport de la thèse de bachelor.*

## Management Summary

With all the growth that Information Technology has taken on in everyday life during the last decade, many questions have arisen. A very important one, at the very least in our minds, consists of the whole educational side. At what point should Information Technology be taught, in mandatory school, high school, higher vocational, self-taught? In Switzerland, we are starting to introduce some optional courses in this field already in primary school. Of course, they are not taught too deep and complicated concepts, it is more the way of reasoning and improving logical concepts. To help in this matter some school developed projects to ease the entry into the world of Information Technology, such as the Swiss Federal Institute of Technology of Lausanne did with the project know as Thymio. Thymio is a little robot that can be programmed on a computer with four different languages, with varying degrees of difficulty. Thus, it allows very young persons to experience and learn some of the basics. Unfortunately, this robot can have a few disadvantages, for example, the space required or the setup of its environment. To fill these gaps, we have decided to create a simulation of the Thymio. Thus, the objective for this project became the creation of a Thymio simulator, it had to be a web application to ease the accessibility, no software installation required, and be compatible with modern browsers.

We started by learning about the four different languages supported by the robot, viz., VPL, Blockly, Aseba, and Scratch. How to create a simple program with each of them and how the output programs were formatted. As we decided to make the simulator completely web, we needed a way to represent and animate computer graphic elements, thus we chose `three.js` as a library to help us in those matters.

Afterward, we created the base of the application, which consist of loading a Thymio model, having multiple basic playgrounds. We built on top of this a special playground creator tool, that allows the user to create its own playground with different meshes that he can place. The data are then stored inside a `JSON` file on his machine and can be load in the simulator. The first actuator, an actuator is a component of a machine that is responsible for moving and controlling a mechanism or system, to be added were the motors. We tried to make them behave as close to reality as possible, so two motors that can only do three things: Move forward, move backward, or don't move at all.

And last but not least, we had to develop a compiler that would translate a program written for Thymio in one of the four languages. To do this we used the code of the Thymio Suite application developed by **Aseba**. As their application was coded in `C++` it was needed to understand it, or at least understand the part important for this, and translate them into `JavaScript`.



# Contents

<b>1. Environment</b>	<b>7</b>
1.1. Three JS . . . . .	7
<b>2. Thymio</b>	<b>9</b>
2.1. What is Thymio . . . . .	9
2.2. How does it works . . . . .	10
<b>3. Requirements Documentation</b>	<b>13</b>
3.1. Vision . . . . .	13
3.2. Goals . . . . .	13
3.3. System Context . . . . .	13
3.4. Risk Analysis . . . . .	13
3.5. Stakeholder Descriptions . . . . .	14
3.6. User Stories . . . . .	14
3.7. Use Cases Model . . . . .	15
<b>4. Architecture</b>	<b>19</b>
<b>5. What already exist</b>	<b>23</b>
<b>6. Our Approach</b>	<b>25</b>
6.1. Base Development . . . . .	25
6.2. WebServer . . . . .	25
6.3. Playgrounds . . . . .	26
6.4. Interpreter . . . . .	27
6.4.1. Tokenize . . . . .	29
6.4.2. Parser . . . . .	30
6.5. Physics . . . . .	32
6.6. Sensor and Actuator . . . . .	33
6.7. Customize playgrounds . . . . .	37
6.8. Incorporated Behavior . . . . .	39
<b>7. Conclusion</b>	<b>41</b>
7.1. Results . . . . .	41
7.2. Future research . . . . .	41
7.3. Personal conclusion . . . . .	41

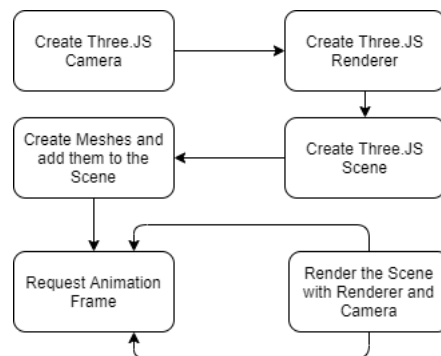
## Contents

<b>A. The different programming languages</b>	<b>43</b>
A.1. VPL . . . . .	43
A.2. Blockly 4 Thymio . . . . .	47
A.3. Aseba . . . . .	50
A.4. Scratch . . . . .	50
<b>B. Product Backlog</b>	<b>51</b>
<b>C. Sprint Backlog</b>	<b>53</b>
C.1. First Sprint . . . . .	53
C.2. Second Sprint . . . . .	53
C.3. Third Sprint . . . . .	54
C.4. Fourth Sprint . . . . .	55
C.5. Fifth Sprint . . . . .	56
C.6. Sixth Sprint . . . . .	56
<b>D. Gantt Diagram</b>	<b>59</b>
<b>E. Version control</b>	<b>63</b>
<b>F. Configuration</b>	<b>65</b>
<b>G. Meetings</b>	<b>67</b>
<b>H. Problems encountered</b>	<b>69</b>

# 1. Environment

## 1.1. Three JS

`three.js` is a 3D library for JavaScript which uses a default WebGL renderer. It allows the user to display, create and animate 3D computer graphics in a web browser. Its basic rendering pipeline is as follow :



Every `three.js` application is composed of at least one **Camera** element, a **Renderer** element and a **Scene** element. The different meshes are added to the scene and this scene along with the camera are rendered using the method `requestAnimationFrame` from `three.js` on the renderer element. Using the `three.js` library it is possible as well to create Augmented Reality application working in web browser, which is very neat. For example it can display information from a database based on a marker. If this topic picked your interest, we would recomend you the presentation, examples and discussion from Jerome Etienne.





## 2. Thymio

### 2.1. 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.

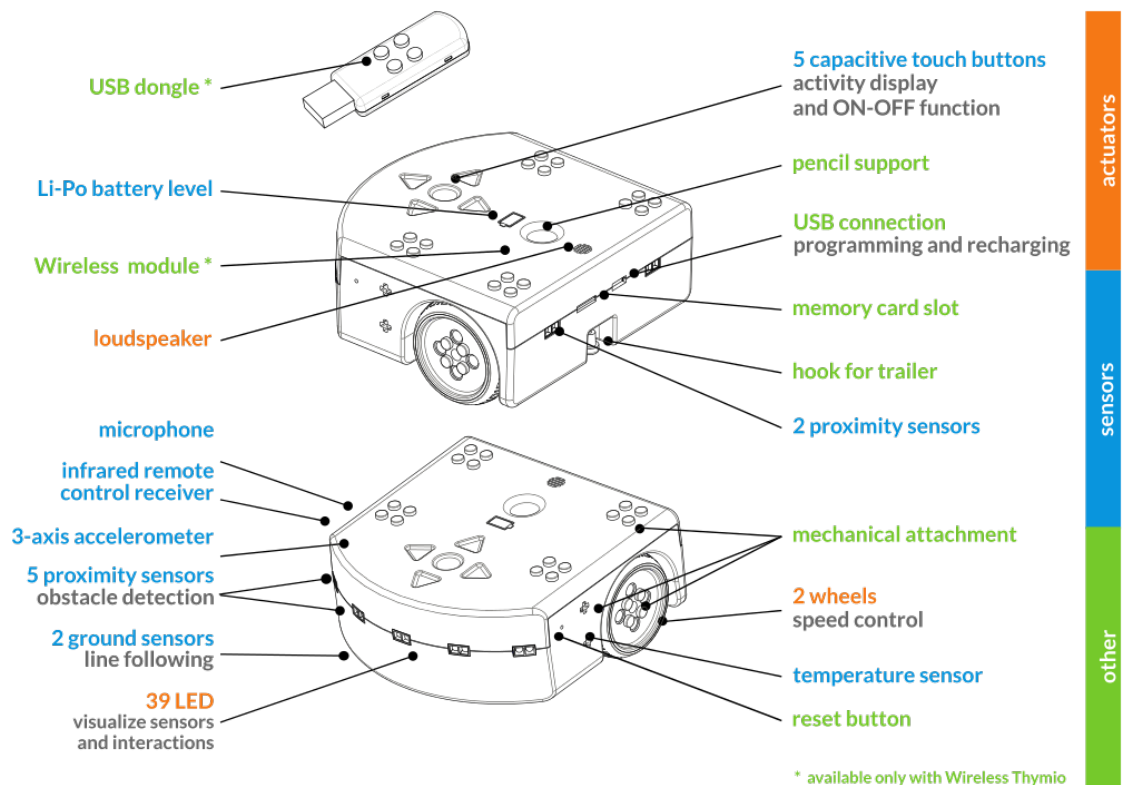
The Thymio project is based on a collaboration between the MOBOTS group from the Swiss Federal Institute of Technology in Lausanne (EPFL) and the Lausanne Arts School (ECAL). MOBOTS being the Miniature Mobile Robots Group, they are mainly focused around system design for small robots of the kind. It 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 behaviors. It was used as a user study to gather feedback from clients to know what features needed to be implemented on the Thymio II.



## 2. Thymio

From left to right, "Monsieur Patate", Thymio, 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.



Thymio II sensor and actuator

### 2.2. 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 plugged in via USB cable or USB dongle (possible only if it is the Wireless Thymio) and

## 2.2. *How does it works*

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 behavior. A more detailed section on the four different programming languages can be found in the section A at the page 43.



## 3. Requirements Documentation

### 3.1. Vision

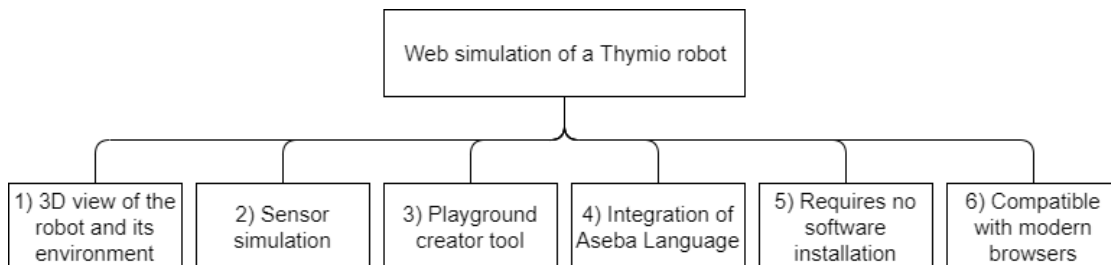
A start-up from the EPFL has developed a robot, the Thymio robot or Thymio II, that promotes the programming and robotic activities among children. To feed program to the robot, a software has been developed, it integrates the four following programming languages:

- VPL
- Blockly4Thymio
- Aseba
- Scratch

The project Web Simulation of a Thymio robot is aimed to create a simulator for the Thymio II so as to allow people to see their programmed behavior directly.

### 3.2. Goals

This aim has been split into 6 different defining goals in order to create this application.



### 3.3. System Context

### 3.4. Risk Analysis

In order to carry out the project successfully, we must consider the following possible complications. The possible complications are on one hand assessed according to their impact and on the other hand according to their likelihood to occur. Thus, we obtain a predictable risk factor that allows us to have an overall view and take preventive measures if necessary.

### 3. Requirements Documentation

	1	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
	0,9	0,09	0,18	0,27	0,36	0,45	0,54	0,63	0,72	0,81	0,9
	0,8	0,08	0,16	0,24	0,32	0,4	0,48	0,56	0,64	0,72	0,8
Likelihood (A)	0,7	0,07	0,14	0,21	0,28	0,35	0,42	0,49	0,56	0,63	0,7
	0,6	0,06	0,12	0,18	0,24	0,3	0,36	0,42	0,48	0,54	0,6
	0,5	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5
	0,4	0,04	0,08	0,12	0,16	0,2	0,24	0,28	0,32	0,36	0,4
	0,3	0,03	0,06	0,09	0,12	0,15	0,18	0,21	0,24	0,27	0,3
	0,2	0,02	0,04	0,06	0,08	0,1	0,12	0,14	0,16	0,18	0,2
	0,1	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,09	0,1
		0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
							Impact (B)				

Risk Matrix

Event	Likelihood (A)	Impact (B)	Risk Factor (A*B)
Financial issues	0	1	0
Collisions not implemented	0.3	0.6	0.18
Behavior pipeline not working	0.4	0.9	0.36
Playground creator not working	0.4	0.6	0.24

### 3.5. Stakeholder Descriptions

**Product Owner** Flückiger Quentin flucq1@bfh.ch

*Interests:*

- The product owner wants to satisfy the customer.

**Development Team** Flückiger Quentin flucq1@bfh.ch

*Interests:*

- The development team wants to develop a usefull application for the customer.

### 3.6. User Stories

**Users User Stories**

As a user, I want to upload an .aesi file, so that I can witness the simulated behavior.

Description:

The user wants to upload an .aesi file to see the programmed behavior simulated. Success:

- The simulation works.

Failure:

- The .aesi file doesn't contain a program.
- The .aesi file contains behavior not included in the simulator.

**As** a user, I want to create a simple testing environment, so that I can diversify the experiences.

Description:

The user wants to create home made playground with a simple playground creation tool.

Success:

- The playground was successfully created and saved.

Failure:

- The created playground isn't saved properly.
- The user encounters trouble while creating the playground, be it meshes creation or placement.

**As** a user, I want to use the application without having to install anything, so that the application can be accessed easily.

Description:

The user wants to access and use the application without installing anything. Success:

- The use can start the application directly in his browser.

Failure:

- The webserver isn't accessible.

## 3.7. Use Cases Model

**Use Case:** Access the application

**Primary Actor:** User

**Stakeholders and Interests:** User: Wants to access the application through a web browser.

**Preconditions:** User has access to the bfh network.

**Success Guarantee (Postconditions):** The user can access the application via a modern web browser.

**Main Success Scenario:**

### 3. Requirements Documentation

1. User start web browser.
2. User navigate to website address.

**Extensions:**

1. a) No available internet connection.
2. a) Not logged in the bfh network.  
b) Web Server currently offline.

**Special Requirements:** Modern web browser compatibility.

**Technology and Data Variations List:** -

**Frequency of Occurrence:** Could be nearly continuous.

**Open Issues:** -

**Use Case:** Interpret .aesi file

**Primary Actor:** User

**Stakeholders and Interests:** User: Wants to load .aesi behavior file to be translated and simulated.

**Preconditions:** User has a .aesi file containing behavior code for Thymio.

**Success Guarantee (Postconditions):** File is correctly compiled, and simulation simulate expected behavior.

**Main Success Scenario:**

1. User access the website.
2. User input a .aesi file.
3. System control file integrity.
4. System compile file to JavaScript code that can be run as behavior.
5. System run given program.

**Extensions:**

2. a) File too large for application.
3. a) System signals error and reject file because not conform to awaited structure.
4. a) System signals error while compiling file.

**Special Requirements:** -

**Technology and Data Variations List:** -

**Frequency of Occurrence:** Very often.

**Open Issues:** -

**Use Case:** Change playground



**Primary Actor:** User

**Stakeholders and Interests:** User: Wants to change the rendered playground.

**Preconditions:** User has access to the bfh network.

**Success Guarantee (Postconditions):** The playground is changed accordingly the wishes of the user.

**Main Success Scenario:**

1. User access the website.
2. User chooses the wanted playground.
3. System load wanted playground.

**Extensions:**

2. a) The input file is not from the right file extension.
3. a) Fail to load playground because file not conform to awaited structure.  
b) Internal error when playground was loaded.

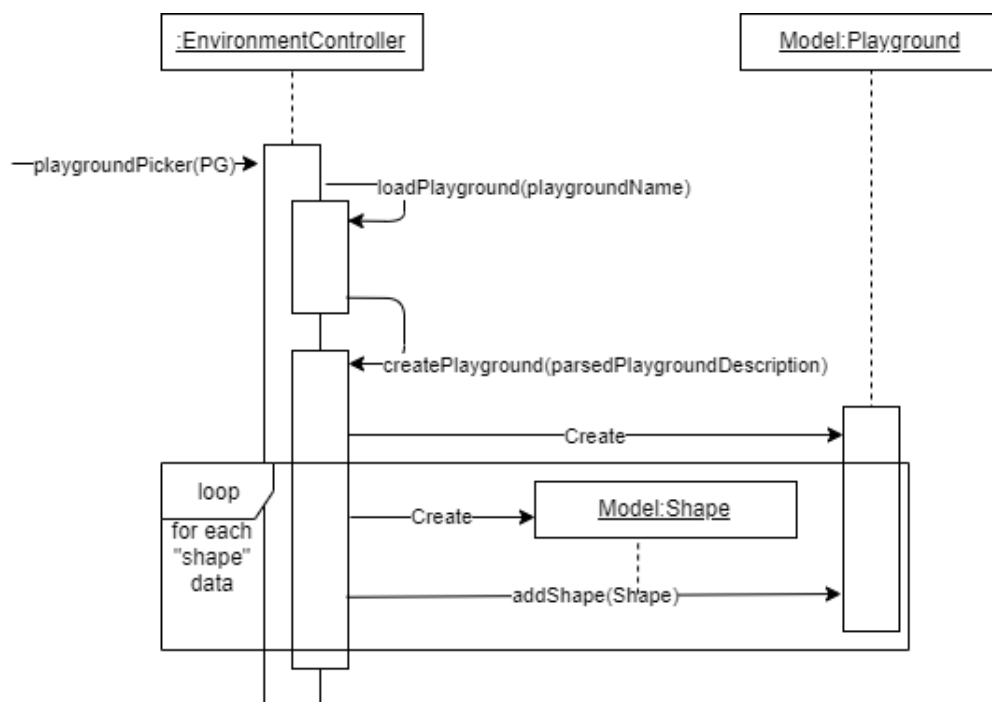
**Special Requirements:** -

**Technology and Data Variations List:** -

**Frequency of Occurrence:** Often.

**Open Issues:** -

**Sequence diagram:**

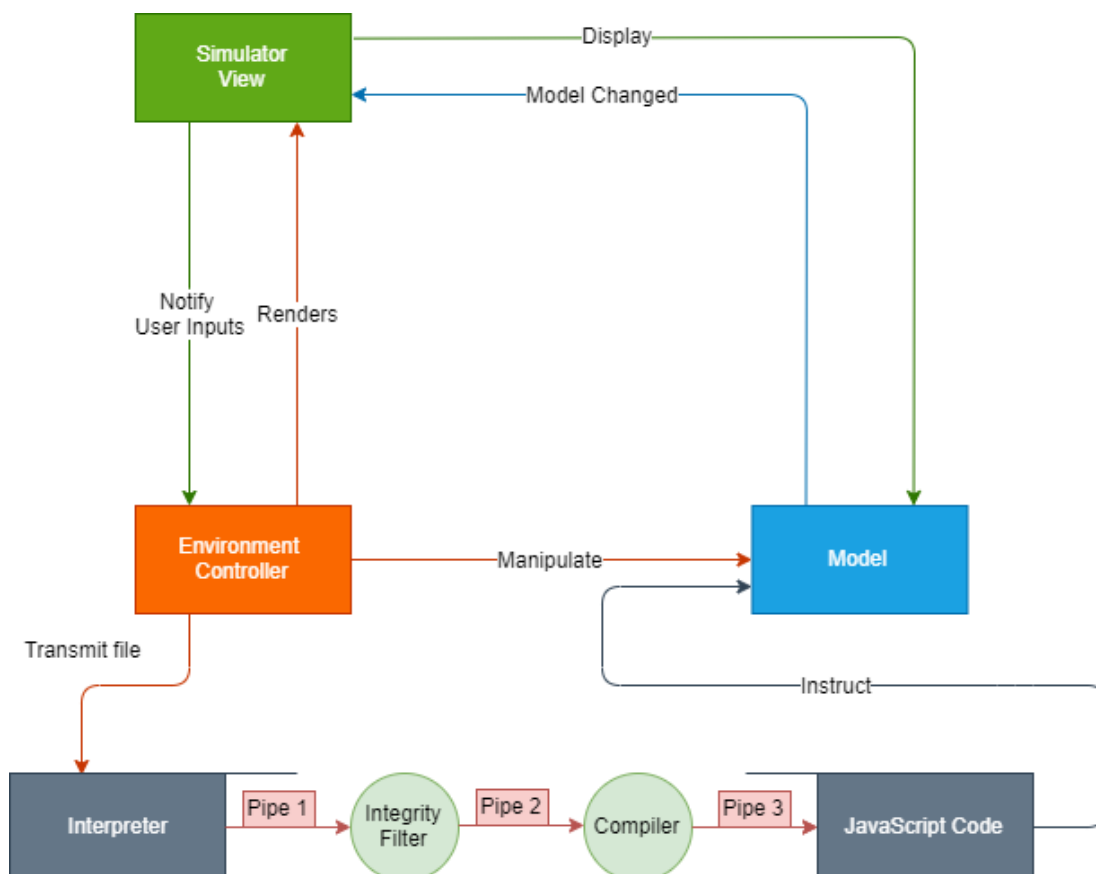




## 4. Architecture

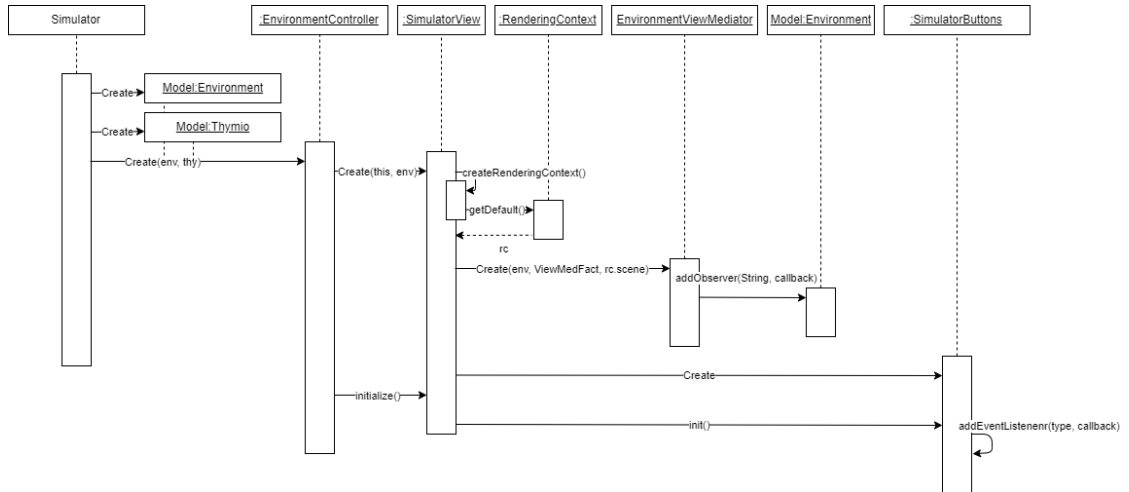
We decided to divide the architecture into two different parts. One for the simulation side and the other one for the customization, although they are very similar there are still some differences that push us to look at them with two different angles. Both are build based on a **Model View Controller** system where the elements of the playground, be it a wall or the Thymio robot, are models and the page is seen by the user is the view. This view registers user inputs and transmits them to the controller.

As said we have split the architecture, bellow is a graphic representing the architecture for the simulator. We can see the **Model View Controller** and a second component which is the interpreter, that is singular to the simulator page. Its role is to test the integrity of the file given as input through the **Controller** and compile it into **JavaScript** code for it to be used as behavior code for the Thymio model.

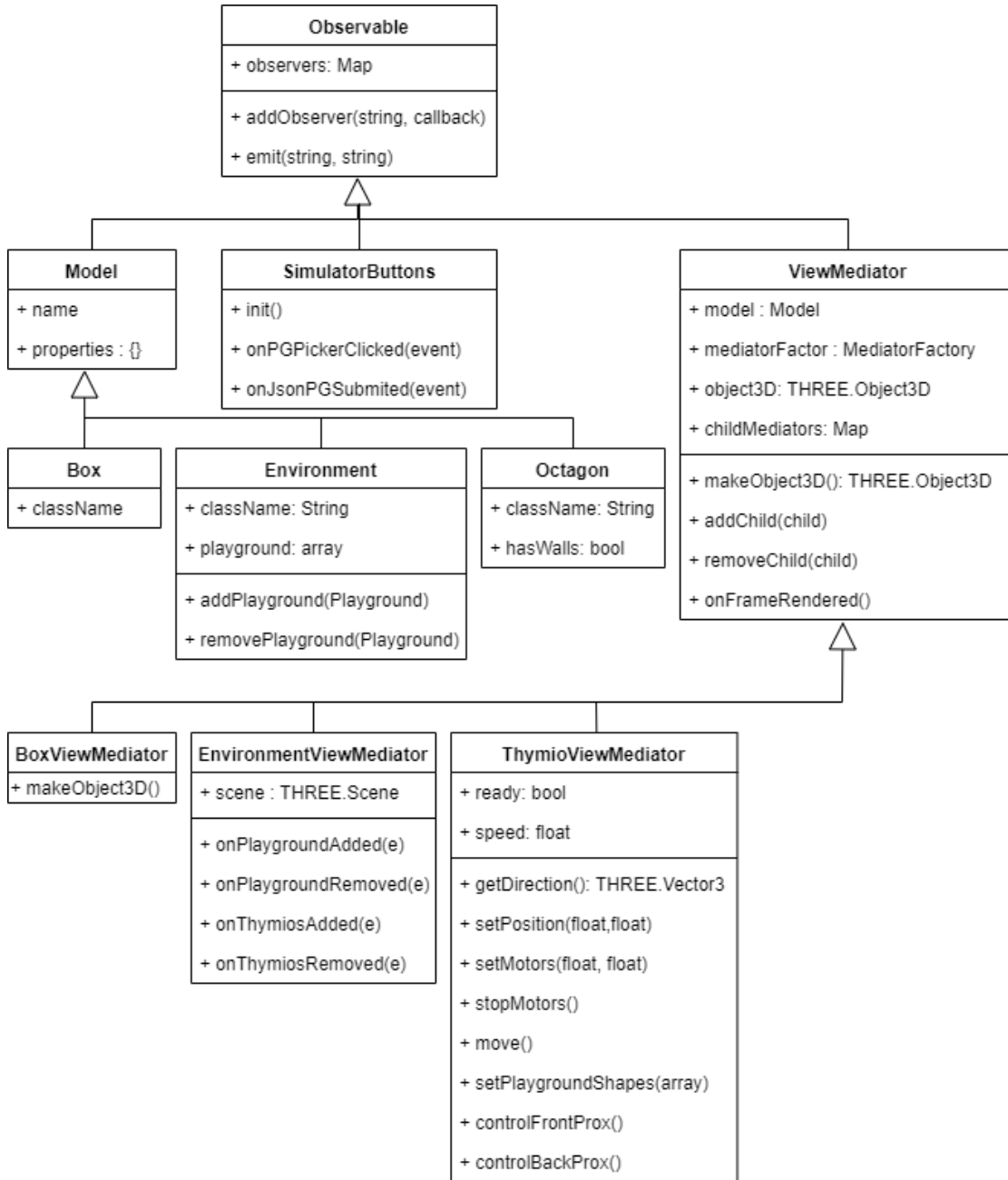


#### 4. Architecture

The MVC we are using is based on an example from Lucas Majerowicz. Once the application starts the following process is run in order to create the **View**, **Model**, and **Controller**.



The **Model** part is split into three categories that extends the same base class **Observable** which works as an event system with the `addObserver` and `emit` method. The first category of the three that we will cover is the one responsible for the buttons, we instantiate one of the right types depending on the **View**, at the start of the page and add `addEventListener` to the corresponding HTML elements. Those events will trigger the `emit` method of its base class to notify the **View** that this particular button has been clicked. The **View** will catch this event because of the second base method of the **Observable** class, forwarding it to the **EnvironmentController** class to take care of the logic. The two categories left are linked as they operate as the two sides of a coin. They are used for the `three.js` elements. The first one, the **Model**, holds the data of the object such as its name, a list of properties containing the dimension, color and other attributes for the object. It is those models that are added to the playground element, and whenever a shape is added to the playground, this one will call its base method `emit` in order to notify the **ViewMediator**. The **ViewMediator** is the last category and is responsible to create the 3D objects with the data from its model. It is as well responsible for the logic that we would apply to a model, such as the animation of the 3D object, and the supervising of deleting or adding models. The image below shows a class diagram of the **Model** part of the MVC, not all classes are represented for clarity purpose, as their configuration is very similar to one comprised in the diagram.

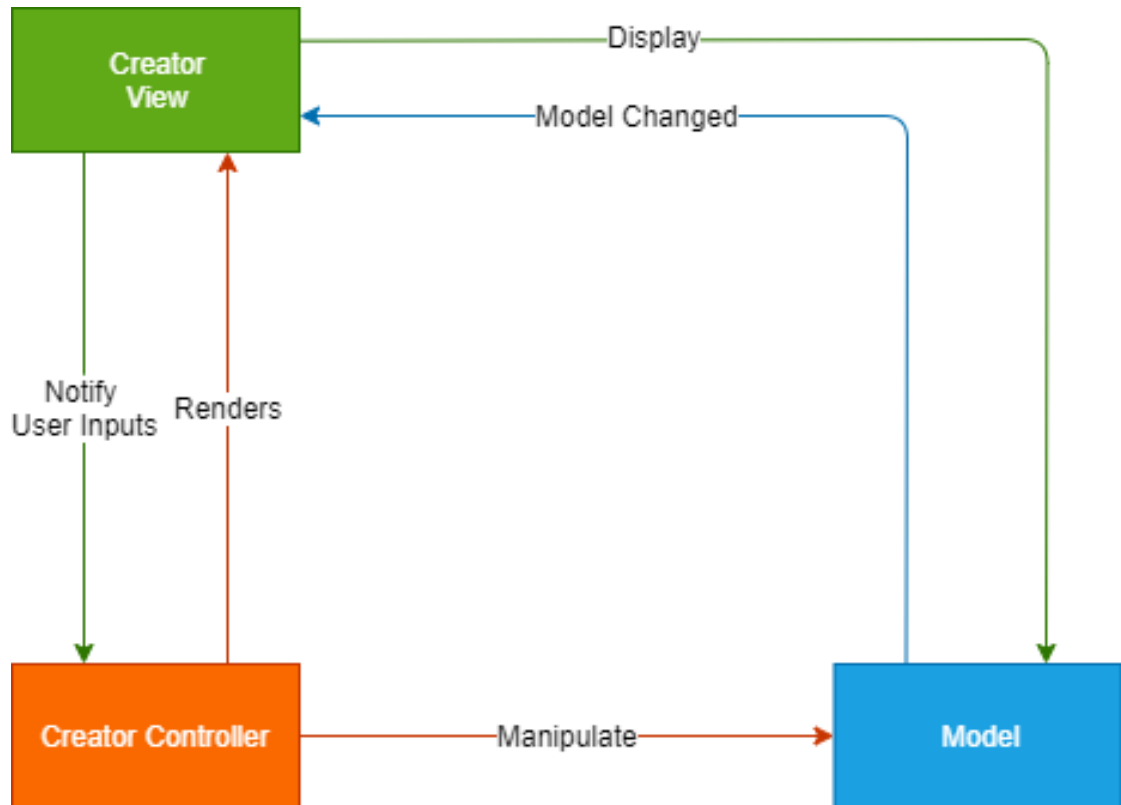


The **View** and the **Controller** elements are different in the two pages whereas the **Model** is not. The **Controller** in the simulation side loads the playgrounds be it built-in or coming from the device, and upload the aesl file to the begining of the filter/compiler. And in the customization side it is responsible for writing the **JSON** file with the data from the meshes of the scene, instantiating the meshes and registering them once positioned, or deleting them. The two **Views** are very similar, the differences are the observers added to its **CreatorButtons/SimulatorButtons**. This is how an observer is added.

#### 4. Architecture

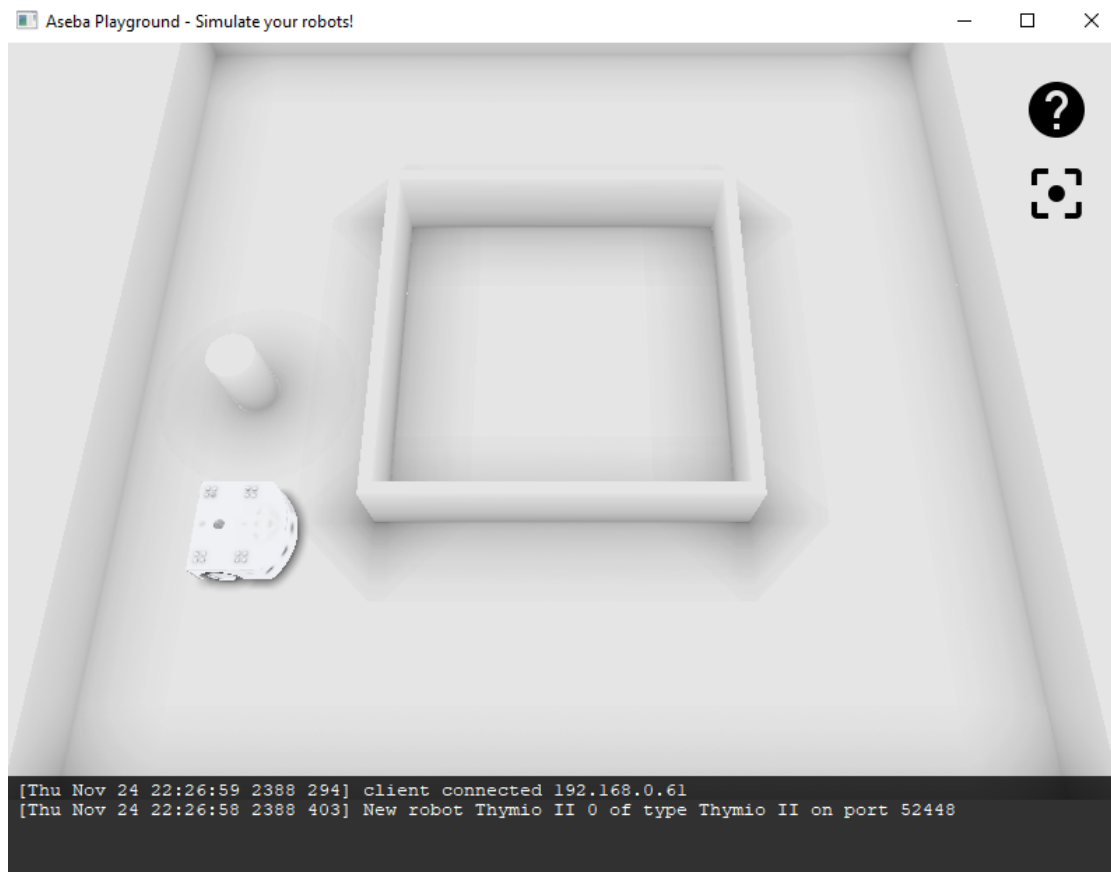
```
this.simulatorButtons.addObserver('jsonPGSubmitted',  
  (e) => this.controller.onJsonPGSubmitted(e));
```

And here is the second Model View Controller responsible for the playground customization, its architecture is the same as the other one except it doesn't implement an interpreter.



## 5. What already exist

There exist two possibilities to simulate the behavior of a Thymio II Robot on a computer. The first one is through the Thymio Suite application developed by the creator of Thymio. It is an application that regroups multiple features such as coding the robot in one of the four available languages, uploading the program to a real robot or simulating its behavior via a simulator developed in the programming language C++. This simulator allows one to load a playground among multiple pre-set and run the coded program for the robot.



The other option is to use WeBots, which is an open-source 3D robot simulator for industry, education and research purpose. It has been developed in 1996 at the Swiss Federal Institute of Technology in Lausanne since then it became a property license software of Cyberbotics in 1998, and in December 2018 was lastly released under the

## *5. What already exist*

free and open-source Apache 2 license. WeBots is a very powerful software that can do a lot of things, and one of these things is an accurate Thymio II model with almost all its sensors and actuator. The two Aseba Studio and VPL for Thymio can be directly connected to the software and its simulated robot. The specification and usage can be found on their website with the following link [WeBots Thymio](#) .



## 6. Our Approach

### 6.1. Base Development

The first steps were to learn how to use the `threejs` library and thus we integrated the needed code directly onto some `.html` file, which was not very efficient for a big application but was okay to test the functionalities. At that time, we looked for some useful methods that we would need later on, such as the way to resize the window and its content. Another functionality added at this period was the controls of the camera, that comes from the `OrbitControls.js` file and allows the creation of an `OrbitControls` object which let the user move the camera, rotate it and zoom with the mouse. Afterward, we implemented method to create different Shapes with more ease for a larger application. We then moved the scripting part outside of the `html` and inside different `JavaScript` file as it was not convenient to have all the code into one single `html` file.

### 6.2. WebServer

Regarding the goal number 5, "Require no software installation", we decided to use a webserver in order to access the application. At first we choose the IIS Manager that is built in with Windows 10, 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.
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.

#### **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

## 6. Our Approach

"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" />
        <mimeTypeMap fileExtension=".mtl" mimeType="text/plain" />
        <mimeTypeMap fileExtension=".obj"
          mimeType="application/octet-stream" />
      </staticContent>
    </system.webServer>
  </configuration>
```

Unfortunately, after implementing the new MVC architecture for our application we encountered an issue where the application wasn't able to locate some JavaScript file and giving the following error message in the console : **javascript file not found on server, net::ERR\_ABORTED 404 (Not Found)** . After a while of debugging and asking questions to persons who could know the issue, we decided to move away from ISS Manager as our knowledge of this software is too small and the time needed to acquire this knowledge wasn't worth the effort. The alternative solution we came up with was to use XAMPP, we managed to push the same version of the application without any problem and without having to do anything special. So we decided to go forth with XAMPP.

### 6.3. Playgrounds

We decided to create three different build-in playgrounds for the application. A basic, with four walls and a square plane. A borderless, composed of a track that leads out of the octagon plane. And one with multiple obstacle, walls and a track. Each one of them would be composed of one **Group** element that is filled with different meshes created from the previously implemented method found in the **GeometricalMeshes.js** file.

We decided to enhance the the amount of different meshes and thus added an algorithm to create tracks. There was two steps for the algorithm to create tracks, the first one was to compute a simple line between two points. But the result was too thin and therefore a better solution was found. The second iteration for this algorithm takes an array of points, those points are of type **Vector3** so as to register the three coordinates, and compute a new **Vector3** that holds the resulting vector position of the next point minus the current point. The x and z values of this vector are then used as the center to position a box object, which represent the track, and then it is aligned to this vector.

```
for (let i = 0; i < points.length-1; i++) {

  const trackWidth = new THREE.Vector3().copy(points[i+1]).sub(points[i]);
```

```

const track = new THREE.Mesh(
  new THREE.BoxGeometry(trackWidth.length(), TrackHeight, TrackDepth),
  material
)

track.position.x = points[i].x + trackWidth.x/2;
track.position.z = points[i].z + trackWidth.z/2;
track.quaternion.setFromUnitVectors(new THREE.Vector3(1, 0, 0),
  trackWidth.clone().normalize());
container.add(track);
}

```

We decided to load only once the Thymio model, as it takes in average 500ms to 1'000ms to load it, and we encountered some issues where the model would not load everytime when we would change the playgrounds. Therefore we reset it's position and rotation whenever we change the playground, or if a position is given in the playground.json data file, the Thymio is moved to the wanted position.

Thinking about the creation of playground we decided to change how the playgrounds data were recorded and instead of having them as JavaScript file, we moved them all into a JSON file. So later on it would not require more work to load a customized playground. To do so we open and read a JSON file of a given name, and then skim through the JSON file and create the Shapes accordingly of the data. Bellow an example for the boxes.

```

if (file.bboxes) {
  for (const boxRecord of file.bboxes) {
    var box = new Box(boxRecord.name, boxRecord.props);
    playground.addShape(box);
  }
}

```

Unfortunately we found later on that `threejs` has a built in function that translate a ThreeJS Object into a JSON file element, and inversely. But we are still using the solution we developed.

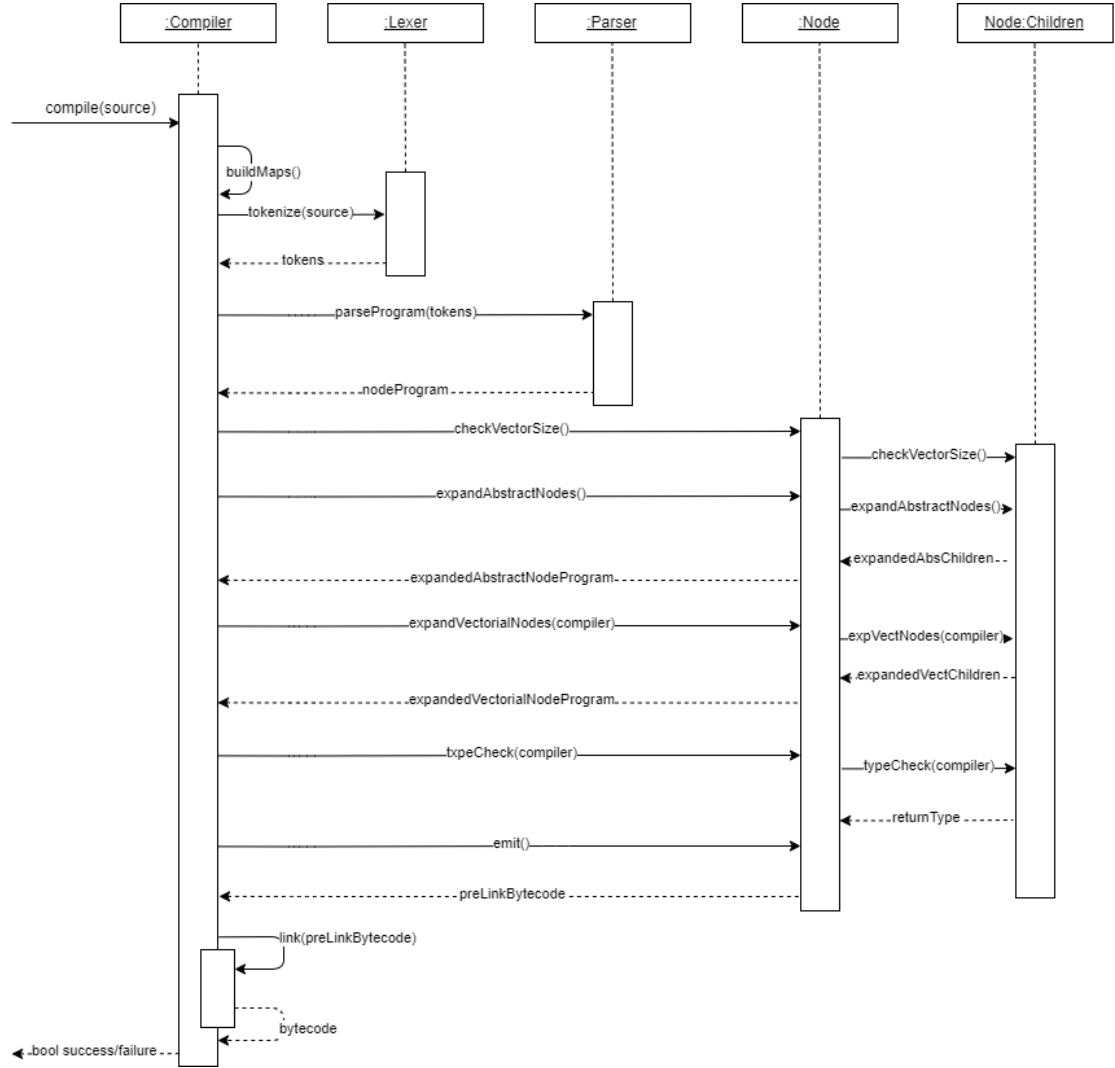
## 6.4. Interpreter

The interpreter found that compile/translate .aesi file into another programming language is the one used in the Thymio Suite application, its source code can be found in the following repository Aseba git. Unfortunately, it is a C++ one, so we had to understand it and find a way to either use it and then use a C++ to JavaScript interpreter, or translate the compiler from C++ to JavaScript by hand. We decided to translate the already existing one from Aseba and started by the compiler. After looking at his behavior we recognize how it was built and separated, then came the time to translate it to JavaScript. First thing first it creates multiple maps of variable, constant and events that need to be rebuild with each call to the compiler in case the previous ones

## 6. Our Approach

produced errors. Afterwhat comes the tokenization of the source file, which consist of creating Tokens with the position of the element, its type in the environment, and its value if provided. Then this tokenized source is parsed into Nodes, which are expanded and type checked. Once the program is checked, it is emitted as a first bytecode output and then this bytecode is linked creating the final program.

Here is a sequence diagram of the compilation of a file.



Translating the compiler and everything around was a very hard task for multiple reasons. The first one being the size of the application, it was very hard to understand which component was linked to which, in what manners and how they interact with each other. We needed to spend a tremendous amount of time scavenging the files and folders in order to try to get an understanding of the architecture. The second reason would be the lack of comments in the code, which added to the first point is not helping at all. And as we never expected at the beginning of the project to work with C++ we

were not ready for it. Our knowledge of this programming language is very basic. So if we add the two previous points, we have a full-scale application coded in a language we are not familiar with and with sparse comments. Thus making it a very difficult part for us, and one that we couldn't finish.

### 6.4.1. Tokenize

To tokenize the source file we skim through the document and switch depending on the value of the character, we have basically five categories for this switch. The first one are the tokens which require only to read one character, such as `)`, `,`, those can directly be given their type. Next are the comments, the comment block `/* ... */`, or the simple line comment `/*`. For the block comment, we run through the source in search of the `*/` character association that marks the end of the comment block, throwing an error if not found. The third category encompass the cases that require one character look-ahead, such as when we found the character `+` is it a simple `+` or `+=` or even `++` ? The fourth is almost the same but with two characters look-ahead, such as `<`. The fifth category, and the default case of this switch, is the one where the amount of look-ahead needed is not defined. In this category fit the numbers and the strings, which are found using a regex as replacement of the C++ method `is_utf8_alpha_num()`, which controls that the element is either a letter or a number. Bellow is the code for this regex.

```
isAlphaNumeric(ch) {
  return ch.match(/^[a-zA-Z0-9]+$/i) !== null;
}
```

We have to be wary of one point when using this method, and it is that if the `ch` parameter is not a string, then it will throw an error. Coming back to the switch we first test if the chain of character is a number by combining regex and looking for multiple characters. If it is not the case we check whether or not its value is the same as one of the given keywords, such as `when` or `const`. And if none of the keywords match it is labeled as a string literal and given the value of the chain of characters. Here we encountered a problem of language between C++ and JavaScript, as the type of the token is given with an `enum` in C++, and they don't exist in JavaScript so we had to find a workaround to this language incompatibility. We used the Object method `freeze()`, which prevents the modification of existing property attributes and the addition of new ones, and still had to give a value to each properties.

```
const type = Object.freeze(
{
  TOKEN_END_OF_STREAM : 0,
  TOKEN_STR_when : 1, /*"TOKEN_STR_when",
  TOKEN_STR_emit : 2, /*"TOKEN_STR_emit",
  ...
}
```

## 6. Our Approach

### 6.4.2. Parser

Once the source file has been tokenized with the Lexer, and no error occurred during the process, the **Compiler**, that has been partially translated from C++ to JavaScript, will pass the tokens as an argument to the parser which will parse it and create a program tree.

It will start by creating a new `Node.ProgramNode` object with the position of the first element of the tokens array. Then the algorithm will loop on the tokens while it is not empty, and call methods depending on the type of the token. At this stage, we split into three categories, constant, variable and the rest. In the case of constant, the algorithm will check the next token and control that it is of type string, otherwise the declaration of the constant isn't valid and an error is thrown. If the condition is met we assign the name of the constant to a named variable, and the position to a position variable. Afterward, it controls that no constant with this name already exists, then controls that the next token is of type `TOKEN_ASSIGN` to respect how a constant is declared, get the value that should be at the next token, if that one is not of type `TOKEN_INT_LITERAL` it will return the value 0. This part will be improved by the completion of the translation of the tree. And finally the constant with its name, position and value will be added to the `constantsMap` of the compiler.

If it is a variable, it will be discarded and the following tokens will be discarded as well if they are part of its declaration, we actually replaced the part where it would parse the variable by this removal system. We did so because the tree and the parser are not complete. Here is how it should be, if the tree and parser were finished.

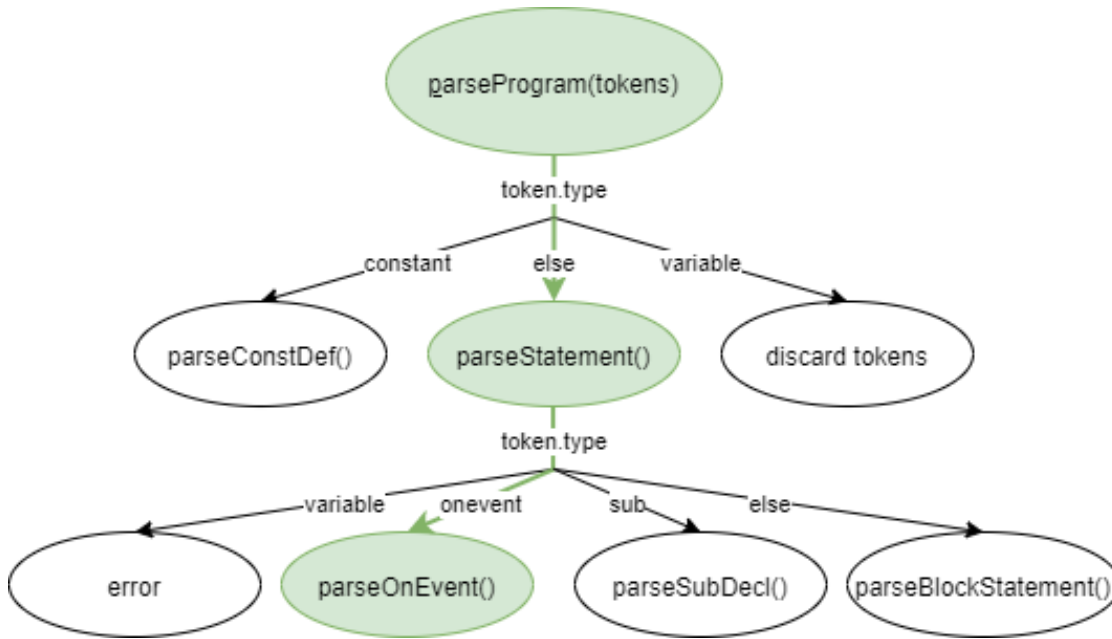
```
var child = this.parseVarDef();
if(child)
{
    programNode.children.addRear(child);
}
```

And this is how it actually is.

```
this.tokens.removeFront();
if(this.tokens.front().type === TT.type.TOKEN_STRING_LITERAL)
{
    this.tokens.removeFront();
    if(this.tokens.front().type === TT.type.TOKEN_STRING_LITERAL ||
        this.tokens.front().type === TT.type.TOKEN_INT_LITERAL)
        this.tokens.removeFront();
}
```

The third part covers the rest of the tokens and is the most complex among the three. The token will go through multiples method based on switch on his type until he reaches the parse method that defines it. Not every parse methods were translated from the C++ program as their amount is rather big and not easy to translate into JavaScript. Below

is a diagram that will show the process of a token of type `TOKEN_STR_onevent` to ease the understanding.



The end method `parseOnEvent()` will control two things, first that an event with this Id exists, the list of `eventId` can be found in the `Compiler.js` under the `buildMaps` method, and that such an event hasn't been implemented already. If the conditions are met a new `Node.EventDeclNode` with the position and the id of this event will be returned and added to the `programNode`. Or if the token is of type `TOKEN_STR_when` it will try to parse the condition and thus will start with the following `parseOr` method.

```

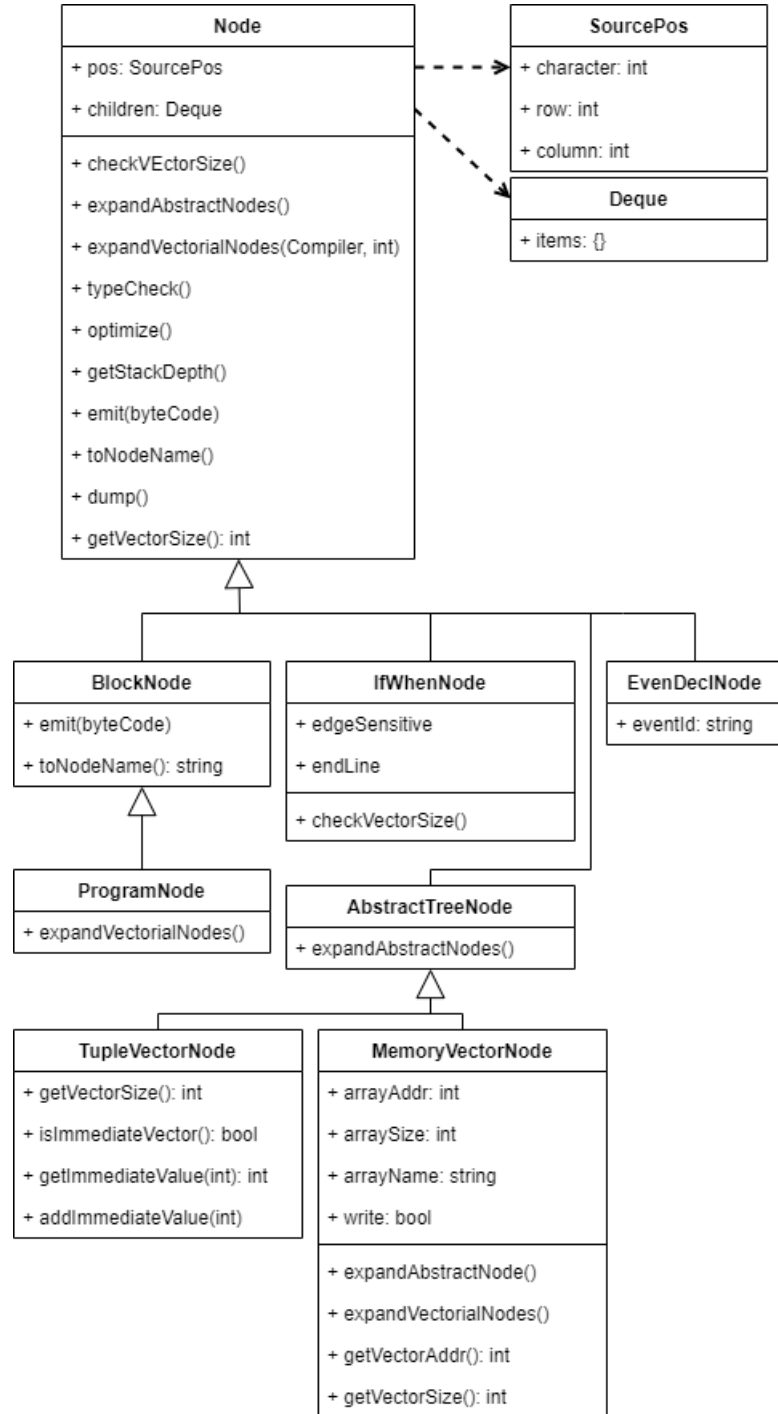
parseOr(){
  var node = this.parseAnd();
  while(this.tokens.front().type === TT.type.TOKEN_OP_OR)
  {
    var pos = new SourcePos();
    pos.setValues(this.tokens.front().pos);
    this.tokens.removeFront();
    var subExpression = this.parseAnd();
    var temp = new Node.BinaryArithmeticNode(pos.getValues(),
      Node.AsebaBinaryOperator.ASEBA_OP_OR, node, subExpression);
    node = temp;
  }
  return node;
}

```

As we can see this method starts by calling another one and this will go on and on until very low methods such as the `parseUnaryExpression`.

## 6. Our Approach

The tree that is built during the parsing which contains the `programNode`, `eventNode` and every other node, is created based on the following class diagram.



This is where we stopped our work on the interpreter, for more information we kindly



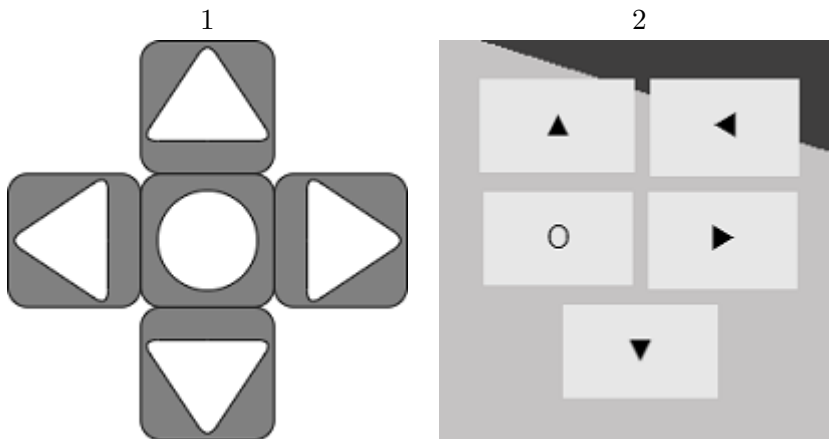
recommend you to look at the section Future research at the page 41.

## 6.5. Physics

Adding physics to allow collisions between the robot and the different objects. Trying to use Physijs, source code can be found in the following repository [Physijs git](#), but with the architecture of our application, we encounter a problem due to the fact we are adding `THREE.Object3D` instead of basic mesh and Physijs works only with a fixed amount of meshes. A solution would be to not use Physijs as we need it only for the Thymio, and instead, shoot rays from the robot and check for collisions this way, as only Thymio needs to move and stop upon collisions.

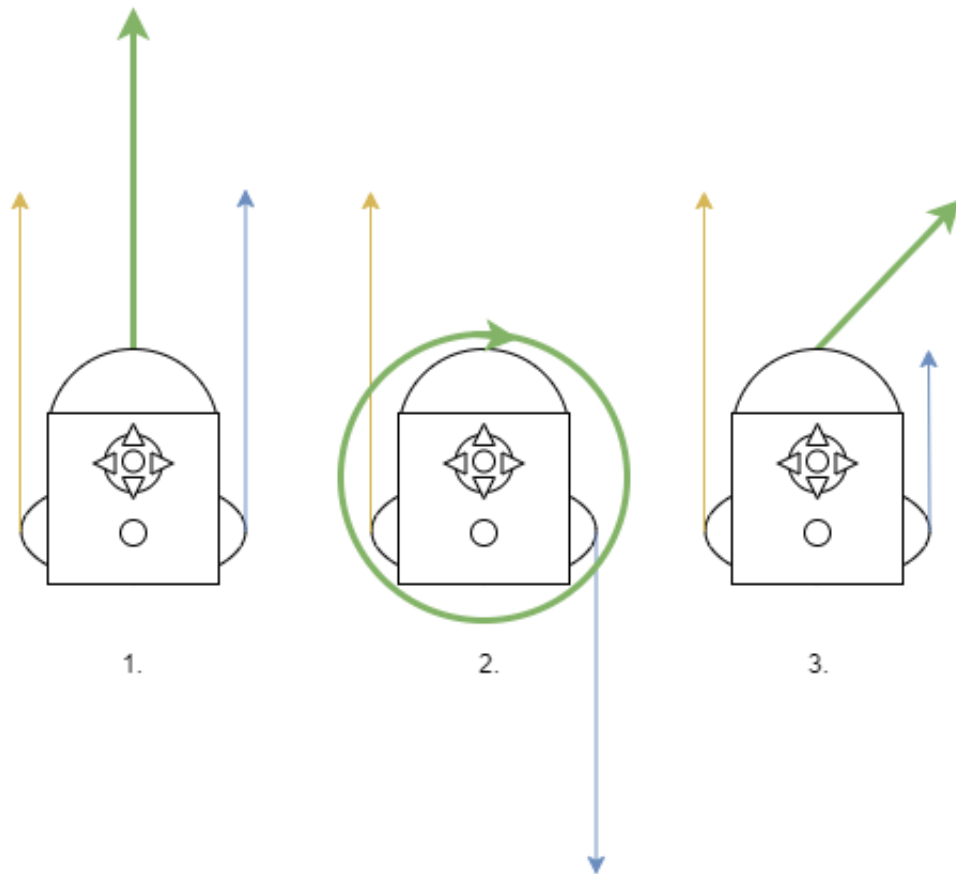
## 6.6. Sensor and Actuator

We represented the five buttons that sit on top of the Thymio directly in the UI and did not give the possibility to interact with the modeled ones. We wanted to create a Directional Pad style group of buttons such as in the first image bellow, but ended with the second image, which is less nice, due to the time needed to make it perfect.



A Thymio robot moves in according with two motors, one for its right wheel and one for its left wheel. They have a value for the output power between -500 and 500, minus meaning the motor powers the wheel in the opposite direction. And those two motors can only output power in a straight line, forward / backward. Bellow are three representations of the expected movements of a Thymio. The yellow arrow is the power for the left motor, the blue arrow is the power for the right motor, and the green arrow is the final vector movement for the robot. On the first representation, the power of the two motors are the same, both in value and direction, thus the final vector is simply the same as either one of the two. On the second one, the value of the power for both motors is still the same but their direction is opposite, hence the final vector is a circle because the robot will turn on itself. And finally, the third representation shows the resultant vector when the value is not the same but the direction is.

## 6. Our Approach



It was complex to convey the way the Thymio robots move onto this application as we move models and don't physically power motors. To achieve the same behavior we decided to use some trigonometry and a few tricks. The method that moves the robot is called only within the `render()` method from the main View, and thus every time the scene is being rendered we move the robot accordingly to the value of the motor. Those values are set using the `setMotors(left, right)` method. The `move()` method, the one that computes the movement of the robot, is split into four categories depending on the values of the motors.

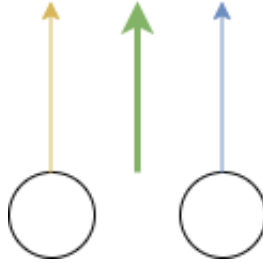
The first category is when both motors have the same power in the same direction, therefore the resulting position change for both x and z coordinate is computed using the `getDirection()`, which returns a `Vector3` representing the direction the robot is facing, a predefined speed variable and the power given to either of the motors.

```
getDirection() {  
  var direction = new THREE.Vector3();  
  return this.object3D.getWorldDirection(direction);  
}
```

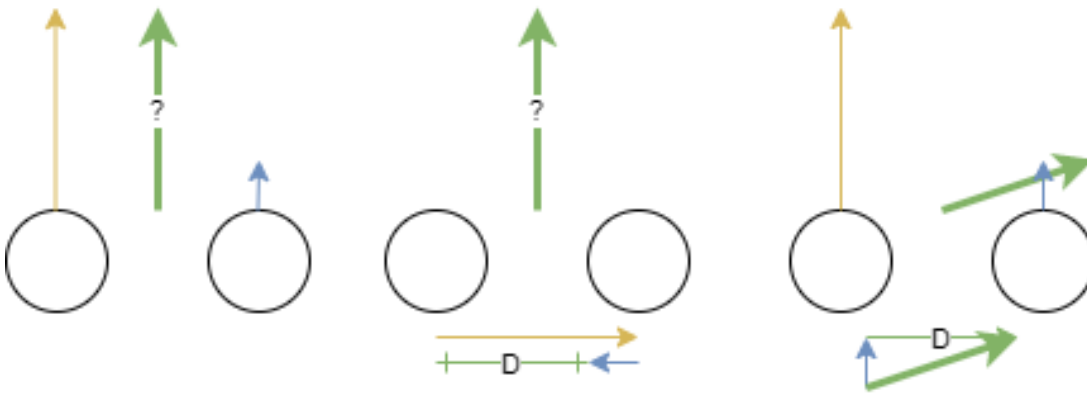
```

this.object3D.position.x +=
  this.getDirection().x * this.speed * this.rightMotor;
this.object3D.position.z +=
  this.getDirection().z * this.speed * this.rightMotor;

```



The second option is when the value of the left motor is bigger than the right, regarding the direction. Alternatively, the third option is the same but when the value of the right motor is bigger than the left.



In this case, we calculate the difference  $D$  of power between the two motors, the biggest minus the smallest, and then we use trigonometry to find the angle between the power of the motor and the resulting vector of the addition of the power of the motor and the difference  $D$ . This angle is then used as a parameter to rotate the model around the  $y$ -axis, to smooth things we multiply it by a variable `turnSpeed`.

```

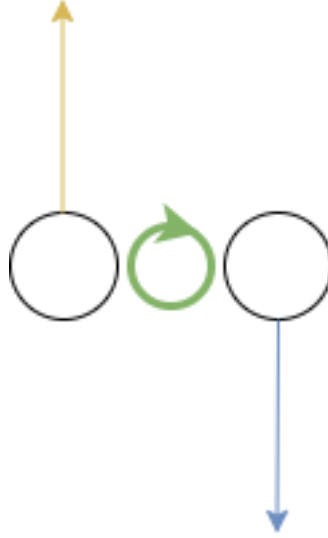
this.object3D.rotateY(
  -(Math.atan(delta/Math.abs(this.leftMotor))*this.turnSpeed)
);

```

The resulting change in position is computed the same way as for the first category. It might be more relevant to use the norm of the resulting vector found earlier to have a more accurate speed.

## 6. Our Approach

The fourth category happens when both power values are the same but in a different direction. In this case, we rotate the robot on itself around the y-axis according to either of its motor speed.



However this algorithm has flaws such as when the power of one motor is 0 the robot should be rotating on itself with said motor as anchor point, but instead it rotate and moves in the direction of the second motors direction.

To determine if a collision occurs between the robot and one of the elements of the environment we decided not to use an external physic library but instead we throw raycast from various positions in the thymio model. To do so we instantiated a **THREE.Raycaster** object and had to find a way to get all 3D objects from our scene. This was quite troublesome because of the implementation of our model view controller, when instantiating a new **Model** we would not get a reference to its **ViewMediator** where the model is stored. We used the benefit of **JavaScript** to find a workaround, whenever a **ViewMediator** is instantiated we add to its model the property **mediator** where the value is the current mediator. Thus we could use the following method in order to skim through the array containing the shapes rendered in the scene and create a new array filled with intersection objects. We then loop through the array to control that the distance is bigger than the given one. If not we operate an action depending on the **className** of the intersected object. In the default case, we assume the robot encounters an element against which he should stop moving.

```
intersects = raycaster.intersectObjects(this.shapes, true);
for(let i = 0; i < intersects.length; i++){
  if (intersects[i].distance < 3.5){
    if(intersects[i].object.mediator){
      model.className === "Plane" ||
```

```

intersects[i].object.mediator.
  model.className === "Octagon"){}
```

```

else if (intersects[i].object.mediator.
  model.className === "Track"){}
```

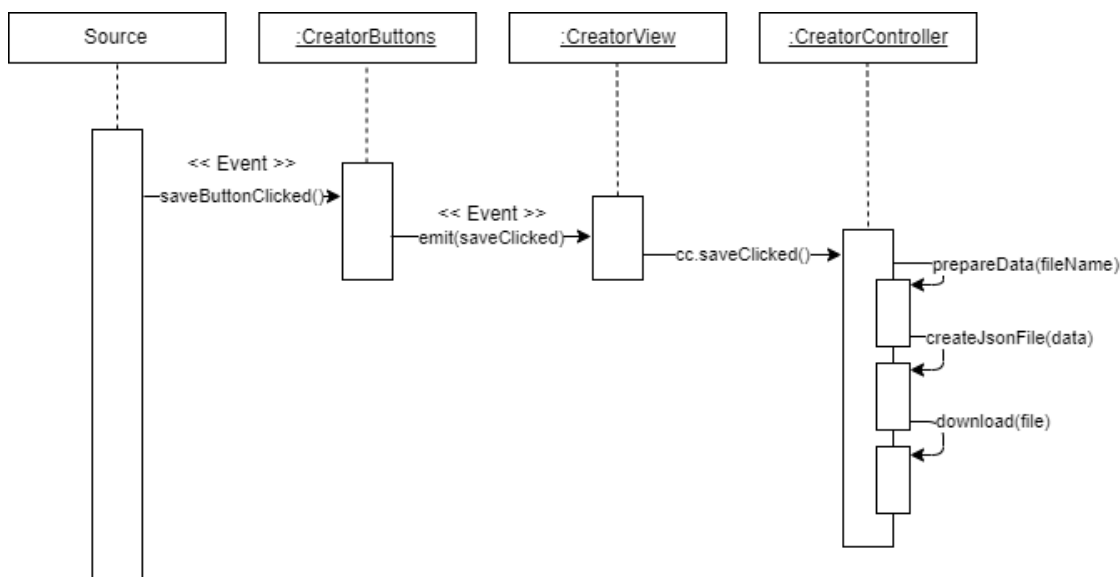
```

else{
  this.stopMotors();
}
}
}
}

```

## 6.7. Customize playgrounds

The creation of a customize playground that can be used later on in the simulation part of the application takes place on a different page. On this page, we keep the same architecture as for the simulation but we change the **Controller** and the **View** elements and don't add the compilation part. Thus most of the changes and logic come from the `CreatorController.js` file. We reflected on how the data of the playground would be carried, or kept, and used in the simulator. Thus we had multiple options, such as using a database to store every customized playground so that they would all be available with the application. Or to download them locally as a file on the user's computer. We went with the latest option as we previously prepared the program to load playgrounds from JSON files. Saving the playground data is done in four different steps, bellow a sequence diagram of the process.



First, once the event that the save button has been pressed the application will open a **prompt** window in which the user will specify the file name. Then the data that composed the customized playground are being separated into different array based on

## 6. Our Approach

their `className` property. Once every shapes contained in the customized playground have been gone through and categorized we create the final JSON file, it is created by using the `forEach` method of JavaScript array and writing the name of the mesh and it's properties. Finally the file is downloaded on the user's computer and the process ends. The JSON file obtained looks like the following.

```
1  "playground":"jailtype",
2    "octagons":[
3      {
4        "name":"ground",
5        "props":{"
6          "segmentLength":35.5,
7          "color":"#bdbbbb"
8        }},
9        "hasWalls":true
10     },
11   ],
12   "boxes":[
13     {
14       "name":"Box0.5",
15       "props":{"
16         "width":4,
17         "height":10,
18         "depth":21,
19         "color":"#ff0202",
20         "positionX":0.5,
21         "positionZ":-17.5,
22         "rotateY":1.6755160819145565
23       }}
24   ],
```

It was needed to add a few more controls over the scene in order to increase the ease of use and the possibilities of creation. The camera controls need to be enabled so that the user can navigate through the scene but once he chose the spot he wants the mesh to be at he needs to deactivate those controls with the `shift` key, and enabled them back with the same key. We decided to add three other controls. The first is the use of the `Esc` key to cancel the current mesh positioning, it removes the current placeholder from the scene. The second and third are linked one to the other, it's the use of the `ctrl-Z` and `ctrl-Y` logic. To do so we create two different arrays, one with the current meshes in the scene and the other one with the removed meshes, which operate as `LIFO Queue`. However we test that the length of the respective array is legal to perform the action and if the current shape the user is creating is from the type `Tracks` then we apply another logic, which removes/reinserts the last point of the track. There are three different shapes at the user's disposal and two types of grounds. A ground element will always be needed to save the current playground, otherwise, an error occurs. The two types of grounds are a simple rectangle and an octagon, with a set of properties. Changing the

properties will not change the mesh in real-time, but once the button **Generate Ground** is pressed it will remove the previous ground and add the new one. The creation of boxes and cylinders takes part in two steps. During the first step, the user will choose the properties, such as **width**, the **length**, the **height**, the **color** and the **rotation** for the box, and then by clicking the **Generate** button of said shape, a placeholder of the shape will be instantiated and it will follow the movements of the mouse while rounding its position to fit into the square represented by the **THREE.GridHelper** element.

```
this.rollOverMesh.position.divideScalar( 1 ).floor()
    .multiplyScalar( 1 )
    .addScalar( 0.5 );
```

Then the user has multiple-choice, either click to fix the mesh onto the scene, or the properties didn't fit what he wanted so he would click once more on the **Generate** button with the new properties, or cancel the action with the **Esc** key. To lay Tracks we needed to create one more step as the mesh is composed of multiple points, those points are laid the same way as a **Box** mesh would but they are added to an array that stores the position of those points. Once the track is finished and upon clicking on the **Generate Track** button the points placeholders will be removed and a track passing through the wanted positions will be computed by looping through the points array and feeding the property element of track with the position of **X** and **Z** of each points.

```
this.points.forEach(pt => {
  this.props.points.push({
    positionX : pt.position.x,
    positionZ : pt.position.z
  });
});
```

However, the algorithm that tracks the movement of the mouse and changes the position of the temporary mesh is not optimized and will slow down the application drastically. Unfortunately we don't know which part of the algorithm has to be optimized.

## 6.8. Incorporated Behavior





## 7. Conclusion

### 7.1. Results

Very difficult to understand a whole C++ application and select the part needed to be translated Interpreter taking way more time than imagined Not possible to use a file as behavior input for the robot, but possible in future with more work

### 7.2. Future research

Write how the next steps come with the tree ( checkVectorSize, expandAbstractNodes, expandVectorialNodes, typeCheck, emit, link) Translate rest of functionality from c++ parser and co, add more sensors and actuator (sounds, pen and co) Better sensors Architecture too heavy and restrictive

### 7.3. Personal conclusion



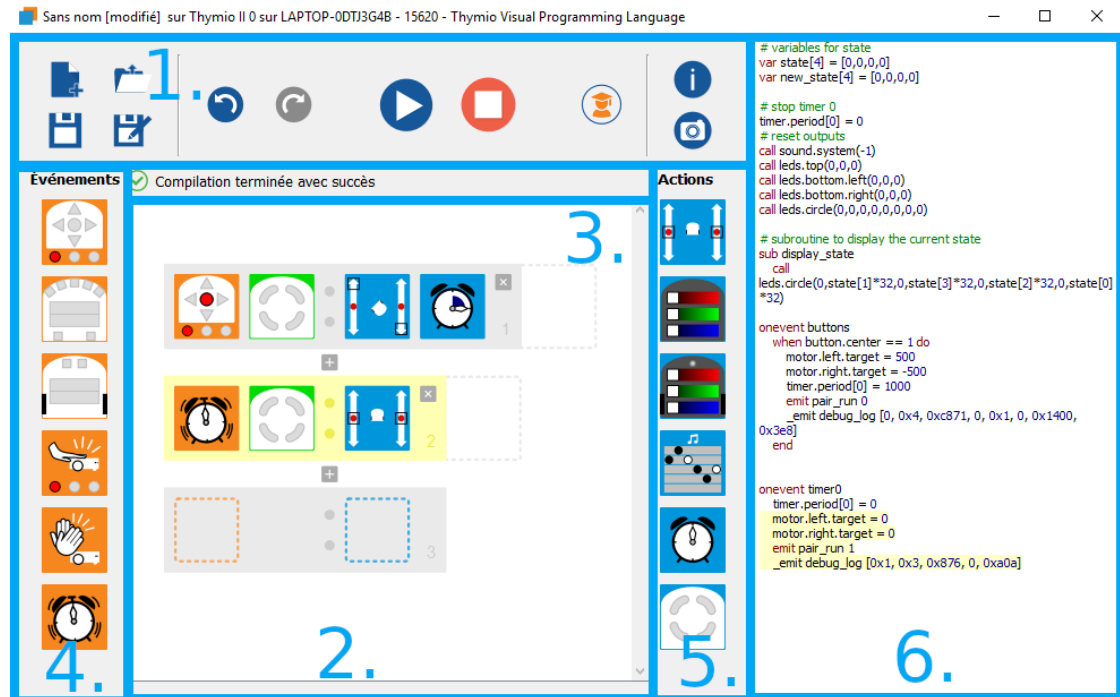
## A. The different programming languages

### A.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.

## A. The different programming languages



Thymio VPL Event and Action blocks

To begin creating a program follow the first steps described in the section 2.2 at the page 10. 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
6. The program translated into AESL

**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.



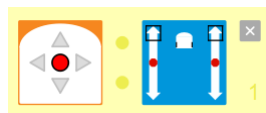
(Advanced mode) Start a timer in the range of 0–4 seconds.  
Click on the clock face to set the time.



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

Thymio VPL Window

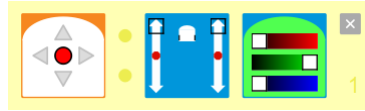
At first, the programming window will be empty of blocks, containing just a placeholder with empty slots. This placeholder is the base of every Thymio VPL program, it contains exactly one event block and one or more action block. This means that whenever the event of the event block happens then the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



Event and one Action relation

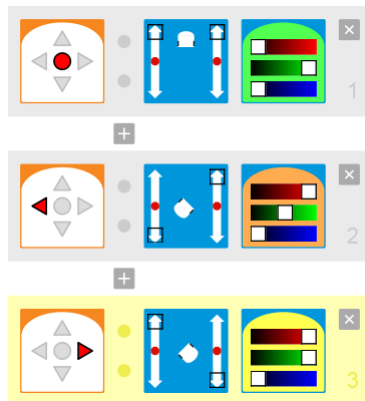
### A. The different programming languages

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:



Event and multiple Actions relation

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:



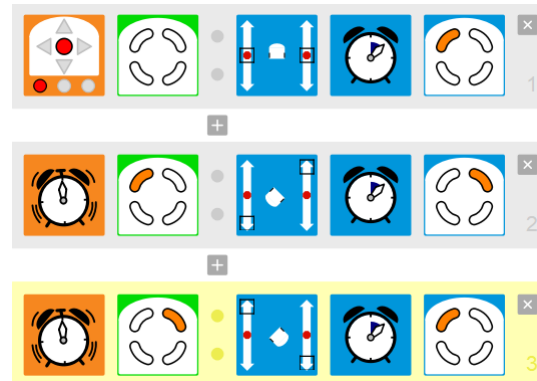
Event and Actions relations

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.

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.

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:



Advanced program

## A.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 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:

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

**Loop:** Blocks for loops.

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

### A. The different programming languages

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

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

**Color:** Blocks with a color definition.

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

**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).



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.



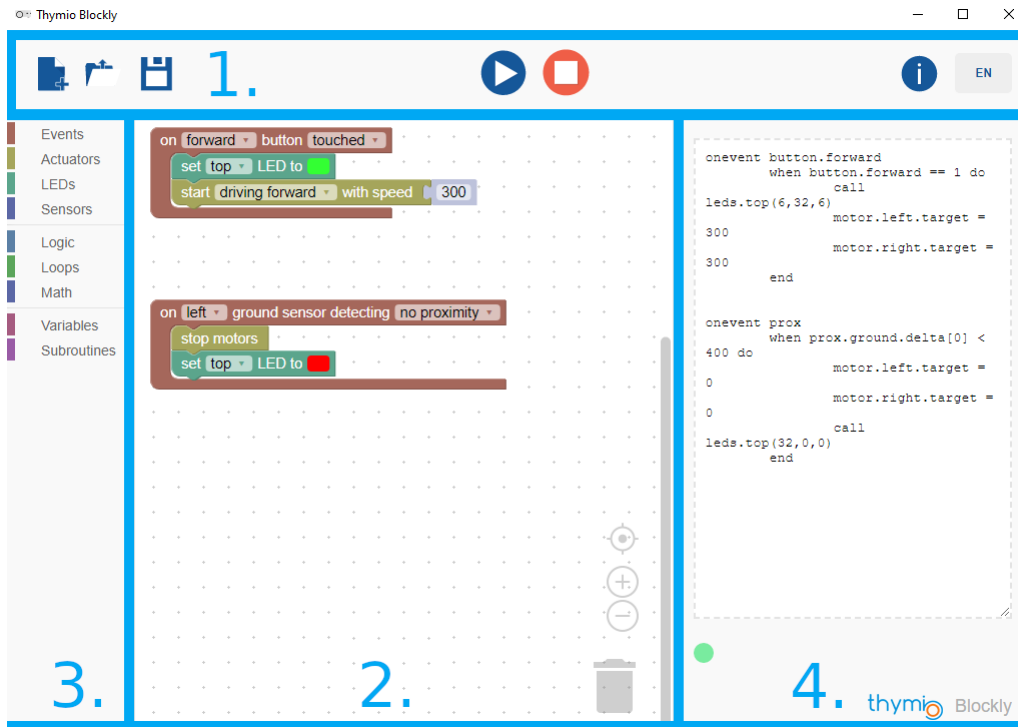
Logic block

Added to the base that Blockly is for `Blockly4Thymio`, is a compiler 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

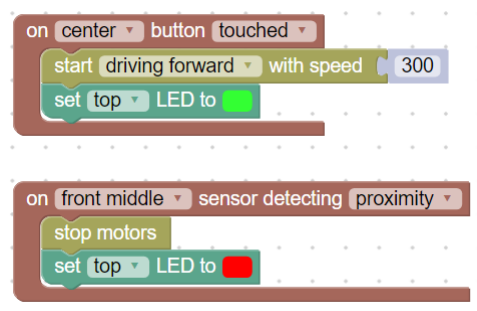


3. The category of blocks
4. The program translated into AESL



Thymio Blockly window

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.

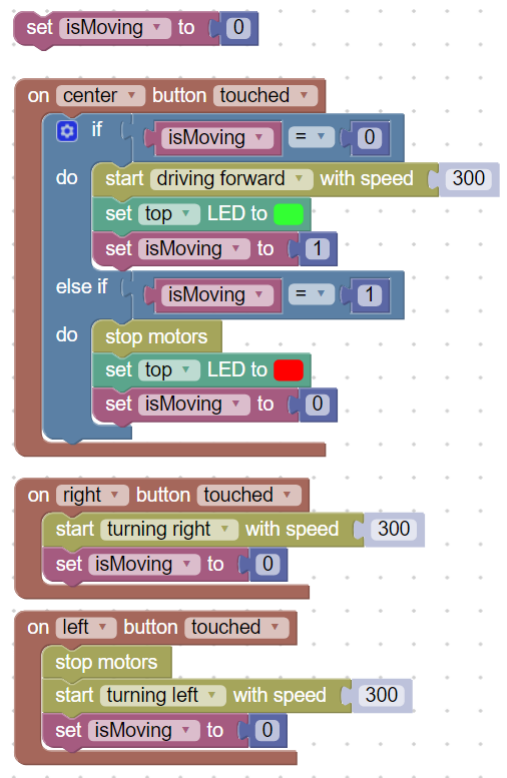


Basic program

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

### A. The different programming languages

stop according to the result. We added two other events for the right and left buttons that are responsible to turn the robot.



More complex program

### A.3. Aseba

For the **Aseba** possibility we kindly recommend you to use the tutorials and example that can be found on the **Thymio** web page.

### A.4. Scratch

For the **Scratch** possibility we kindly recommend you to use the tutorials and example that can be found on the **Thymio** web page.

## B. Product Backlog

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]	Actual Effort [h]	Status
1	Create Documentation	Develop and write the documentation	High	16	16	16	In-Progress
2	Set up the Environment	Setting up and configuring the development environment	High	12	12	12	Done
3	Basic Learning	Learning and training of the different technology used later on	High	44	58	58	Done
4	Develop Playgrounds	Create playgrounds and function to generate meshes	High	40	48	48	Done
5	Update Documentation	Update existing documentation	High	60			In-Progress
6	Architecture Implementation	Refactor the existing code into the designed architecture	High	40	48		In-Progress
7	Web Deployment	Deploy the application on a webserver	High	16	12	12	Done
8	Basic UI	Implement a basic UI	Low	8	4		In-Progress
9	Behavior Pipeline	Create pipeline to take .aesi file and translate/-compile it into behavior in JavaScript for the Thymio II	High	40			To Do
10	Physics Implementation	Implementation of Collisions for threejs Meshes	High	20			To Do

## B. Product Backlog

11	Update Documentation	Update existing documentation	High	20		To Do
12	Enhanced UI	Enhancement of the current UI	Low			To Do
13	Customizable Playgrounds	Implementation of a playground creator for users	High			To Do
14	Update Documentation	Update existing documentation	High	20		To Do
15	Enhanced Behavior Pipeline	Implement more sensors, action and event for Thymio II	High			To Do
16	Finish Documentation	Finish existing documentation	High			To Do
17	Prepare Defense	Prepare the defense	High			To Do
			Total			

## C. Sprint Backlog

### C.1. First Sprint

2019-09-16 until 2019-10-07

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]	Actual Effort [h]	Status
1	Create Documentation	Develop and write the documentation	High	16	16	16	Done
1.1	Template and Content	Choose a latex template and modify the content structure	High	8	8	8	Done
1.2	About Thymio	What is thymio and how does it work.	High	8	8	8	Done
2	Set up the Environment	Setting up and configuring the development environment	High	12	12	12	Done
2.1	GitHub	Create the project in GitHub and the Git environment	High	4	4	4	Done
2.2	Tools	Install Thymio Suite, NodeJS and download ThreeJS	High	8	8	8	Done

### C.2. Second Sprint

2019-10-07 until 2019-10-28

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]	Actual Effort [h]	Status
3	Basic Learning	Learning and training of the different technology used later on	High	44	58	58	Done

### C. Sprint Backlog

3.1	threejs	Read documentation and examples, and practice	High	16	20	20	Done
3.2	JavaScript	Update and deepen knowledge	High	8	8	8	Done
3.3	Thymio languages	Learning and using VPL, Blockly, Aseba and Scratch	Medium	24	30	30	Done
4	Develop Playgrounds	Create playgrounds and function to generate meshes	High	40	48	48	Done
4.1	Two Default Playgrounds	Generating two default playground to be choosen for the simulator	Medium	12	12	12	Done
4.2	Thymio Model	Create or load Thymio model	Medium	4	4	4	Done
4.3	Mesh Generation	Create function to generate meshes for the playgrounds	High	24	32	32	Done
5	Update Documentation	Update existing documentation	High	60	12	/	Done
5.1	Four supported languages	Descibe and initiate to VPL, Blockly, Aseba and Scratch	High	24	12	/	Done

### C.3. Third Sprint

**2019-10-28 until 2019-11-15**

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]	Actual Effort [h]	Status
5	Update Documentation	Update existing documentation	High	60	76	76	Done
5.1	Four supported languages	Descibe and initiate to VPL, Blockly, Aseba and Scratch	High	24	32	32	Done
5.2	Backlogs	Create the Sprint and Project backlog	High	12	16	16	Done

#### C.4. Fourth Sprint

5.3	Architecture	Create DCD, DM, PD, SD, SSD and proposition of architecture	High	12	16	16	Done
5.4	User Stories	Formulate the User Stories	Medium	4	4	4	Done
5.5	Risk Analysis	Create risk analysis	High	8	8	8	Done
6	Architecture Implementation	Refactor the existing code into the designed architecture	High	40	48	48	Done
6.1	Refactor Code	Refactor existing code into MVC Pattern	High	20	44	44	Done
6.3	Unit Testing	Write the JavaScript tests	High	12	/	/	To Do
6.4	JSDoc	Write the JavaScript-Doc	High	8	4	4	Done
7	Web Deployment	Deploy the application on a webserver	High	16	12	12	Done
7.1	Virtual Machine Setup	Set up the Virtual machine	High	8	4	4	Done
7.2	WebServer	Create WebServer and publish it on bfh network	High	8	8	8	Done
8	Basic UI	Implement a basic UI	Low	8	8	8	Done
8.1	Pages UI	Create three pages UI, one for each of the following index, simulation and creation pages	Low	8	8		Done

#### C.4. Fourth Sprint

2019-11-15 until 2019-12-09

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]	Actual Effort [h]	Status
9	Behavior Pipeline	Create pipeline to take .aesi file and translate/-compile it into behavior in JavaScript for the Thymio II	High	40			To Do

### C. Sprint Backlog

10	Physics Implementation	Implementation of Collisions for ThreeJS Meshes	High	20			To Do
11	Update Documentation	Update existing documentation	High	20			To Do

## C.5. Fifth Sprint

**2019-12-09 until 2019-12-30**

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]	Actual Effort[h]	Status
12	Enhanced UI	Enhancement of the current UI	Low	10	/	/	To Do
13	Customizable Playgrounds	Implementation of a playground creator for users	High	40	24	24	Done
14	Update Documentation	Update existing documentation	High	20	24	24	Done

## C.6. Sixth Sprint

**2019-12-30 until 2020-01-17**

ID	Story Name	Story / Task Description	Priority	Est. Effort [h]	Update Effort [h]	Actual Effort[h]	Status
15	Enhanced Behavior Pipeline	Implement more sensors, action and event for Thymio II	High				To Do
16	Finish Documentation	Finish existing documentation	High				To Do
16.1	Create Video	Create the video file	High	16			To Do
16.2	Prepare Presentation Day	Create the poster and the presentation for the Presentation Day	High	16	4		In-Progress



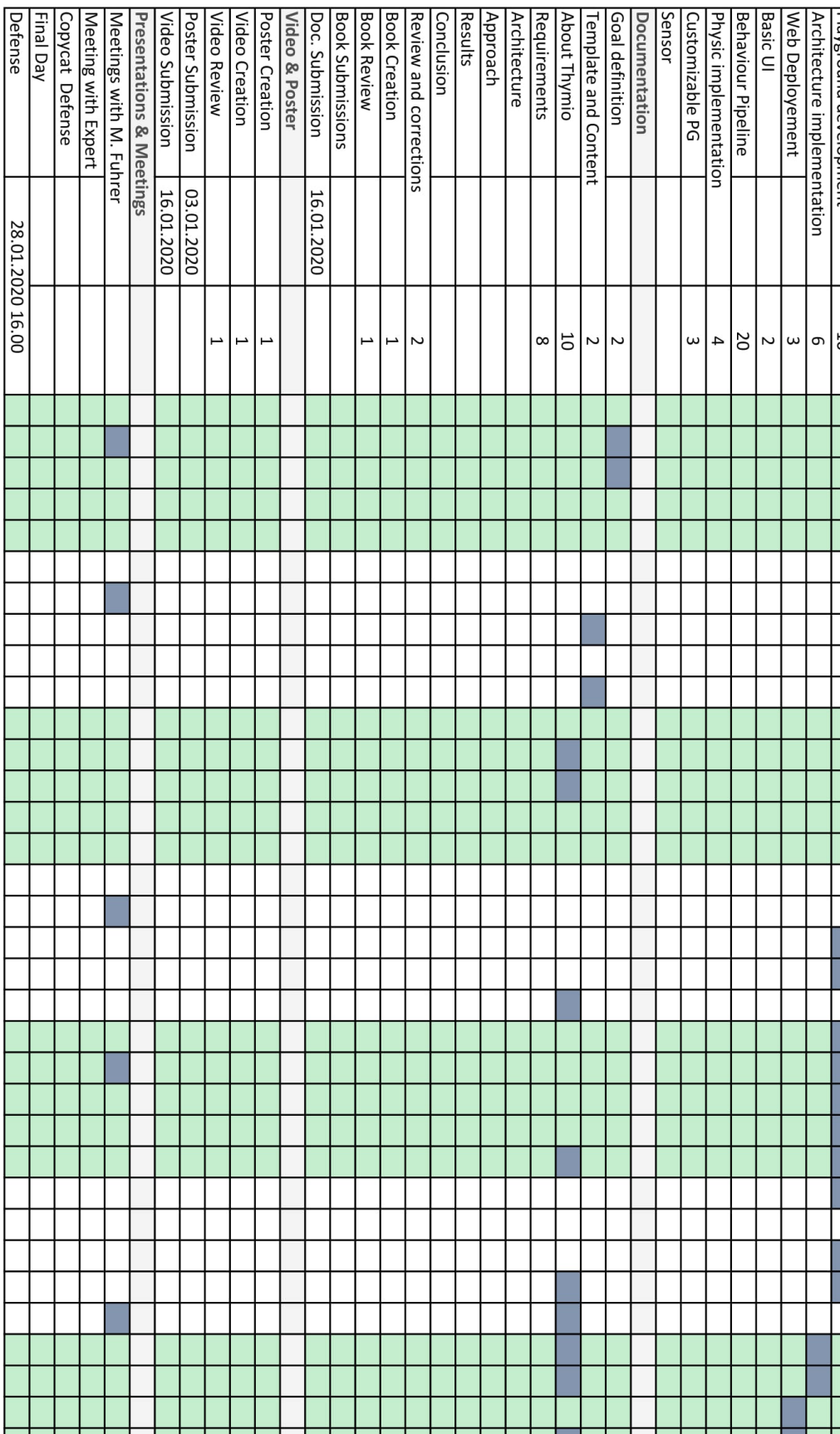
### C.6. Sixth Sprint

16.3	Write in the Book	Write the page for the Book	High	8	4	4	Done
16.4	Finish Writting Documentation	Terminate the writting part of the documentation	High	32	24		In-Progress
16.5	Prepare to Submit	Check spelling mistake, check images, print it, put the project on a USB stick	High	8			To Do
17	Prepare Defense	Prepare the defense	High				To Do



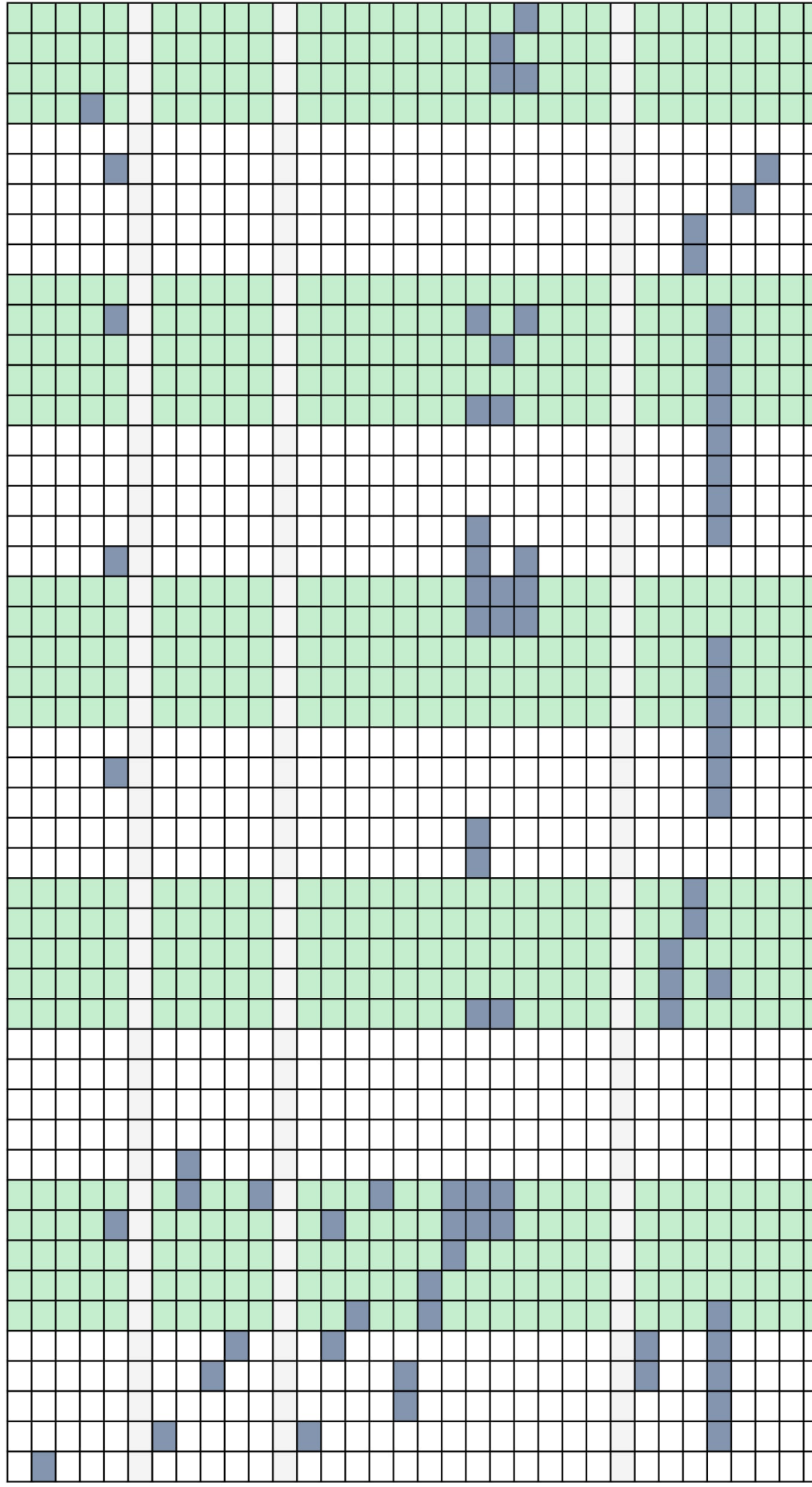


## D. Gantt Diagram





*D. Gantt Diagram*



## **E. Version control**





## F. 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 network to access it) <https://intranet.bfh.ch/TI/fr/Studium/Bachelor/Informatik/Tools/VMsHowto/Pages/default.aspx?k=vm>

#### First Configuration of XAMPP

The configuration of XAMPP is very basic. The steps done during the setup of the IIS Manager at 6.2, concerning forwarding and firewall, should still be followed. First we need to download and install it, it can be found under this link : <https://www.apachefriends.org/index.html> . Afterward it is needed to travel through the folder of the application until the folder `htdocs`. There the default placeholder files was stored, we put them into a new default folder and instead added our content in this folder. Do not forget to take the `.webconfig` file from the 6.2 section.





## G. Meetings

Date	Content
17.09.2019	<b>Kick Off meeting</b> <ul style="list-style-type: none"><li>- Documentation/Management</li><li>- Technology to use : ThreeJS and Typescript</li><li>- Setting up the goals</li></ul>
24.09.2019	<b>Second meeting</b> <ul style="list-style-type: none"><li>- Documentation language : English</li><li>- Thymio model</li><li>- Base talk about riks management</li></ul>
08.10.2019	<b>Third meeting</b> <p>Workplace</p> <ul style="list-style-type: none"><li>- Discussion on the choice of Windows as the Virtual Machine</li><li>- Create a configuration file with the information of the VM</li><li>- And an architecture proposal</li></ul>
15.10.2019	<b>Fourth meeting</b> <ul style="list-style-type: none"><li>- Which shapes and meshes should the user be able to create for his own custom playground</li><li>- Problems with webserver, has to be accessible from outside the vm, so maybe switching from windows to linux</li><li>- 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</li></ul>
25.10.2019	<b>Fifth meeting</b> <ul style="list-style-type: none"><li>- 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</li><li>- First little talk about the meeting with the expert, report</li></ul>
19.11.2019	<b>Sixth meeting</b>
07.12.2019	<b>Seventh meeting</b>
19.12.2019	<b>Eighth meeting</b> <ul style="list-style-type: none"><li>- Documentation review</li><li>- What to hand in to the secretariat and to Mr. Fuhrer</li></ul>
07.01.2020	<b>Ninth meeting</b> <ul style="list-style-type: none"><li>- Documentation state</li><li>- Latex commands</li><li>- Final day and defense talk</li></ul>

## H. 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 all 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 .aesi file previously created with it, but I could load them if I used the index.html one.
- Thymio model not always loading correctly -> Load one at the start of the page and reset its position/rotation upon change of playground.
- problem with dat.gui, it was creating a new creator view so the creator wasn't accessible. Decided to use html buttons instead
- c++ to javascript



## Sujet de mémoire de bachelor

pour Quentin Flückiger  
Division Informatique  
Responsable(s) Claude Fuhrer

### Simulation web d'un robot Thymio

Le robot Thymio (et ThymioII) a été développé par une startup de l'EPFL dans le but de promouvoir la programmation et les activités robotiques chez les enfants. Il supporte 4 langages de développement, à savoir:

- Programmation visuelle (VP)
- Scratch
- Blockly
- Aseba (langage orienté événement)

Dans le cadre de ce travail nous allons développer un environnement de simulation du robot Thymio pur web, c'est-à-dire ne nécessitant aucune installation de logiciel pour l'utilisateur. Le simulateur proposera les fonctionnalités suivantes:

- Vue 3D du robot et de son environnement
- Simulation des capteurs du robot réel
- Un outil simple pour la création d'un environnement d'expérimentation.
- Intégration du langage de programmation Aseba.

La technologie utilisée pour développer ce simulateur devra être compatible avec les browsers modernes (Firefox, Chrome, Safari, Edge)

Début du travail 16 septembre 2019  
Fin du travail 16 janvier 2020

Le responsable:

Le directeur de division: