Web Simulation of a Thymio Robot

Bachelor thesis

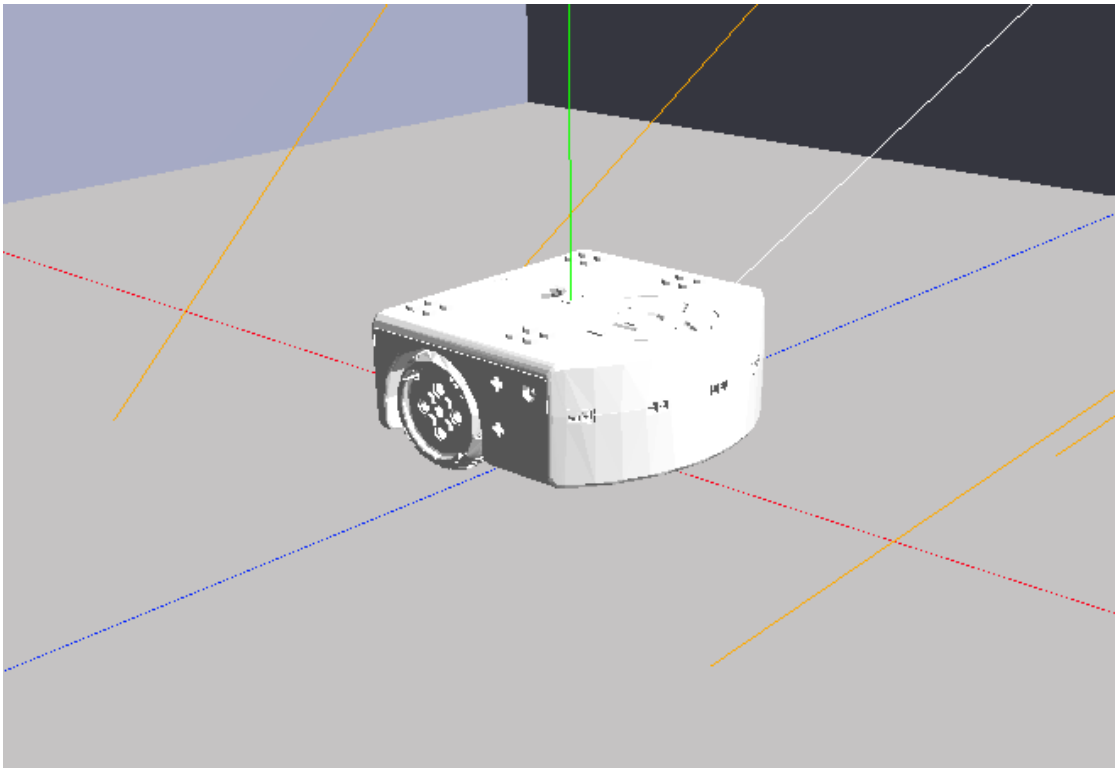
Degree programme: Computer Science

Author: Quentin Flückiger (`flucq1@bfh.ch`)

Thesis Advisor: Prof. Claude Fuhrer

Expert: Dr. Eric Dubuis

January 16, 2020



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 fonctionalités suivantes:

- Vue 3D du robot et de son environnement
- Simulation des capteurs du robot réel
- Un outil simple pour la création d'un environment 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:

Management Summary

As information technology has grown massively in everyday life during the last decade, many questions have arisen. A very important one, at the very least in our minds, concerns the educational side. To what extent should information technology be taught, in mandatory school, high school, vocational education, or even self-taught? In Switzerland, some optional courses in this field have already been introduced in the curriculum of some primary schools. Of course, the schoolkids are not taught too deep and complicated concepts, but they are rather familiarized with the way of reasoning and improving logical concepts. To ease the entry into the world of information technology, some schools have developed projects. One example is the Thymio project of the Swiss Federal Institute of Technology of Lausanne. Thymio is a small robot that can be programmed on a computer with four different languages, with varying degrees of difficulty. Thus, it allows very young people to experience and learn some of the basics of information technology. Unfortunately, this robot has a few disadvantages. One example is the space required or the setup of its environment. In order to fill these gaps, we have decided to create a simulation of the Thymio. Therefore, the objective of this project is the creation of a Thymio simulator. The simulator has to be a web application to ease the accessibility, where no software installation is required, and it has to be compatible with modern browsers.

We started the project by learning about the four different languages supported by the robot, namely VPL, Blockly, Aseba, and Scratch. We were interested in knowing 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. In this sens, we chose `three.js` as a library.

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

Last but not least, we had to develop a compiler that would translate a program written for Thymio in one of the four languages. In order 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 and to translate them into `JavaScript`.

Contents

1. Requirements Documentation

1.1. Introduction

As information technology has grown massively in everyday life during the last decade, many questions have arisen. A very important one, at the very least in our minds, concerns the educational side. To what extent should information technology be taught, in mandatory school, high school, vocational education, or even self-taught? In Switzerland, some optional courses in this field have already been introduced in the curriculum of some primary schools. Of course, the schoolkids are not taught too deep and complicated concepts, but they are rather familiarized with the way of reasoning and improving logical concepts. To ease the entry into the world of information technology, some schools have developed projects.

1.2. Vision

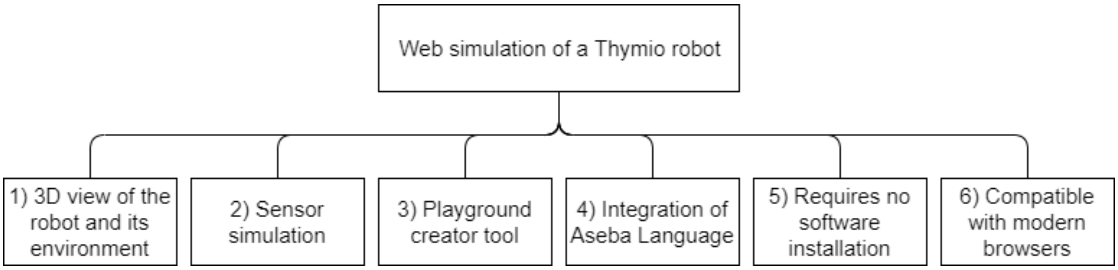
The Thymio robot or Thymio II is a robot developed by a start-up from the EPFL in order to promote programming and robotic activities among children. To feed program to the robot, a software has been developed. It integrates the four following programming languages:

- VPL
- Blockly4Thymio
- Aseba
- Scratch

The project Web Simulation of a Thymio robot aims at creating a simulator for the Thymio II so as to allow people to directly see their programmed behavior.

1.3. Goals

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



1.4. Risk Analysis

In order to carry out the project successfully, one must consider several possible complications. These 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.

	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
	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
Likelihood (A)	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

1.5. 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.

1.6. User Stories

Users User Stories
As a user, I want to upload an .aesl file, so that I can witness the simulated behavior.
Description:
The user wants to upload an .aesl file to see the programmed behavior simulated. Success:

- The simulation works.

Failure:

- The .aesl file does not contain a program.
- The .aesl 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 a home made playground with a simple playground creation tool. Success:

- The playground is successfully created and saved.

Failure:

- The created playground is not 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 is not accessible.

1.7. 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:

1. User starts the web browser.
2. User navigates to the 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: Interprete .aesl file
Primary Actor: User
Stakeholders and Interests: User: Wants to load .aesl behavior file to be translated and simulated.
Preconditions: User has a .aesl file containing behavior code for Thymio.
Success Guarantee (Postconditions): File is correctly compiled, and simulation simulate expected behavior.
Main Success Scenario:

1. User accesses the website.
2. User input a .aesl file.
3. System controls the file integrity.
4. System compiles the file to `JavaScript` code that can be run as behavior.
5. System runs the given program.

Extensions:

2. a) File too large for application.

3. a) System signals error and rejects the file because it is not conform to the 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 whishes of the user.
Main Success Scenario:

1. User accesses the website.

2. User chooses the wanted playground.

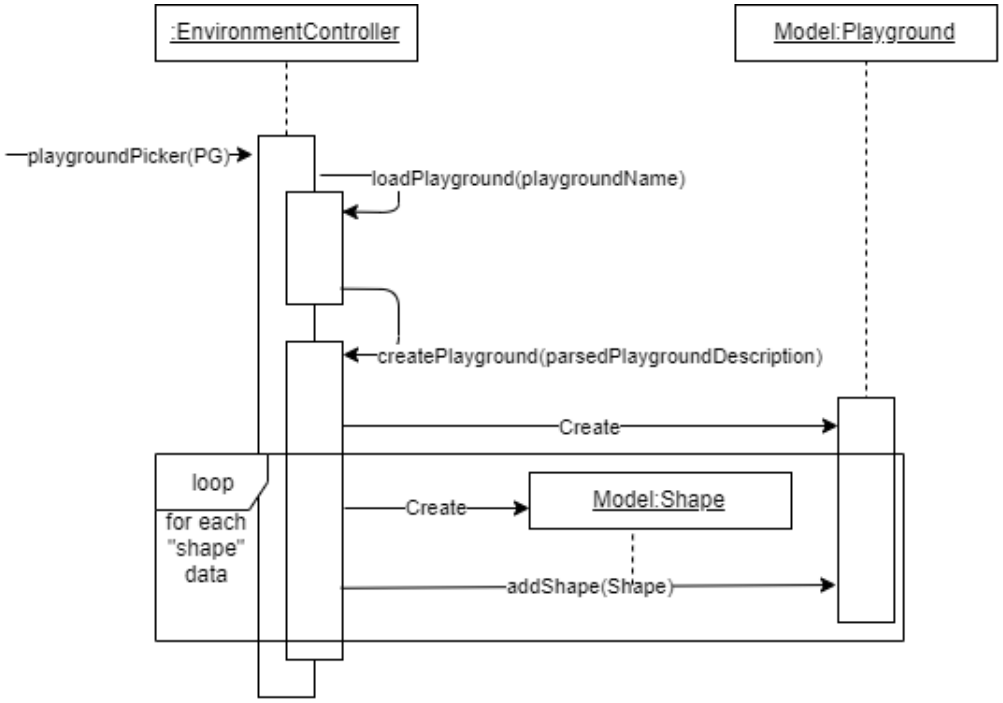
3. System loads the wanted playground.

Extensions:

2. a) The input file is not from the right file extension.

3. a) Fail to load playground because the file is not conform to the awaited structure.
   b) Internal error when the playground was loaded.

Special Requirements: -
Technology and Data Variations List: -
Frequency of Occurrence: Often.
Open Issues: -
Sequence diagram:


:EnvironmentController
Model:Playground
playgroundPicker(PG)
loadPlayground(playgroundName)
createPlayground(parsedPlaygroundDescription)
Create
loop
for each "shape" data
Create
Model:Shape
addShape(Shape)

2. Thymio

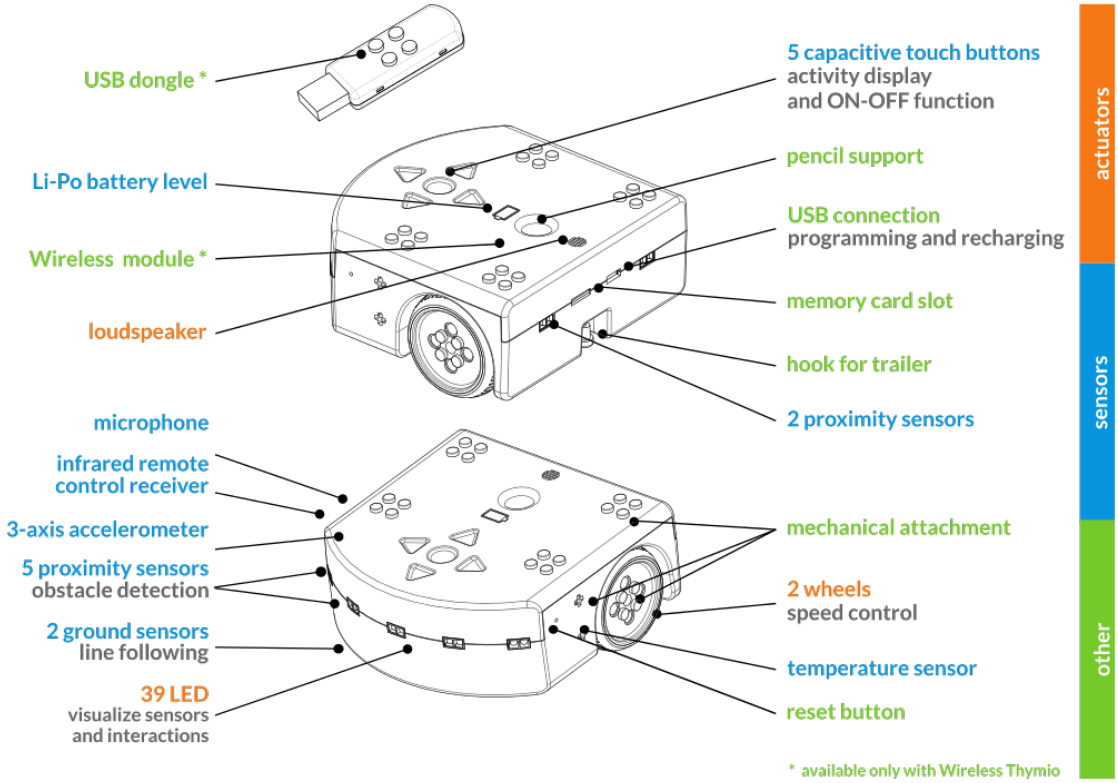
2.1. What is Thymio

Thymio is an educational robot that aims at improving early education (starting in primary school) in Science, Technology, Engineering and Mathematics (STEM), computational thinking, base computer science and at 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 mainly focus on system design for small robots of the kind. The project started with a strange 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. "Monsieuer Patate" saw life during the first workshop between the two contributors. Afterward, the first "Thymio" was developed, it was a four-block robot that could be self-assembled, but not self-programmed as it was delivered with pre-programmed behaviors. Thymio was used as a user study to gather feedback from clients in order to know which features needed to be implemented on the Thymio II.

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. The production distribution and communication of the robot was overseen by Mobsya, a non-profit organization that creates robots, software, and educational activities to broaden young people's mind about technology and science. Every step of the Thymio project is open-source and has a non-profit aim, namely to enhance the quality of it with the user's project and research as well as to reduce the cost and augment the lifetime for educational platforms and materials.



Thymio II sensor and actuator

2.2. How does it works

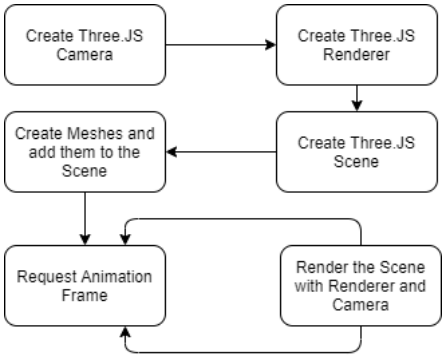
As pictured in the figure above there are two Thymio models, namely Thymio and Wireless Thymio. The difference between them lies in the ability of the latter one to be programmed wirelessly, as suggested by its name. There are two possibilities to begin the creation of a program for the robot there. 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 powered on. Then, the software can be used to connect to said robot and to start programming in one of the four different programming languages. These 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. As the software possesses ots pwm bioét-in simulator it does not require a physically or wirelessly connected Thymio robot. The same four programming languages are available and one has to be chosen. After having selected the language comes the choice of either connecting a physical Thymio or starting a simulation to emulate the programmed behavior. A more detailed section on the four differents programming languages can be found in section A on page 40.

3. Environment

3.1. 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 :

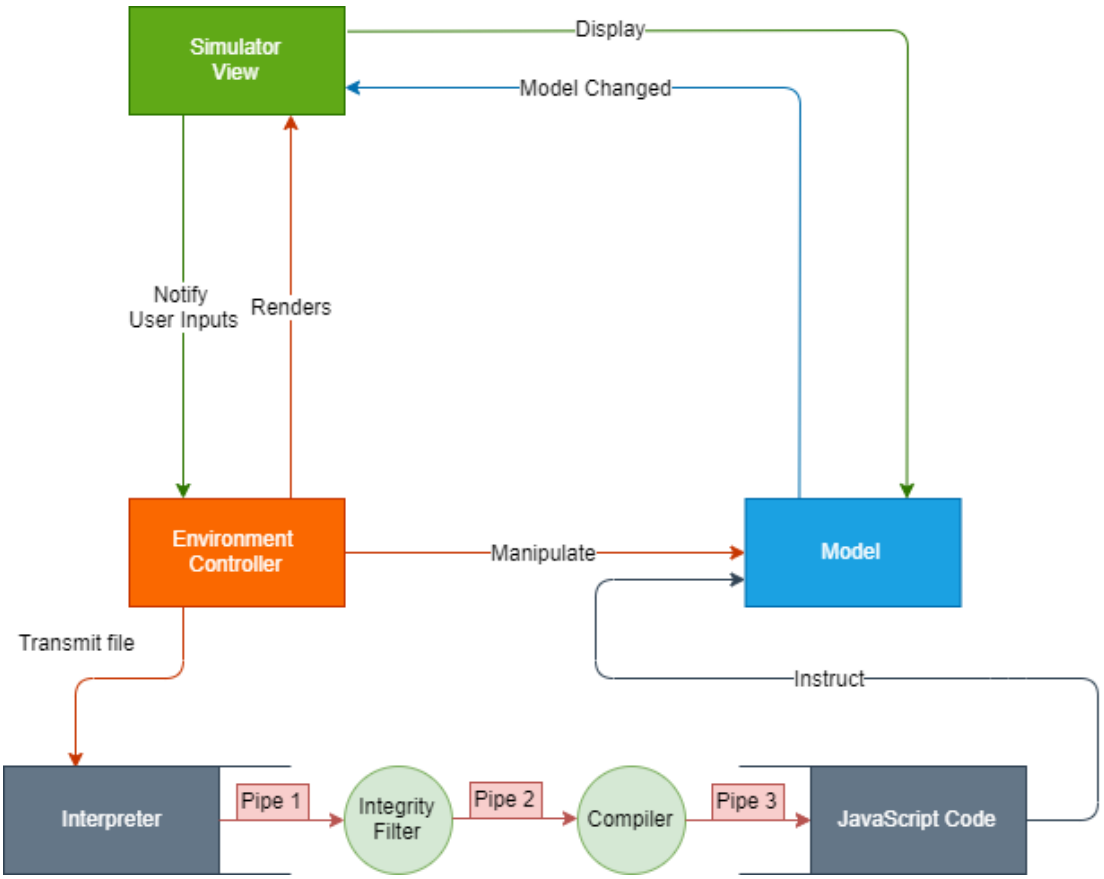


Every `three.js` application is composed of at least one `Camera` element, a `Renderer` element and a `Scene` element. The different meshes are added to the scene and this scene along with the camera is rendered using the `three.js` method `requestAnimationFrame` on the renderer element. Using the `three.js`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

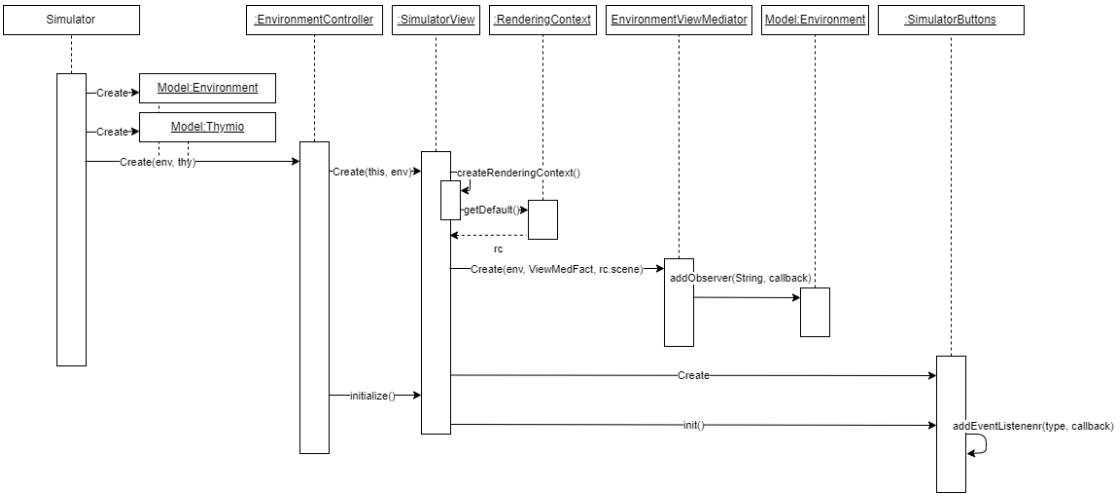
4. Architecture

It was 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 from two different angles. Both parts 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 seen by the user is the `View`. This `View` registers the user inputs and transmits them to the `Controller`.

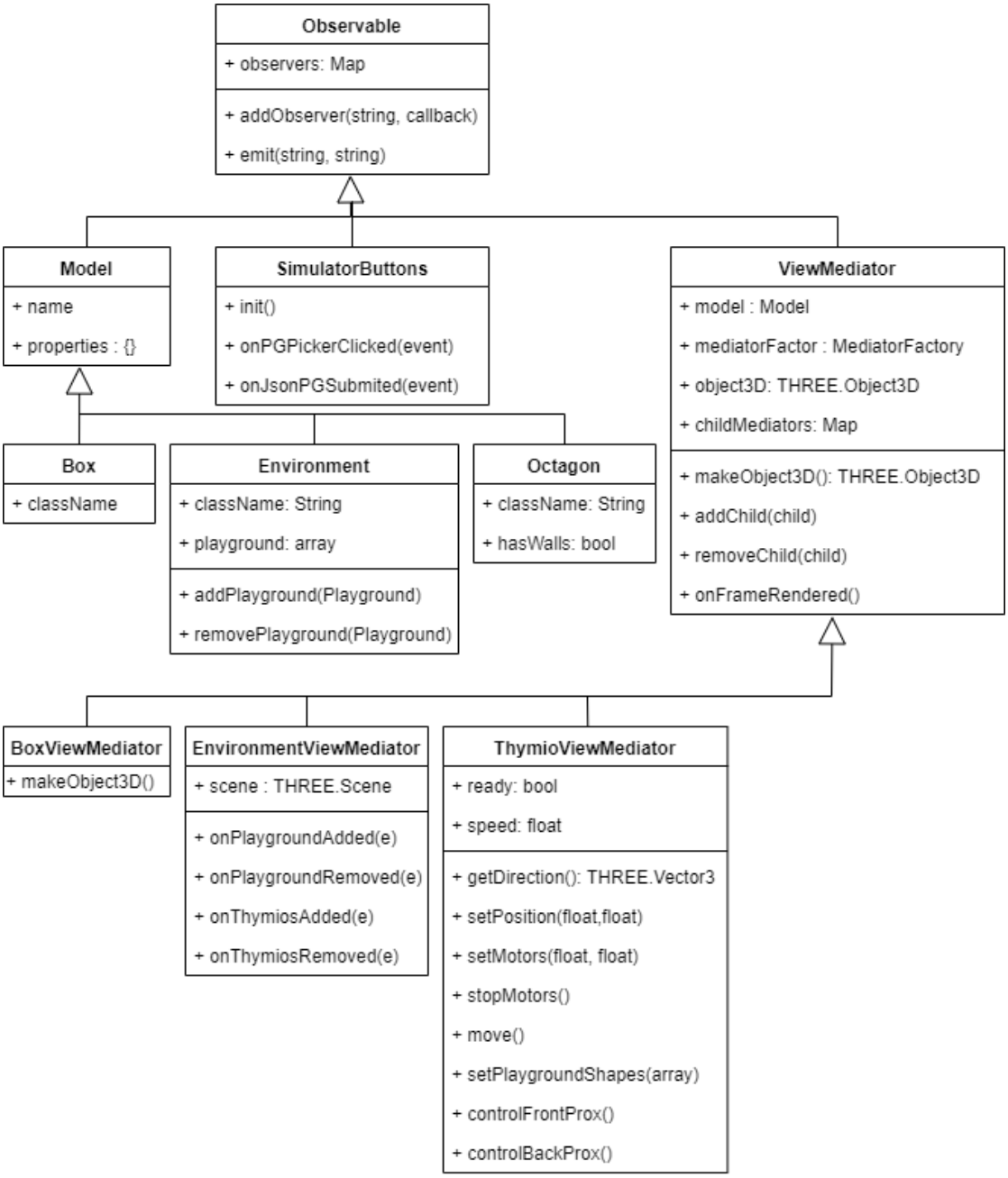
As said we have split the architecture, bellow is a graphic representing the architecture for the simulator. The `Model View Controller` can be seen as well as a second component which is the interpreter. The interpreter, is singular to the simulator page. Its role is to test the integrity of the file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



The used `MVC` 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`.



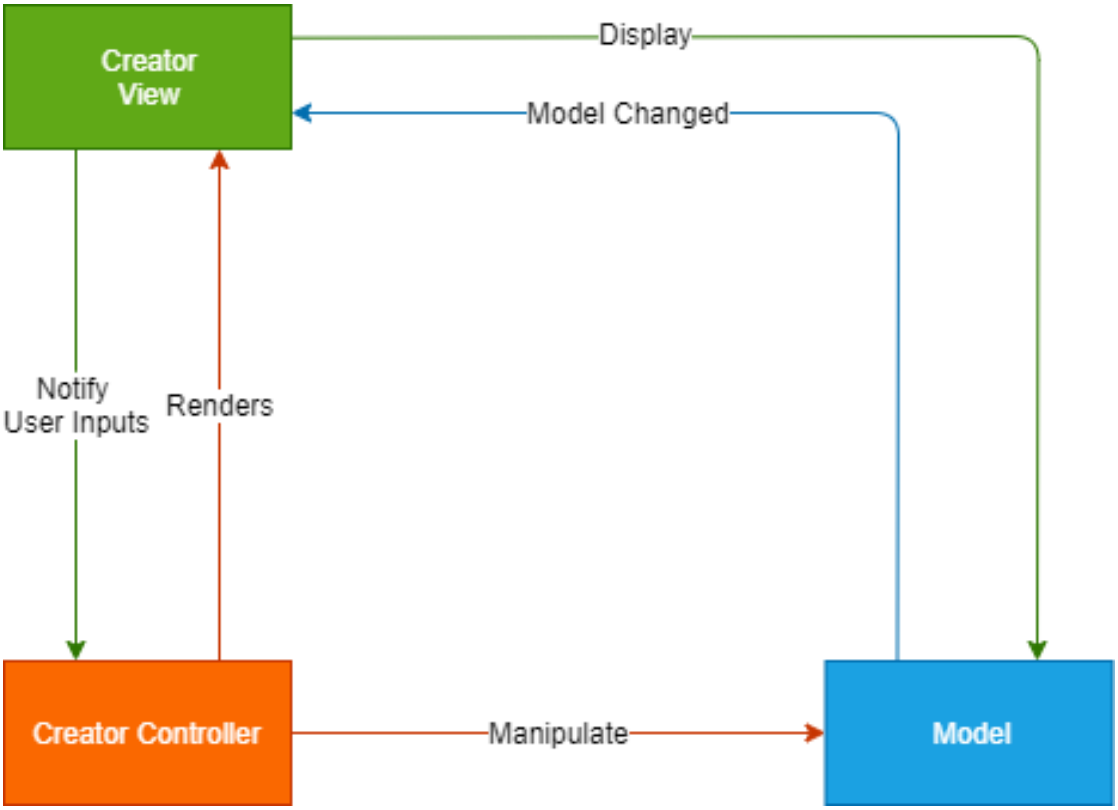
The `Model` part is split into three categories. They extends the same base class `Observable` which works as an event system with the `addObserver` and `emit` method. The first category is the one responsible for the buttons. We instantiate one of the right types, depending on the `View`, at the start 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 it is responsible for creating the 3D objects with the data from its model. Further, it is responsible for the logic that could be applied 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`. For clarity purpose, not all classes are represented as their configuration is very similar to one comprised in the diagram.



While the `View` and the `Controller` elements are different in the two pages, the `Model` is not. Int the simulation side, the `Controller` loads the playgrounds, be it built-in or coming from the device, and uploads the aesl file to the beginning of the filter/compiler. 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`, which is how an observer is added.

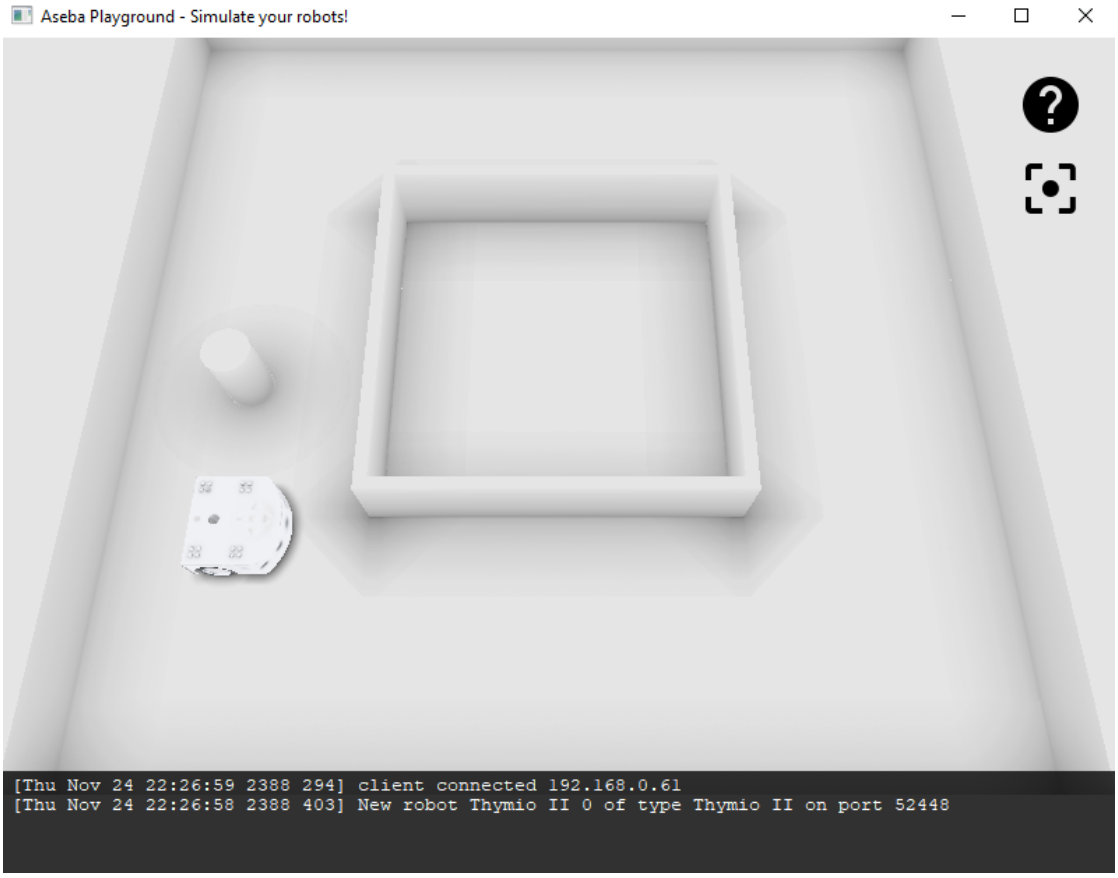
```
this.simulatorButtons.addObserver('jsonPGSubmited',
 (e) => this.controller.onJsonPGSubmited(e));
```

Here is the second `Model View Controller` responsible for the playground customization. Its architecture is the same as the other one, except that it does not implement an interpreter.



5. What already exists

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 groups 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-sets and to run the coded program for the robot.



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

Apache 2 license. WeBots is a very powerful software that can do a lot of things. One of these features 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. The Approach

6.1. Base Development

The first step was to learn how to use the `three.js` library. Thus, the needed code was directly integrated into some .html file. This integration was not very efficient for a big application, but it was acceptable to test the functionalities. At that time, some useful methods that would be needed later on were looked for, such as the way to resize the window and its content. Another functionality added at this period was the control of the camera which comes from the OrbitControls.js file and allows the creation of an OrbitControls object. This object enables the user to move the camera, rotate it and zoom with the mouse. Afterward, a method was implemented to create different `Shapes` with more ease for a larger application. Then the scripting part was moved 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", it was decided to use a webserver in order to access the application. At first, the IIS Manager that is built-in with Windows 10 was chosen. The `ISS` configuration followed the steps shown in this tutorial video: `https://www.youtube.com/watch?v=rPRLe7QeVHM` .

Then, the port had to be opened in order to access it from within the LAN. This was done as follows:

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

```
<?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>
```

Unfortunately, after having implemented the new `MVC` architecture for the application, a further issue was encountered where the application was not able to locate some `JavaScript` file and gave 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 specialists, it was decided to move away from ISS Manager. The reasons for this decision were the limited knowledge of this software. The time needed to acquire this knowledge was not worth the effort. The alternative solution we came up with was to use `XAMPP`. As the same version of the application could be pushed without any problem and without having to do anything special, it was decided to continue with `XAMPP`.

6.3. Playgrounds

It was 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 obstacles, walls and a track. Each build-in playground would be composed of one `THREE.Group` element that is filled with different meshes created from the previously implemented method found in the `GeometricalMeshes.js` file.

In order to enhance the amount of different meshes, an algorithm to create tracks was added. The tracks creation happened in two steps. The first one consisted into computing a simple line between two points. Unfortunately, the result was too thin and therefore a better solution was found. The second iteration for this algorithm takes an array of points, which are of type `THREE.Vector3` so as to register the three coordinates, and computes 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 represents the track, and are then 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);
}
```

It was decided to load the Thymio model only once, as it takes in average 500ms to 1'000ms to load it, and as some issues where the model would not load every time the playground was changed were encountered. Therefore, its position and rotation are reset whenever the playground is changed. 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, it was decided to change how the playgrounds data were recorded. Instead of having them as `JavaScript` files, they were all moved into a `JSON` file. In this way, it would not require more work to load a customized playground later on. To do so, a `JSON` file of a given name was opened, read, and then skimmed through the `JSON` file. The `Shapes` were created accordingly of the data. Bellow an example for the boxes.

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

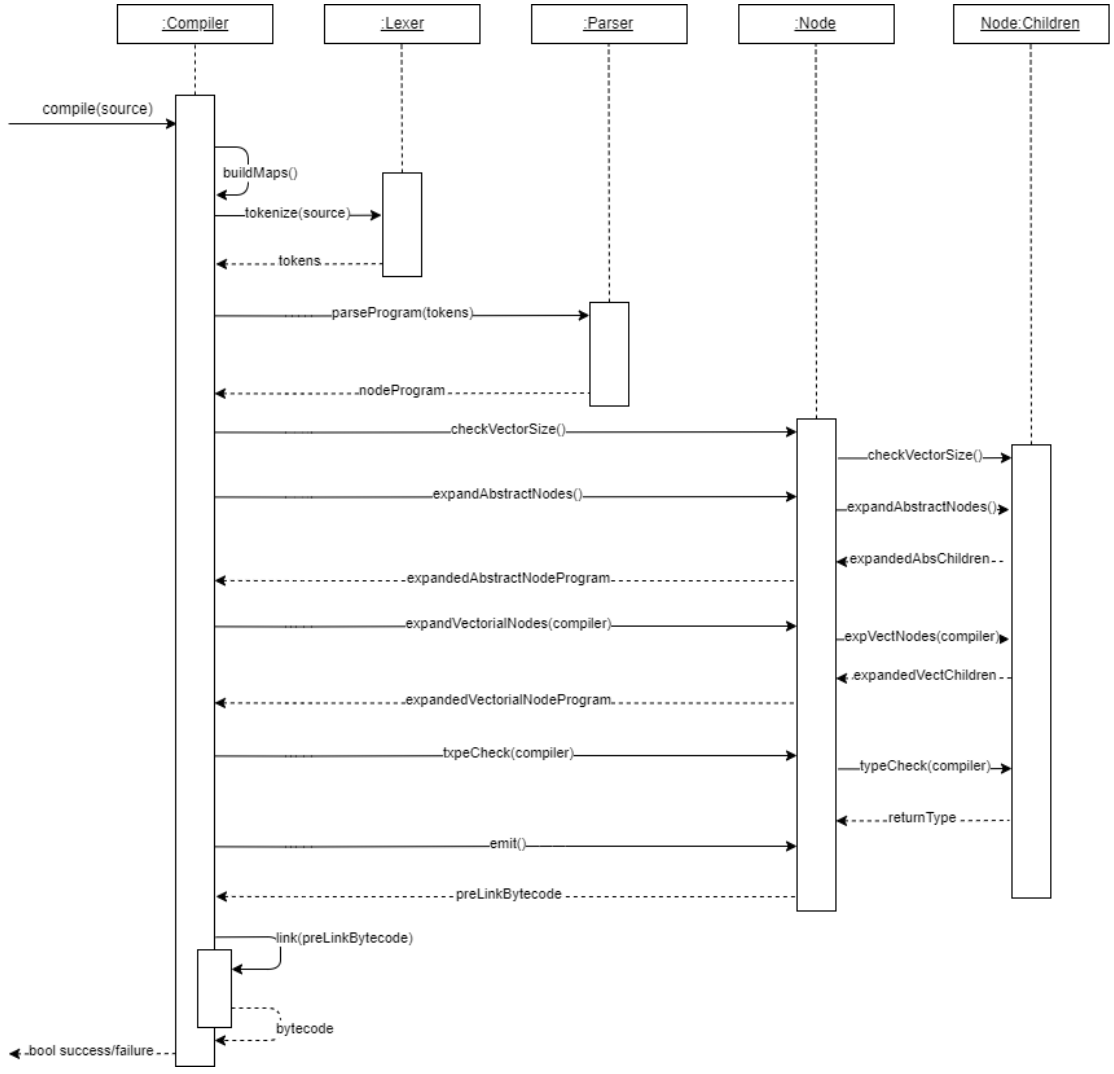
Unfortunately, it was later found that `three.js` has a built-in function which translates a `ThreeJS` Object into a `JSON` file element, and inversely. However, dthe solution developed is still being use.

6.4. Interpreter

The interpreter found that compile/translate .aesl 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 which means that an operation was needed in order 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. The decision was taken to translate the already existing one from `Aseba` and

to start with the compiler. After looking at its behavior and recognizing how it was built and separated, the time came to translate it to `JavaScript`. First, the translation creates multiple maps of variable, constant and events. These maps need to be rebuilt with each call to the compiler in case the previous ones produced an error. Afterward comes the tokenization of the source file, which consists in 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. This bytecode is then linked, creating the final program.

The following figure displays 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 is the size of the application. Being a large application, it was very hard to understand which component was linked to which, in what manners and how

they interacted with each other. A tremendous amount of time was required, scavenging the files and folders in order to try to get an understanding of the architecture. The second reason is the lack of comments in the source code. Considering the size of the application it was a significant difficulty. In addition to that, it was not planned to work with C++ and we were not ready for it. Our knowledge of this programming language is very basic. To summarize, we have a full-scale application coded in a language we are not familiar with and with sparse comments. Consequently, it was a very difficult part of the project and could not be completed.

6.4.1. Tokenize

To tokenize the source file it was needed to skim through the document and switch depending on the value of the character. There are basically five categories for this switch. The first one comprises the tokens requiring only one character to read, such as `)` or `,`. These can directly be given their type. The second category is made of the comments, that is to say the comment block,`#* ... *#`, or the simple line comment,`#`. In case of 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 the character is not found. The third category encompasses the cases that require one character look-ahead. An example is the character `+` which could either be 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, the numbers and the strings fit, 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. Below is the code for this regex.

```
isAlphaNumeric(ch) {
  return ch.match(/^[a-z0-9]+$/i) !== null;
}
```

One has to be careful about one point when using this method. If the `ch` parameter is not a string, then it will throw an error. Coming back to the switch, it first has to be tested if the chain of character is a number. This is done by combining regex and looking for multiple characters. If the chain of character is not a number, it has to be checked whether its value is the same as the one of the given keywords, such as `when` or `const`. In case none of the keywords matches, the chain of character is labeled as a string literal and it is given the value of the chain of characters. Here a problem of language between C++ and `JavaScript` was encountered, as the type of the token is given with an `enum` in C++ and they do not exist in `JavaScript`. Consequently, a workaround had to be found in order to solve this language incompatibility. The Object method `freeze()`, which prevents the modification of existing property attributes and the addition of new ones, was used and a value had to be given 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",
  ...
}
```

6.4.2. 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.

The `Parser` 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, the token is split into three categories, namely 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. Should this not be the case, the declaration of the constant is not valid and an error is thrown. Should the condition be met, the name of the constant is assigned to a named variable, and the position to a position variable. Afterward, the `Parser` controls that no constant with this name already exists. Then it controls that the next token is of type `TOKEN_ASSIGN` to respect how a constant is declared and gets the value that should be at the next token. If the next token does not belong to the type `TOKEN_INT_LITERAL`, it will return the value 0. This part will be improved by the completion of the translation of the tree. Finally, the constant with its name, position and value will be added to the `constantsMap` of the compiler.

If the token is a variable, it will be discarded and the following tokens will be discarded as well if they are part of its declaration. In fact, the part where it would parse the variable was replaced with this removal system. This was done because the `tree` and the `parser` are not complete. Here is how it should be, if the `tree` and the `parser` were finished.

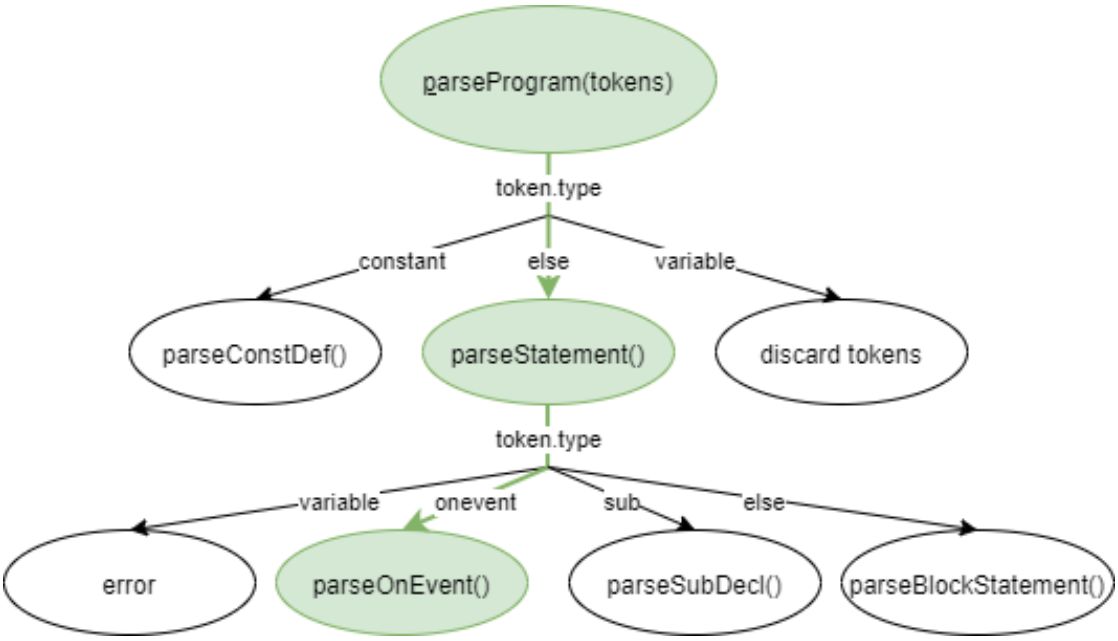
```
var child = this.parseVarDef();
if(child)
{
    programNode.children.addRear(child);
}
```

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 multiple methods based on a switch on its type until it reaches the parse method that defines it. Not every parse method was translated from the `C++` program as the amount of parse method is rather big and not easy to translate into `JavaScript`. Below is a diagram showing the process of a token of type `TOKEN_STR_onevent` to ease the understanding.



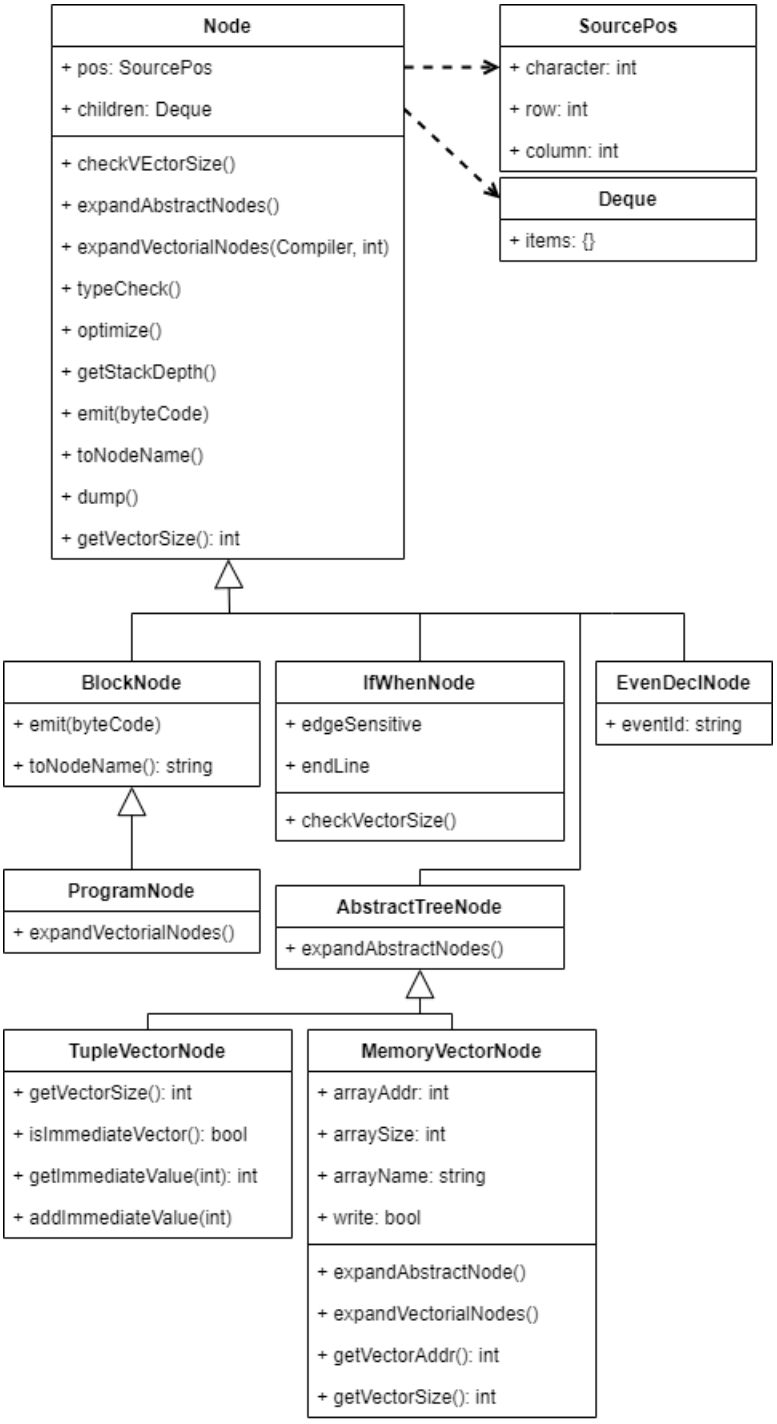
The end method `parseOnEvent()` controls two things. First it checks whether an event with this `Id` exists. Then it verifies that such an event has not been implemented already. The list of `eventId` can be found in the `Compiler.js` under the `buildMaps` method. If these two conditions are met, a new `Node.EventDeclNode` with the position and the id of this event will be returned and added to the `programNode`. In case the token is of type `TOKEN_STR_when`, the `Parser` 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 it can be observed, this method starts by calling another one. This will go on and on until very low methods such as the `parseUnaryExpression`.

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 the point that was reached, for more information on the possible completion and future work refer at the section Future research on page 38.

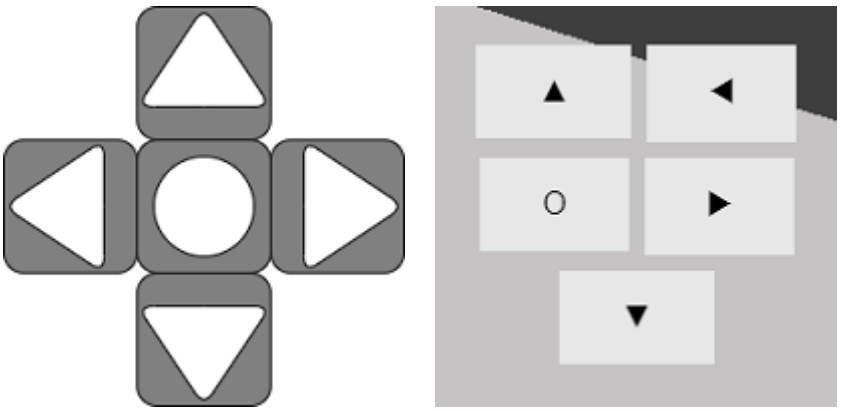
6.5. Phyisics

Adding physics allows collisions between the robot and the different objects. It's been tried to use `Physijs`, the source code can be found in the following repository Physijs git, but with the architecture of the application, a problem was encountered due to the fact that `THREE.Object3D` are added instead of basic mesh and `Physijs` works only with a fixed amount of meshes. A solution would be not to use `Physijs` as it is only needed for the `Thymio` to move and to stop upon collisions. Instead of `Physijs`, the robot could shoot rays in order to check for collisions.

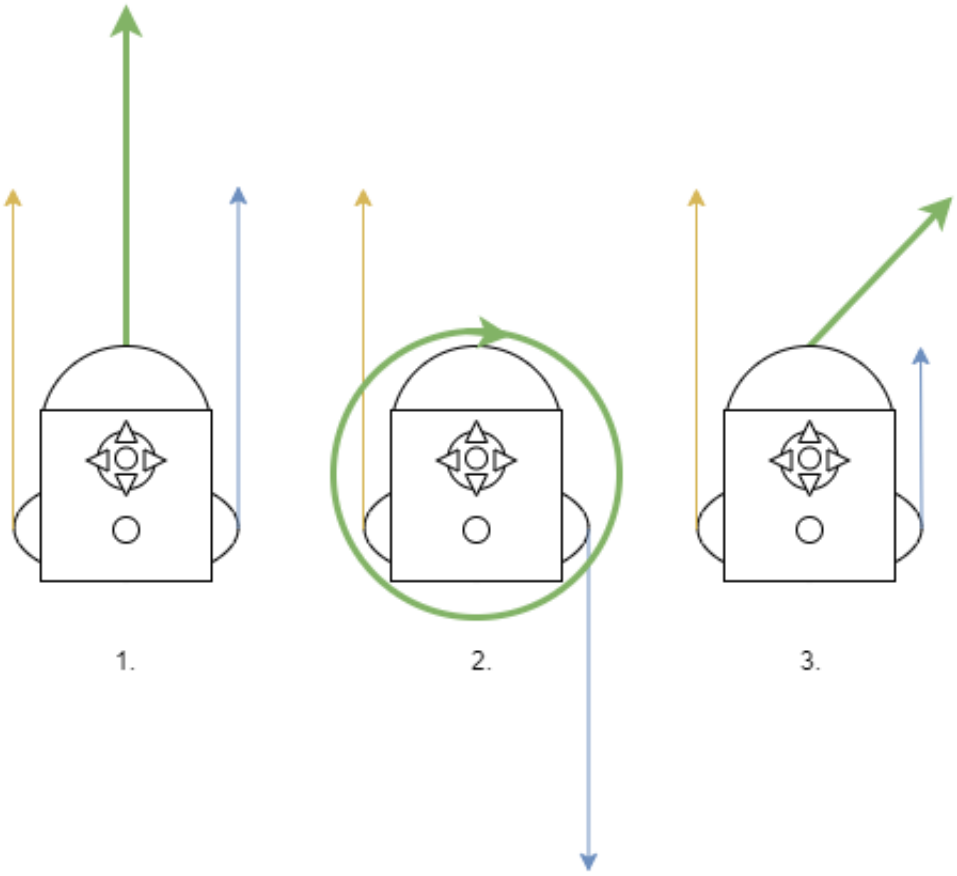
6.6. Sensor and Actuator

The five buttons that sit on top of the `Thymio` were represented directly in the `UI`. There was no interaction with the modelized ones. Initially, we wanted to create a directional pad style group of buttons similar to the one on the first image below. However, due to time constraints, the created group of buttons could not be perfectly arranged, as pictured on the right.

1 2



A `Thymio` robot moves according to 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, where the minus sign means that the motor powers the wheel in the opposite direction. Those two motors can only output power in a straight line, forward / backward. Below are displayed 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 is the same, both in value and direction. Thus, the final vector is simply the same as either one of the two. On the second representation, 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. Finally, the third representation shows the resultant vector when the value is not the same but the direction is.

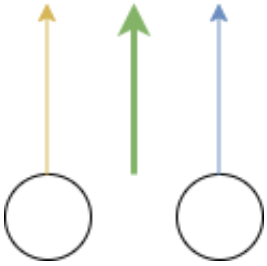


It was complex to convey the way the Thymio robots move onto this application as we move models and do not physically power motors. To achieve the same behavior, it was decided to resort to trigonometry and a few tricks. The method that moves the robot is called only within the `render()` method from the main View, and thus every time the scene is being rendered, the robot is moved according 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.

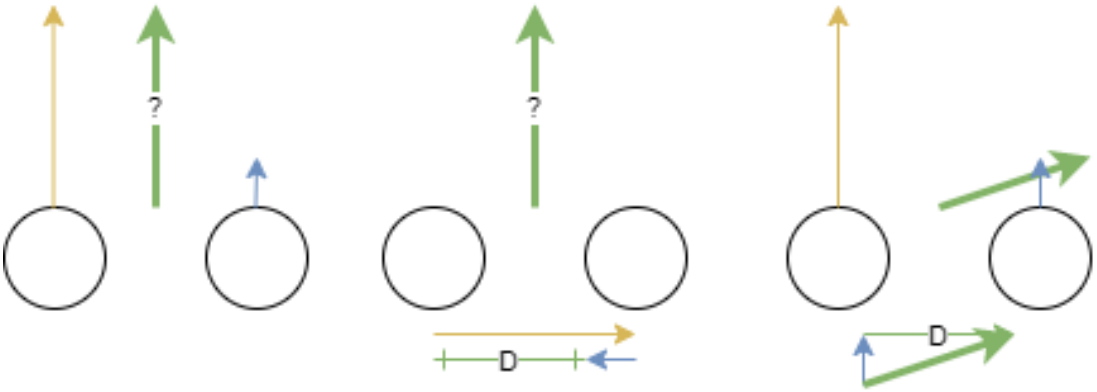
The first category is when both motors have the same power in the same direction. Thus, the resulting position change for both x and z coordinate is computed using the `getDirection()`, which returns a Vector3 representing the direction the robot is facing, a predefined speed variable and the power given 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 one of the right one, regarding the direction. Alternatively, the third option is the same but when the value of the right motor is bigger than the one of the left one.

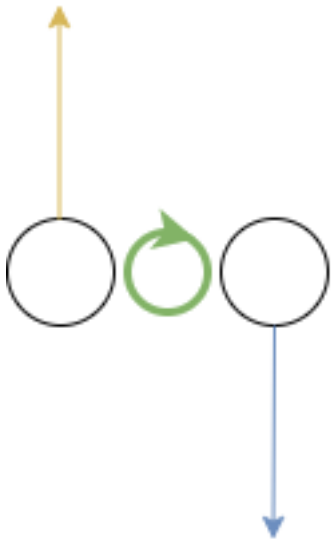


In this case, we calculate the difference `D` of power between the two motors, subtracting the smallest from the biggest. Then, trigonometry was used to find the angle between the power of the motor and the resulting vector of the addition of the power of the motor and the difference `D`. This angle is then used as a parameter to rotate the model around the y-axis. In order to smooth the rotation, 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.

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



However, this algorithm has a few flaws. One of these happens when the power of one motor is 0. While the robot should be rotating on itself with the said motor as an anchor point, it rotates and moves in the direction of the second motor's direction.

In order to determine whether a collision occurs between the robot and one of the elements of the environment, it was decided not to use an external physic library but rather to throw raycast from various positions in the thymio model. In this sense, we instantiated a `THREE.Raycaster` object and looked for 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, the property `mediator` is added to its model where the value is the current mediator. Thus, the method displayed below was used in order to skim through the array containing the shapes rendered in the scene and create a new array filled with intersection objects. Then, the array was loop through to control that the distance is bigger than the given one. If not it operates an action depending on the `className` of the intersected object. In the default case, we assume that the robot encounters an element against which he should stop moving.

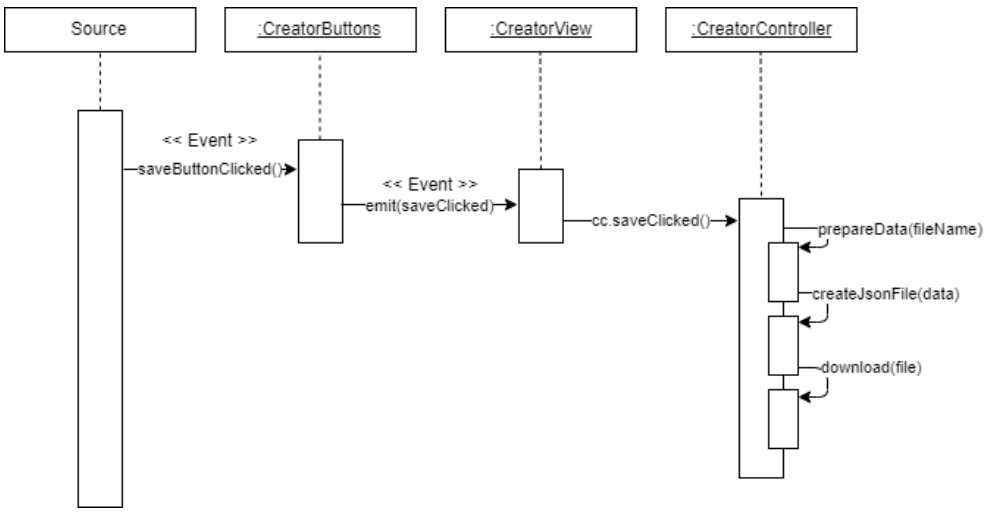
```
intersects = raycaster.intersectObjects(this.shapes, true);
for(let i = 0; i < intersects.length; i++){
  if (intersects[i].distance < 3.5){
    if(intersects[i].object.mediator.
       model.className === "Plane" ||
```

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

A variant of this algorithm was used in order to control the ground collision. Thus allowing the robot to know if it is progressing on the ground or not. If the robot is not progressing on the ground, the algorithm would make it fall by changing its `Y` coordinate. However, for some reasons the `intersectObjects` method would not detect any intersection if the robot was in the upper half part of the playground. Thus, the ground check were disabled.

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, the architecture is the same as for the simulation, but the `Controller` as well as the `View` elements are changed and the compilation part is excluded. 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, multiple options appear, such as using a database to store every customed playground so that they would all be available with the application. Another option would be to download them locally as a file on the user's computer. We continued 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. A sequence diagram of the process is shown below.



First, once the event that the `saveButton` has been pressed is thrown, 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 arrays based on their `className` property. Once the shapes contained in the customized playground have been gone through and categorized, the final `JSON` file is created. Its creation uses the `forEach` method of `JavaScript` array and writes the name of the mesh as well as its properties. Finally, the file is downloaded on the user's computer and the process ends. The `JSON` file obtained looks like as follow.

```
"playground":"jailtype",
  "octagons":[
      {
      "name":"ground",
      "props":{
          "segmentLength":35.5,
          "color":"#bdbbbb"
          },
      "hasWalls":true
      }
  ],
  "boxes":[
      {
      "name":"Box0.5",
      "props":{
          "width":4,
          "height":10,
          "depth":21,
          "color":"#ff0202",
          "positionX":0.5,
          "positionZ":-17.5,
          "rotateY":1.6755160819145565
          }
      },
```

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 the user chooses his desired location for the mesh, he needs to deactivate those controls with the `shift` key, and enabled them back with the same key. It was decided to add three other controls. The first is the use of the `Esc` key to cancel the current mesh positioning. This control removes the current placeholder from the scene. The second and third are linked with each other. It is 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 that the current shape being created by the user is from the type `Tracks`. Then, another logic is applied, 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 the `width`, the `length`, the `height`, the `color` and the `rotation` for the box. Then, by clicking the `Generate` button of the 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 );
```

Afterward, the user has multiple-choices. He can either click to fix the mesh onto the scene, or the properties did not 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. In order to lay Tracks, it was 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 be but they are added to an array that stores the position of those point. 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. It was necessary to delete the last point added as it was registering and instantiating a point when the user was clicking on `Generate Track`.

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

However, the algorithm tracking the movement of the mouse and changing the position of the temporary mesh is not optimized and will slow down the application drastically. Unfortunately, we do not know which part of the algorithm has to be optimized.

6.8. Incorporated Behavior

The `Thymio` robot has multiple pre-programmed behavior which can be chosen using the central button of the D-pad. Thus, three behavior were implemented in the simulation as well. First, a basic behavior. This behavior allows user to input power to the motors using the D-pad. Second, is an explorer behavior. In this state, the robot will act as an

explorer by moving forward and upon collision with an element of the playground, it will drive back, turn and drive forward again until it is stopped by the user. To achieve this sequence of instructions, a `JavaScipt` method that allows a delayed call to a function was used. It is the `setTimeout` method, which was used to stop the motors after a given time and start to turn.

```
this.setMotors(-250,-250);
var thm = this;
this.timeOuts.push(setTimeout(function(){
  thm.stopMotors();
  thm.halfTurn(true);
},1000));
```

The third behavior that was programmed, is a track follower one. Upon pressing the forward button, the robot will look for the closest point of the track and start to follow the track from that point. It was decided to reach the points from the track starting by the closest one, so the previous would be discarded. Although by uncommenting a line in the code this behavior can be achieved.

```
var smallest = 100;
var index;
for (let i = 0; i < this.points.length; i++) {
  if (this.object3D.position.distanceTo(this.points[i]) < smallest)
  {
    smallest = this.object3D.position.distanceTo(this.points[i]);
    index = i;
  }
}
```

In order to reach the next point the algorithm is not using the motors of the robots, this could be improved. To control whether the robot is close enough to the target point a method is called, in the `onFrameRendered` method only if the state corresponds to the track follower, and if the robot is following a track, which will change the behavior depending on the distance to the target and the number of points left.

7. Conclusion

7.1. Results

We have reached five out of the six goals that were determined at the beginning of the project. We have created an environment where a Thymio robot can move around a given playground. The playground can be chosen between built-in ones and others that are fully created by the user with a tool enabling it. One sensor and three actuators have been analyzed and implemented into the robot, the proximity sensors, the two motors, the lights, and the top d-pad buttons. We have also noticed that although our application architecture, MVC, allows the addition of new models very easily, it is too heavy and restrictive in terms of diversity. For example, to add the proximity sensors we had to implement them in a non-optimized manner. The application runs smoothly, except for the placement of new meshes in the playground creator due to an algorithm issue, on any modern browser without further installation required and is accessible from inside the BFH network. Unfortunately, it is not possible to input a .aesl file containing a program to the application and to use it as a behavior for Thymio. Thus, the sixth objective, the Integration of Aseba Language, was not brought to terms due to multiple problems and issues. As we came into this project without experience regarding the creation of a compiler/interpreter, we underestimated greatly the amount of work required by this part during the project. We decided to use an already existing compiler/interpreter, namely the one from Aseba used for their Thymio Suite application. The code was written in C++ and considering our basic knowledge of this programming language, it was a second slowdown in the project. Thus, it was rather difficult to convert this program into JavaScript. However, in order to compensate for this, we analyzed a Thymio robot and discovered that it is delivered with some built-in behaviors. Consequently, we decided to incorporate two of them to demonstrate that a web simulation of a Thymio is possible. In addition to that, if the steps in the next section are brought to completion it is completely possible to fulfill the integration of the Aseba language into the application.

7.2. Future research

In order to continue this project the next step would be to complete the goal that could not be brought to an end. This would require to finish translating the various .cpp and .h file from the folder compiler of Asebas Git. The further step of the process of compiling a file would be to implement the checkVectorSize method. This method recursively walks through the tree and checks the different Nodes vector size and throws an error in case the return value does not match the expected one. The compiler would

then continue on to expand the syntax tree to `Aseba` like syntax, through a recursive walk on the tree. The tree would be recreated with the new syntax. Afterward, it would expand the vectorial nodes into scalar operations and rebuild the tree. In the next step, the type of each node would be checked to see if it corresponds to the expected one. It would then optimize the current tree, and then emit the first `Bytecode` generated from the tree. Using a subroutine table, the compiler would fix-up the generated `Bytecode` that might be missing some `STOP` or `RET`. Then, the `verifyStackCalls` method, as its name suggests it, would verify the integrity of the stack with no overflow. And it would end by linking the bytecode into the final program. Here, the linking is a flattening of complex structure into linear vector.

Once this part would be over, more sensors could be added. Those sensors could, for example, be sound sensors to listen to the clap of hands, or a "Pen holder" to draw figures by moving Thymio around. Ideally, a way to ease the addition of sensors and actuators would be designed and implemented. It could be something similar to our `Model` / `ViewMediator`. And to improve the already existing one. For example by getting rid of the algorithm issue on the motor actuators.

7.3. Personal conclusion

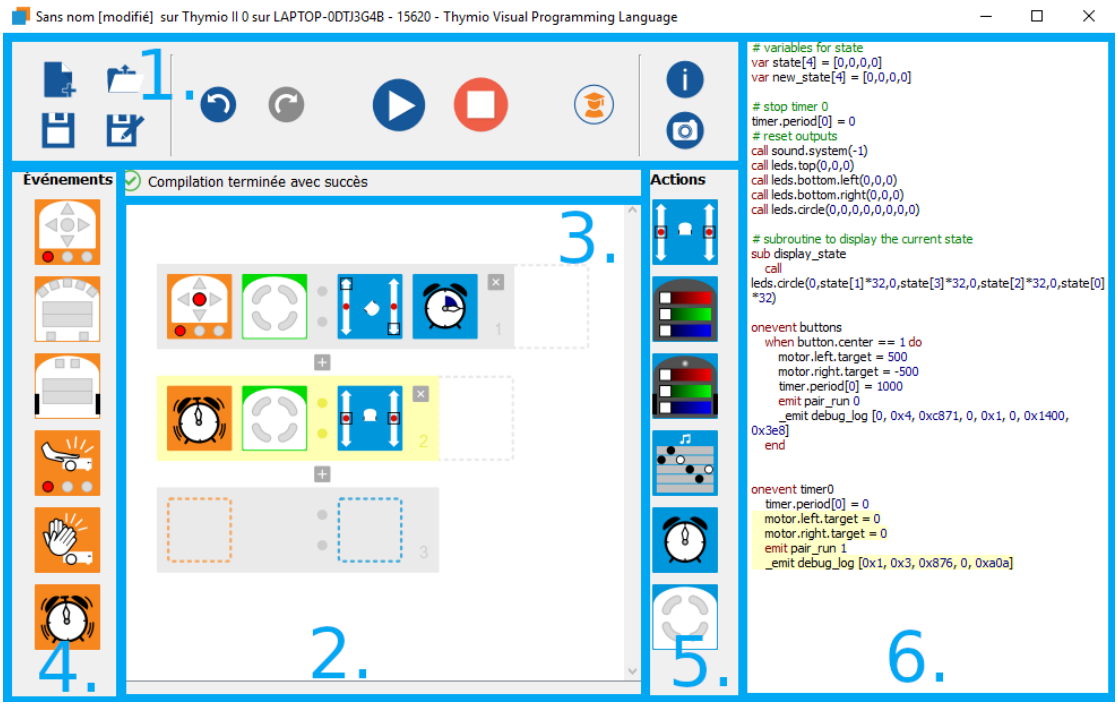
The work on this thesis was very interesting and challenging. There were many new technologies to learn, understand and master. Initially, we did not expect to be confronted with a full-scale `C++` application, but we have learned a lot from it. The difficulties that emerge when translating a program in one programming language into another programming language were also very instructive. We observed that it was rather difficult to estimate the amount of work needed for the various part that we were not accustomed to. For example, the compiler was very time-consuming, while the customizable playground required less time. In conclusion, we gained a lot of knowledge and experience during this project.

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 them. 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 which reduce drastically, if not eliminate, the syntactic errors made by the user.

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



Thymio VPL Event and Action blocks

To begin creating a program follow the first steps described in section 2.2 on 13. Once the VPL option has been chosen and the Thymio Visual Programming Language window appears programming begins. The window is split into six different regions, where each has its own purpose as listed below.

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.

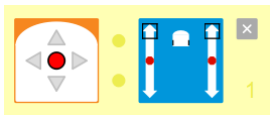
Play music.
Click on a bar to set a note.
White notes are longer than black notes.
Click on a note to change white ↔ black.
Click again to silence this note.

(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.

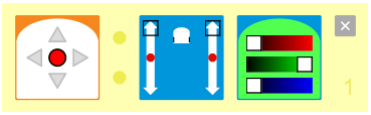
Thymio VPL Window

At first, the programming window is 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:



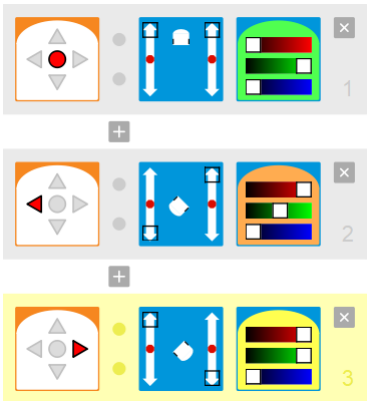
Event and one Action relation

Both wheels are powered to the maximum when the middle button is pressed. However, more than one action can be attributed to one event. In order to do that, another action block has to be dragged into the previous pair. It should be noted 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:



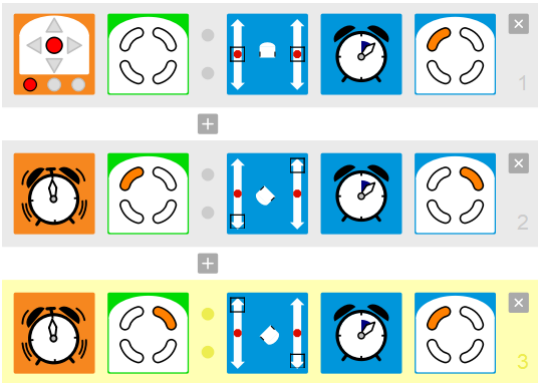
Event and Actions relations

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. In order to help us develop a more interesting program, we have now access to a condition, namely 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. However, it has to be decided which pair should

be executed by the program, namely 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 to program the `Thymio` is to use `Blockly4Thymio`, which is an environment based on Blockly. Blockly was released in May 2012 and was initially a replacement for OpenBlocks 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 it is 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, it shows similar advantages to the first possibility. For example, it can apply 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`. Second, it 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. Additionally, it has been translated into over forty languages, and there exist right-to-left versions.

Blockly includes a set of pre-defined blocks that can be used to develop more easily 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 operations, a few basic math functions (for example cos, sin, square root) and some mathematical constant (Pi).

Text: Blocks to create text and text operations.

Lists: Blocks to create lists and standard list operations (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 ones describe what happens during or what are the arguments, the 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

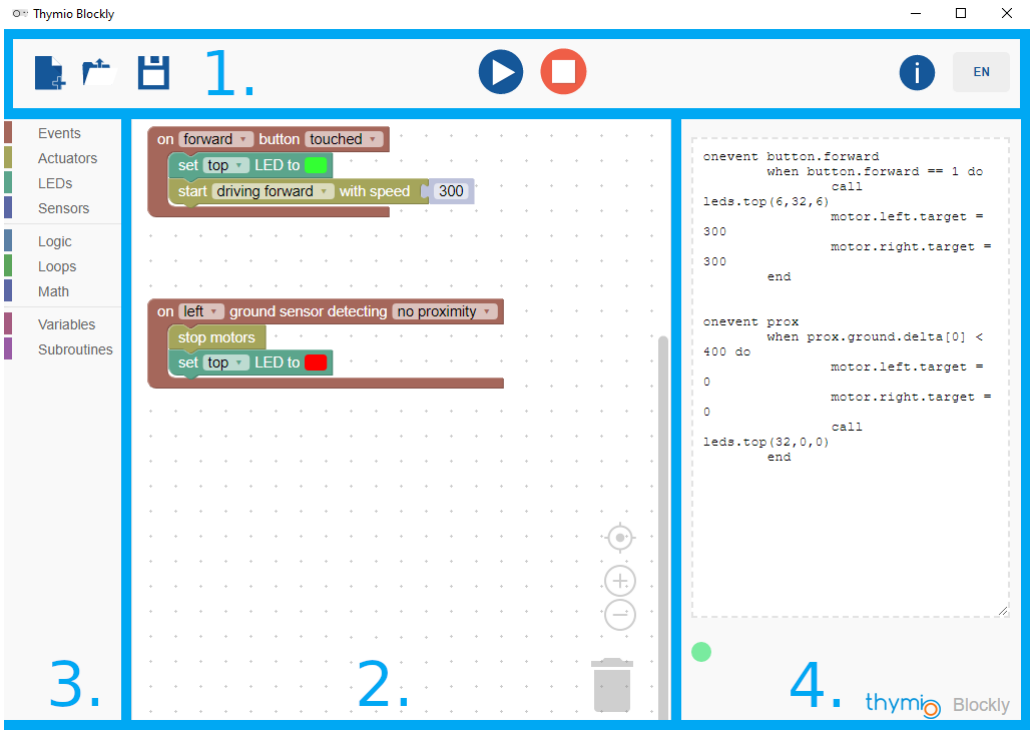
Using the same logic as above we created a `Limit` variable with the value of 5 to demonstrate the next example. The block used is from the logic family and it tests whether the `Count` variable is smaller or equal to the `Limit` as internal blocks. It can then be added to a loop, a function or to other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

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 small 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). In order to do so, open the location of Thymio, the downloaded and not the installed one, and select thymio_blockly, and then index. The environment window that opens after choosing the Blockly option is split into four parts.

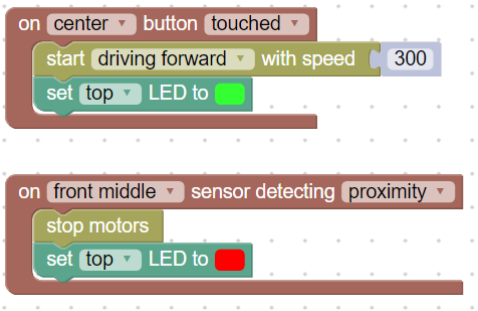
1. A tool bar
2. A programming window

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



Thymio Blockly window

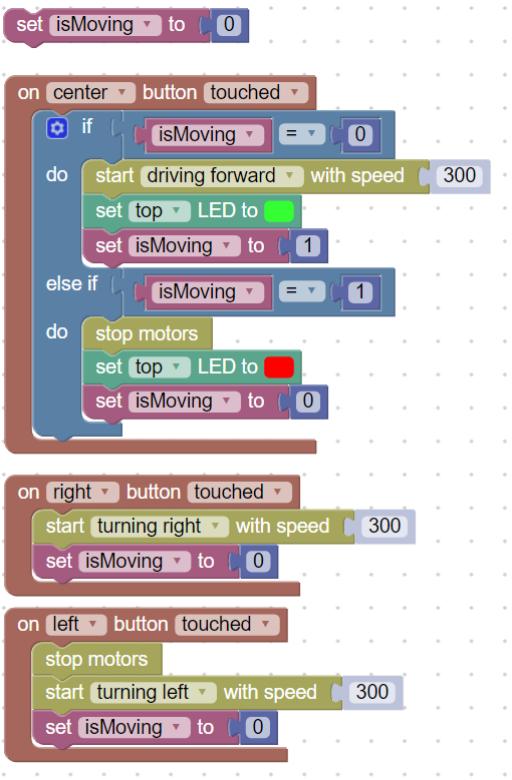
The following figure displays a simple program. Once run the program listens to two different events. When the center button is pressed and when the front middle proximity sensor detects a wall. The first one will activate the two motors at the same speed, as to drive forward, and light the top LED to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



Basic program

Here we set a variable to control whether the `Thymio` is moving or not. This information is then used into a test when clicking the middle button, and either moving forward

or stoping according to the result. Two other events were added 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 examples 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 examples 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	Done
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	76	76	Done
6	Architecture Implementation	Refactor the existing code into the designed architecture	High	40	48	48	Done
7	Web Deployement	Deploy the application on a webserver	High	16	12	12	Done
8	Basic UI	Implement a basic UI	Low	8	8	8	Done
9	Behavior Pipeline	Create pipeline to take .aesl file and translate/-compile it into behavior in JavaScript for the Thymio II	High	40	80	80	Done
10	Phyisics Implementation	Implementation of Collisions for threejs Meshes	High	20	16	16	Done
11	Update Documentation	Update existing documentation	High	20	20	20	Done

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
15	Enhanced Behavior Pipeline	Implement more sensors, action and event for Thymio II	High	/	/	/	To Do
16	Finish Documentation	Finish existing documentation	High	80	88	88	Done
17	Prepare Defense	Prepare the defense	High	24			To Do

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

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 defalut 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

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 Deployement	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 .aesl file and translate/-compile it into behavior in JavaScript for the Thymio II	High	40	80	80	Done

10	Phyisics Implementation	Implementation of Collisions for ThreeJS Meshes	High	20	0	/	To Do
11	Update Documentation	Update existing documentation	High	20	20	20	Done

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
10	Phyisics Implementation	Implementation of Collisions for ThreeJS Meshes	High	20	16	16	Done
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

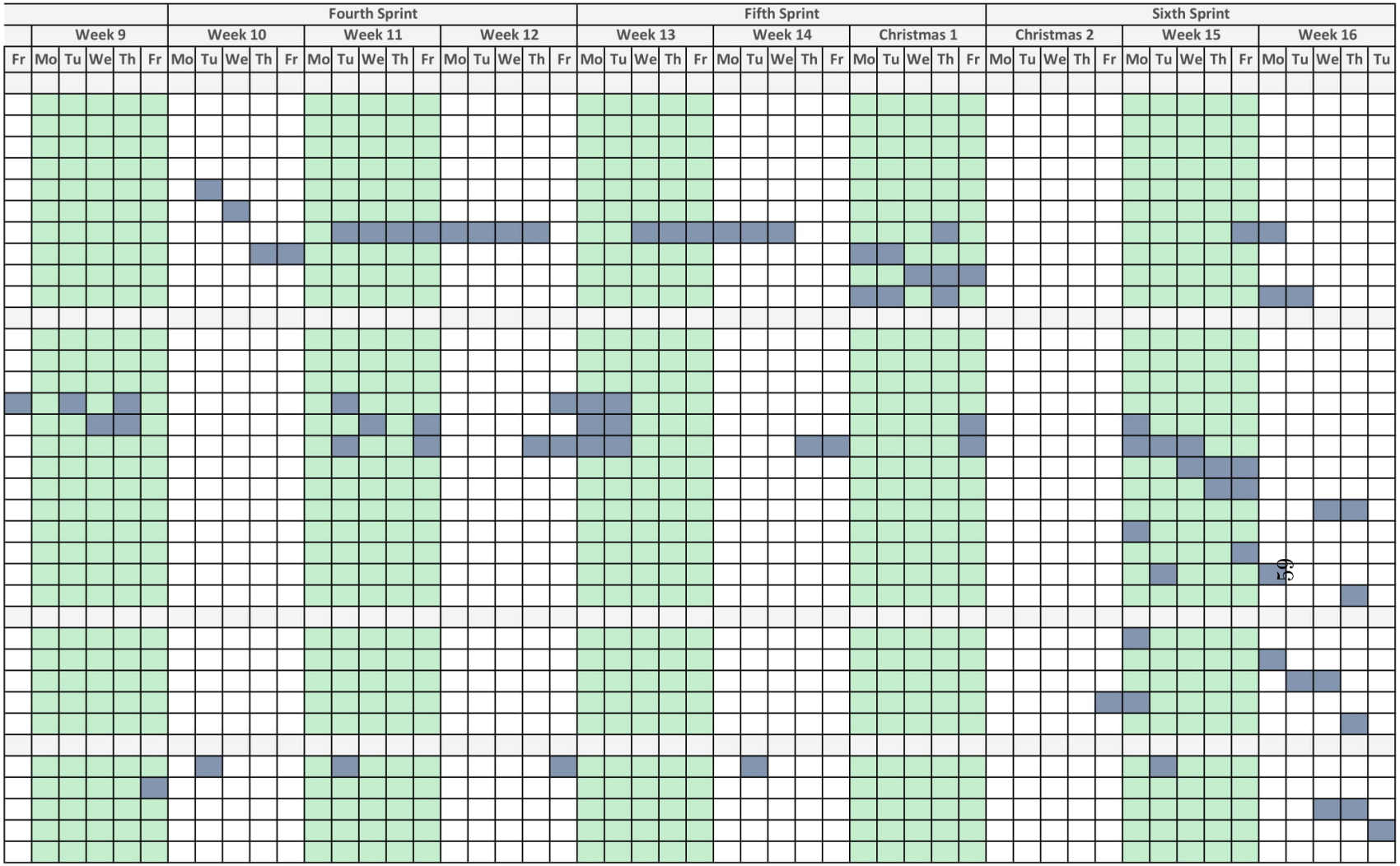
2019-12-30 until 2020-01-17

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	80	88	88	Done
16.1	Create Video	Create the video file	High	16	16	16	Done

16.2	Prepare Presentation Day	Create the poster and the presentation for the Presentation Day	High	16	16	16	Done
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	44	44	Done
16.5	Prepare to Submit	Check spelling mistake, check images, print it, put the project on a USB stick	High	8	8	8	Done
17	Prepare Defense	Prepare the defense	High	24			To Do

D. Gantt Diagram

Mission	Deadline	Duration (d)	First Sprint										Second Sprint										Third Sprint										
			Week 1					Week 2					Week 3					Week 4					Week 5					Week 6					Week 7					Week 8			
			Mo	Tu	We	Th	Fr	Mo	Tu	We	Th	Fr	Mo	Tu	We	Th	Fr	Mo	Tu	We	Th	Fr	Mo	Tu	We	Th	Fr	Mo	Tu	We	Th	Fr	Mo	Tu	We	Th	Fr	Mo	Tu	We	Th
Development		63																																					
Environment setup		3																																					
Basic Learning		7																																					
Playground development		10																																					
Architecture implementation		6																																					
Web Deployement		3																																					
Basic UI		2																																					
Behaviour Pipeline		20																																					
Physic implementation		4																																					
Customizable PG		3																																					
Sensor/Actuator		5																																					
Documentation		51																																					
Goal definition		2																																					
Template and Content		2																																					
About Thymio		10																																					
Requirements		8																																					
Architecture		8																																					
Approach		12																																					
Results		3																																					
Conclusion		2																																					
Review and corrections		2																																					
Book Creation		1																																					
Book Review		1																																					
Book Submissions	13.01.2020	x																																					
Doc. Submission	16.01.2020	x																																					
Video & Poster		4																																					
Poster Creation		1																																					
Video Creation		1																																					
Video Review		2																																					
Poster Submission	03.01.2020	x																																					
Video Submission	16.01.2020	x																																					
Presentations & Meetings																																							
Meetings with M. Fuhrer																																							
Meeting with Expert																																							
Presentation		2																																					
Final Day	17.01.2020	x																																					
Defense	28.01.2020 16.00																																						


Fourth Sprint
Fifth Sprint
Sixth Sprint
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Christmas 1
Christmas 2
Week 15
Week 16
Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Fr Mo Tu We Th Tu

E. Configuration

E.1. User Information

On demand.

E.2. 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 the link below 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`

E.3. 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`. This is where the default placeholder files was stored. There, we put them into a new default folder and instead added our content in this folder. It should not be forgotten to take the .webconfig file from the 6.2 section.

F. 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 risk 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, where the software only allows the user to create programs 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
19.11.2019	**Sixth meeting**
07.12.2019	**Seventh meeting**
19.12.2019	**Eighth meeting** - Documentation review - What to hand in to the secretariat and to Mr. Fuhrer
07.01.2020	**Ninth meeting** - Documentation state - Latex commands - Final day and defense talk
14.01.2020	**Tenth meeting** - Latex commands - Organizational questions

G. 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 was not able to load any .aesl 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 was not accessible -> decided to use html buttons instead
- c++ to javascript

Erklärung der Diplomandinnen und Diplomanden
Déclaration des diplômant-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.