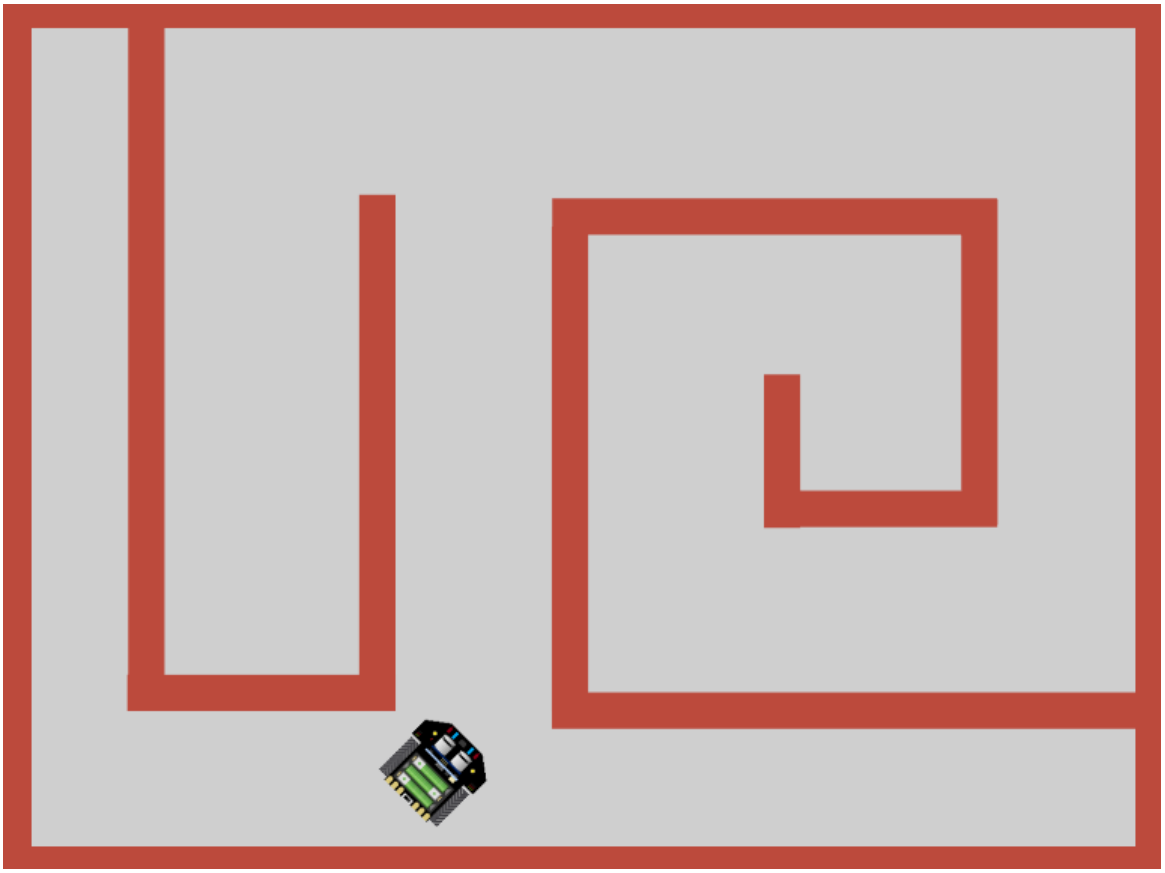
ETH zürich

Department of Computer Science
Algorithms and Didactics



2D Robotics Simulator for WebTigerPython

Kai Zürcher

Bachelor's Thesis

2025

Supervisors: Prof. Dr. Dennis Komm
Alexandra Maximova
Clemens Bachmann

Abstract

This thesis adds a microcomputer as well as a robotics simulator to WebTigerPython, a web-based integrated development environment (IDE) used for teaching Python programming, often in combination with microcomputers and robotic extensions. The simulator is not meant to replace the physical hardware, rather, it aims to reduce limitations such as hardware unavailability, compatibility issues, and testing delays by providing a virtual testing environment.

This thesis integrated existing simulators for the micro:bit and the Calliope mini 3 into WebTigerPython. Then, a robotics simulator was built upon these microcomputer simulators using Phaser 3. Unfortunately, the pre-existing simulators had no pin simulation, which is essential for the robotics simulation. Thus, a workaround was devised to bridge the missing pin simulation by parsing the code before execution.

The simulator models realistic movement and sensors for the Maqueen Lite, Maqueen Plus V2, and the Calli:bot. To enable meaningful sensor simulation, an interactive environment editor was also added to the robotics simulator.

Acknowledgment

I would like to thank everyone who helped me realize this thesis, which, without their help, would not have been possible. Prof. Dr. Dennis Komm for giving me the opportunity to write this thesis in his group. My two supervisors, Alexandra Maximova and Clemens Bachmann, both answered all of my many questions in our weekly meetings and provided me with useful guidance and great inputs during the development, as well as assisting me with the administrative requirements to realize this thesis. Cédric Donner and Noe Schaller for sharing their assets of the Maqueen Lite and Maqueen Plus V2. Gianluca Danieletto [4], who provided me with helpful data on the Maqueen robots. Jeannine Marty for giving me insightful feedback for the simulator. Alexandra Maximova and Leo Strelen for proofreading and providing feedback on earlier drafts of this thesis.

Contents

Chapter 1

Introduction

The Center for Informatics Education of ETH Zurich (ABZ) [1] supports schools and teachers in teaching computer science concepts in primary and secondary schools. They develop various educational materials and educational software. One of their current projects is WebTigerPython [2], a web IDE used to teach Python programming with support for various libraries and programmable microcomputers and their robotic extensions. This thesis aims to add a simulator for these programmable microcomputers and some of their robotic extensions, focusing on driving robots.

1.1 Motivation

The simulator is not meant to replace the physical microcomputers or their robotic extensions, as working with physical hardware is very rewarding and beneficial for students. The simulator is primarily developed to be used in addition to the physical components and aims to reduce some of the rather frustrating parts of working with the physical hardware or to be used if the hardware is broken or not available.

Faster feedback

WebTigerPython is often used by beginner programmers, who tend to code by making small changes in their code and running it to see if the changes have their intended effect. To do this with the physical hardware can require a lot of steps. To test their code on a driving robot like the Maqueen Lite [6], students need to first write the code, then plug in the micro:bit [7], then flash the code to the micro:bit, unplug the micro:bit, and then turn on the robot. All these steps take quite some time and can lead to frustration after several failed attempts. With the simulator, the students only need to write their code and then can directly test it in the simulation, which can save a lot of tedious steps for small code changes.

Teachers often build some challenges for the students to solve, such as mazes. However, since space inside classrooms is rather limited, they often resort to building some of them in the hallway or other rooms. This then adds even more time to test the code, as students have to walk to the challenge. This time is again heavily exaggerated if there is a bigger group of students, as only one robot at a time can go through the same physical maze. Thus, the students have to wait until they can take their turn.

Hardware failures

While working with students and robotics components, malfunctions and broken parts are almost unavoidable. Robots get dropped, batteries run out, hairs get stuck in the wheels, and many other random hardware failures happen. Most of them can easily be fixed or replaced by teachers, but not all students can be helped at once. So, students often have to wait until a teacher can help them and can't continue to test their code until their robot is fixed. Here, the simulator can be a huge improvement as they can continue coding in the simulator until their hardware is fixed.

Incompatible hardware

Another problem that can occur is that students have devices, often tablets, that don't have the right ports to flash the code to the microcomputers. With the simulator, students can at least try their code in the simulation. While still not optimal, this is better than not being able to run code at all.

Not enough/no hardware

The simulator can also be helpful if the teachers don't have enough robots for every student. Students could share robots, run most of their code in the simulation, and then take turns running their final code on the physical robot.

It also enables students to work on their code at home, even if they don't have a robot available.

Demonstration tool

The simulator is very helpful in demonstrating code to the entire class through the projector or for taking screenshots of it to put on worksheets.

Comparable environment

One downside of working with physical hardware is that every robot behaves slightly differently, depending on the state of the hardware. This often leads to some frustration, especially when teachers give students challenges and the same code works for one but not for another robot. The simulation enables a fair and reproducible playing field for challenges for all students by giving them the same environment and a consistently behaving robot.

Missing feature from predecessor TigerJython

The non-web-based predecessor TigerJython [12] already has such a simulator, which was often used and appreciated by students and teachers for the reasons mentioned above and thus has been heavily requested to be added to WebTigerPython.

1.2 Related Work

This section will give an overview of the IDE WebTigerPython, the microcomputers, the driving robots, and the API used in this thesis to realize the simulation for both the micro:bit and the robotics simulation.

1.2.1 WebTigerPython

Since 2010, TigerJython [12] has been used by the ABZ team as an installable IDE to learn and teach coding. It supported all common platforms at the time but is not usable on the now more common tablets, which run on IOS or Android. Thus, in 2018, WebTigerJython [17] was developed, which is a web-based IDE. Due to it being web-based, it is runnable on all platforms. However, some features of TigerJython had to be dropped due to limitations in web development at that time. For example, it was not possible to flash the microcomputers. Because of these limitations, Clemens Bachmann reevaluated the current state of web applications and developed WebTigerPython from the ground up to support all features of TigerJython as his master thesis [2]. Most features of TigerJython are already implemented and in use by the ABZ team. This thesis will add one of the missing components to WebTigerPython, the microcomputer and robotics simulator.

1.2.2 Microcomputers

Microcomputers are often used to teach programming as they are quite affordable for schools and easy and rewarding to work with. There exist many different microcomputers. The ABZ team primarily uses the micro:bit [7] and the Calliope mini 3 [15]. Both of them can be used by students and teachers to write Python code in WebTigerPython and then flash the code to the microcomputers and see it being executed directly. These microcomputers come with many sensors for measuring temperature, acceleration, and other factors. They also have a few buttons and a small 5×5 LED screen. By themselves, they can already be used for experiments to measure data or to program small games. However, they can do even more when used with accessories, such as driving robots or LED rings. The accessories are connected and controlled through the microcomputer's programmable pins. Thus, as we want to simulate the robots for this thesis, we also need to simulate the execution of the corresponding microcomputers.

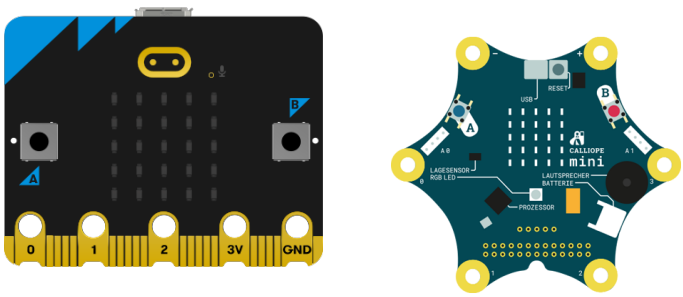


Figure 1.1. Graphic of the micro:bit (left) and the Calliope mini 3 (right).

1.2.3 Pre-existing simulators

As mentioned in Subsection 1.2.1, TigerJython had a simulator for both the microcomputer and their driving robots. However, the simulator is not easily portable to WebTigerPython as it uses Jython, which is no longer the case for WebTigerPython. Thus, a new simulator had to be developed for WebTigerPython. This was already attempted once before by Noe Schaller [16] in his Matura project. While his simulator is very impressive for a Matura project, it was unfortunately not usable for WebTigerPython. Thus, we decided to develop a new simulator for this thesis.

There are several pre-existing simulators for only the microcomputers. One of them is the micro:bit simulator [14] developed by the Micro:bit Educational Foundation and contributors. Another one is the Calliope mini 3 simulator [3], which is an enhanced and adjusted version of the micro:bit simulator, which was developed by Calliope and Lulububu. These two simulators are quite advanced and realistic. However, they lack the simulation of the robotic extensions. Thus, we decided to integrate these simulators into WebTigerPython and build a robotics simulator upon these, as creating such a microcomputer simulator from scratch would heavily exceed the scope of this bachelor thesis.

1.2.4 Robotics

Robotics is an essential part of teaching coding in schools, as seeing your code being performed in the physical world is very rewarding. Thus, the ABZ team frequently works with robots. In particular, the Maqueen Lite [6], the Maqueen Plus V2 [5] and the Calli:bot [11] are being used in combination with WebTigerPython. The two Maqueens can be programmed with the micro:bit, and the Calli:bot is programmable with the Calliope mini. All these robots have two controllable wheels, LEDs, infrared sensors, and an ultrasonic sensor. These can all be used to program the robot to solve a maze, follow a line on the floor, or avoid driving into walls.

To simulate robotics in WebTigerPython, a graphical environment is required in which the robot can be seen driving around. This environment must be accurately simulated while remaining lightweight so it can also run on tablets. The simulator should be easily integrable into WebTigerPython and match the aesthetic. It should also be easy to use for teachers and students. To allow for meaningful simulation of the sensors, a customizable environment with different types of objects for the robot to interact with is needed.

Figure 1.2. Pictures of the Maqueen Lite (left), the Maqueen Plus V2 (middle) and the Calli:bot (right).

1.2.5 Phaser 3

To realize the robotics simulation, a graphical representation was needed. Other parts of WebTigerPython, like the turtle visualization, are based on PixiJS [9], a flexible 2D WebGL renderer. For these purposes, PixiJS is a good choice as it is lightweight and easy to implement. At first, we planned to also implement the robotics simulator with PixiJS. However, we also wanted to implement interactions between objects in our simulator, such as collisions. We noticed that PixiJS as a framework is not ideal for these features. We then decided to use the Phaser 3 API [10] instead, a Javascript game framework. It is an extensive 2D game engine while remaining fairly lightweight. As a game engine, it already provides all the necessary tools for the robotics simulator, such as realistic physics simulation and collision detection.

1.3 Main contribution

The main contribution of this thesis is integrating the pre-existing microcomputer simulators into WebTigerPython and building a robotics simulator upon these, as described in Section 3.1. Due to the pins not being simulated in the pre-existing simulators, a considerable amount of work was put into devising a workaround, as described in Subsection 3.1.3. A significant amount of time was also spent on simulating the robots and most of their commands, as described in Section 2.2. Finally, to enable meaningful simulation of the sensors, an environment editor was also designed and implemented, as described in Section 2.3.

Chapter 2

Features

This section gives an overview of all the features of the newly added microcomputer simulator and the robotics simulator.

To open the simulator, the device setting has to be changed to the corresponding microcomputer, and the simulator will appear in the upper right section. In this section on the left, the microcomputer simulator is shown, and next to it on the right is the robotics simulator. If only the microcomputer simulator is needed, the "hide robotics simulator" button can be pressed, and the robotics simulator will disappear.

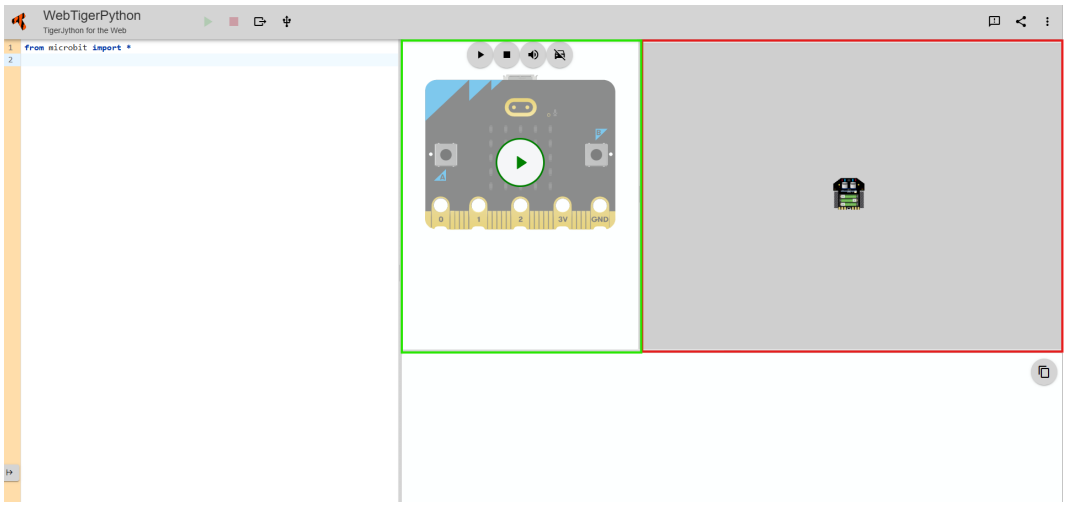


Figure 2.1. WebTigerPython with the device set to micro:bit and the micro:bit simulator (highlighted in green) and robotics simulator (highlighted in red) visible.

2.1 Microcomputer simulator

The microcomputer on the left updates based on your chosen device. Selecting micro:bit will display the micro:bit simulator, while choosing Calliope mini 1/2 or Calliope mini 3 will bring up the Calliope mini 3 simulator. Unfortunately, no simulators for the Calliope mini 1/2 are available. However, the Calliope mini 1/2 code can be run on the Calliope mini 3 simulator, but it has the simulated hardware limitations of the Calliope mini 3.

2.1.1 Visual interface

The simulator will show a graphic of the corresponding microcomputer, either the micro:bit or the Calliope mini 3 interface. The 5×5 display of either is simulated and will display the currently running code. An example can be found in Figure 2.2. Each simulator also has functional A and B buttons that can be interacted with by pressing their visual representation. For the micro:bit simulator, the touch logo is also functional. Unfortunately, the touch logo of the Calliope mini 3 is not interactive in the visual interface.

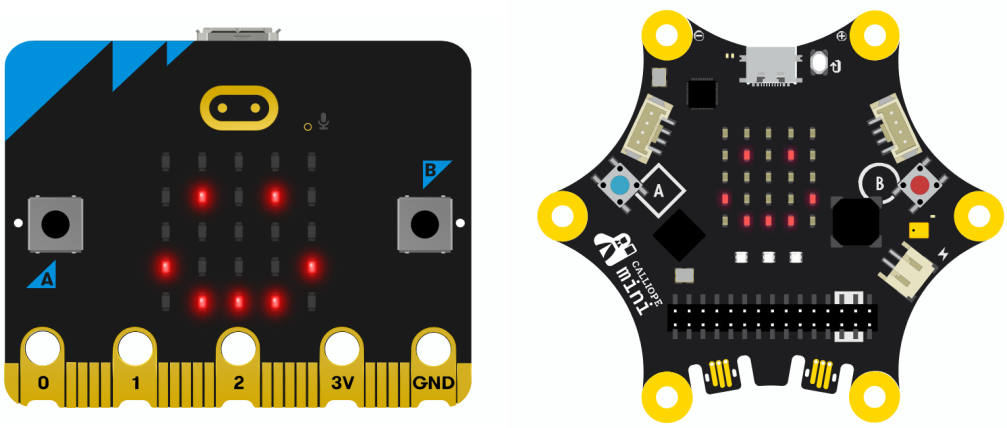


Figure 2.2. Screenshots of the simulated screens of the command `display.show(Image.HAPPY)` on the micro:bit (left) and Calliope mini 3 (right).

2.1.2 Console output

All output made by the simulator is printed in the usual console, this includes print statements and error messages. The error messages are especially helpful as reading the error messages on the microcomputer screens is very tedious.

2.1.3 Sound

Sounds are also played by the simulator, thus, music commands are functional and can be tested in the simulator. While programming with sounds is fun, an entire class of students making beep sounds is not. To address this, a mute button was added above the simulator to quickly and easily disable sounds.

2.2 Robotics simulator

The robotics simulator currently supports simulation for the Maqueen Lite, the Maqueen Plus V2 and the Calli:bot. The robot sprite automatically adjusts by running the simulator according to the imported library. The visual representation, as shown in Figure 2.3, as well as the shape of the robot, the amount and placement of sensors and LEDs are adjusted to enable a more immersive and realistic simulation.

A plain black sprite of the Maqueen Lite and Maqueen Plus V2 was taken from Noe Schaller's matura project [16], where he also developed a robotics simulator. More visual details were added to the sprites to make them more recognizable. The sprite for the Calli:bot is an adjusted version of the Maqueen Lite sprite.

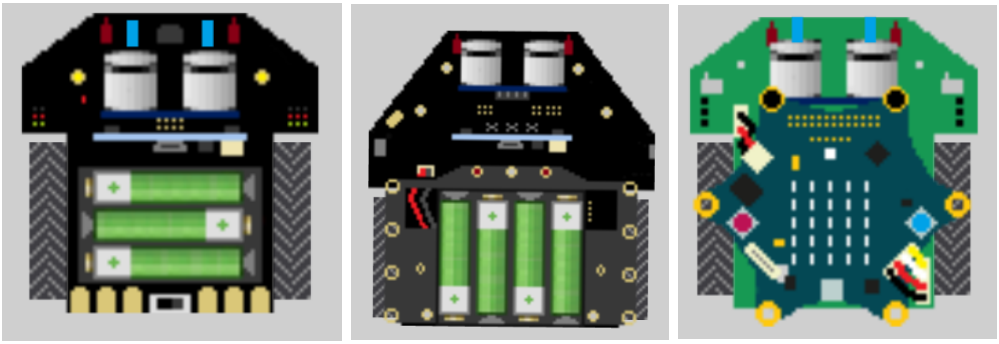


Figure 2.3. Screenshots of the visual representation of the Maqueen Lite (left), the Maqueen Plus V2 (middle) and the Calli:bot (right) in the simulator.

Simulated comands

Currently, only commands from the supported libraries `mbrobot`, `mbrobot_plusV2`, and `callibot` are simulated. All other robotics libraries are currently unsupported and can be run on the microcomputer but will not be simulated in the robotics simulator. However, from these 3 libraries, almost all commands are simulated. A complete list of all simulated and the not-yet-simulated commands can be found in Table 2.1.

Movement commands

All movement commands from the supported libraries are simulated, and the wheels are animated to spin in the corresponding direction. Also, commands to adjust movement such as `setSpeed(x)` and `calibrate(o,d,a)` are simulated. While the command `calibrate(o,d,a)`, used to calibrate the robot based on its physical condition, is not necessary as the simulated robot does have no hardware deficiencies, it still can be used to see the effects.

Sensor commands

The ultrasonic sensors are also simulated and return the distance to the closest object in a cone in front of the robot. However, this is not a fully realistic simulation, as the actual robots can only detect objects accurately up to a certain angle.

The infrared sensors are all also functional and detect if they are over a light or dark surface and return the corresponding value. For digital reads, the simulator returns 1 for light and 0 for dark surfaces. For analogue reads, only supported on the Maqueen Plus V2, the simulator returns an average of the expected value of 230 for a light surface and 100 for a dark surface, these values are based on the work of Gianluca Danieletto [4]. On the Maquenn Lite and the Calli:bot, the two small blue LEDs that glow when the corresponding infrared sensor is over a bright surface are also simulated. These are visible in Figure 2.3. The "pen" tool described in section 2.3 can be used to add a dark surface to the simulator.

The touch sensors, only supported by the Calli:bot, are also functional in the simulator. When any of the three touch sensor commands is used in the code, the Calli:bot sprite will show a bumper, similar to the one used with the actual robot, as seen in Figure 2.4.

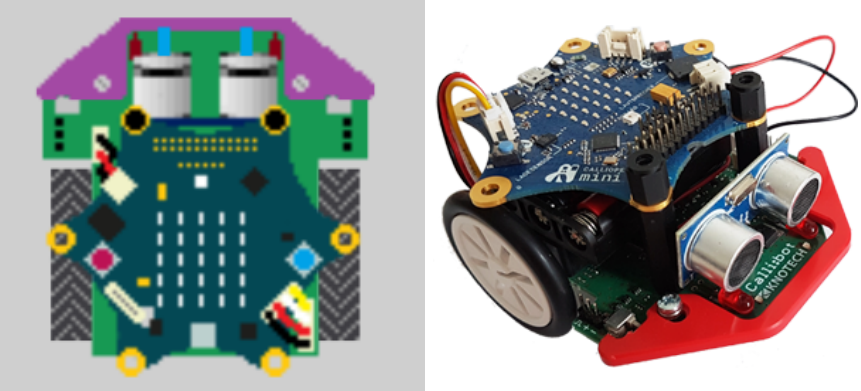


Figure 2.4. Comparison of the visual representation of the Calli:bot with bumper (left) and its physical equivalence (right).

Signalling commands

All robots also have two red LEDs in front of the robot, which can also be controlled. These are also simulated by changing their visual representation as shown in Figure 2.5.

Figure 2.5. Screenshot of the simulated Maqueen Lite with the left LED turned off and the right LED turned on.

The four under-glow LEDs present on the Maqueen Lite and Maqueen Plus V2 are visually represented in the simulator by coloured transparent sprites. Each LED can be set to an individual colour using the RGB colour schema. The lower the RGB values are, the more transparent the sprite is to simulate brightness, as shown in Figure 2.6.

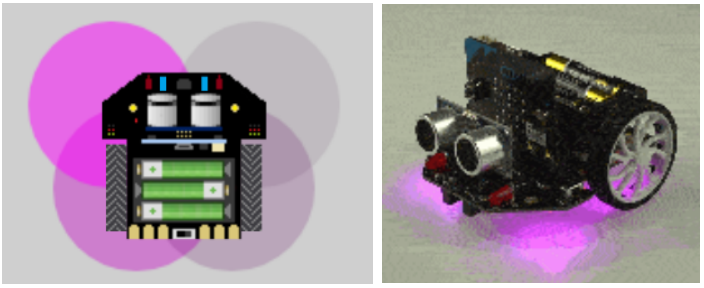


Figure 2.6. Screenshot of the visual representation of the under-glow LEDs on the Maqueen Lite with the four LEDs set to the same colour at different brightness levels (left) and a picture of the real Maqueen Lite with the under-glow LED's turned on (right).

All sound signals are functional as well and are directly played through the microcomputer simulator.

2.3 Environment editor

While movement and signalling commands can be programmed and meaningfully simulated in an empty environment, this is not the case for the infrared and ultrasonic sensors, as they need surfaces and objects to interact with. For this reason, the option to edit and interact with the robotics simulation environment was added through a variety of different tools to change the way you interact with the environment. A visual preview is displayed in Figure 2.7. This section presents details of the functionality of all the tools below.

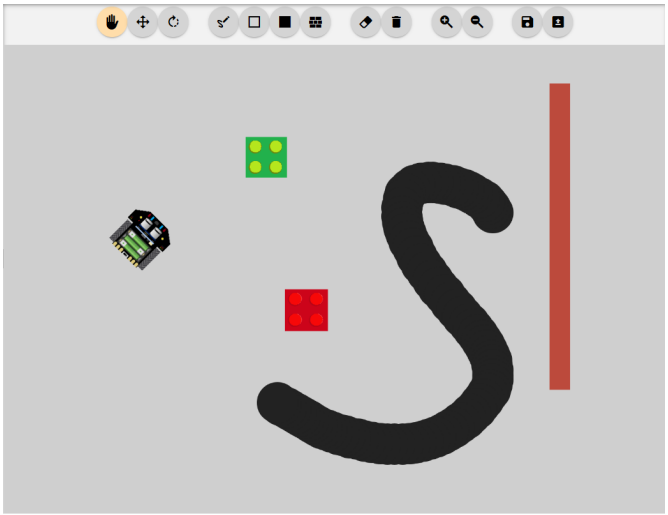


Figure 2.7. Screenshot of the robotics simulator with the UI of the environment editor and all different objects added to the environment.

Hand

The “hand” tool lets you grab and drop the robot as well as the movable blocks. This tool was added to enable the option to interact with the simulation while it is running and make small adjustments as it is done with the physical robots. For example, if a robot drives into a wall, students tend to pick it up and put it where it can drive freely again.

To enable movement and rotation with just one touch interaction, and thus ensuring full functionality with the mouse and touchscreens, the hand tool rotates the objects in the direction it was dragged in.

After restarting the simulator, all objects moved by the “hand” tool are returned to their original location. To make lasting changes in the environment, use the “move” tool described below.

Moving and rotating objects

The “move” tool enables you to make lasting changes to the simulator, such as moving the start location of the robot or other objects in the environment by grabbing them and placing them in the new location. With the “rotation” tool, you can rotate the robot’s start position and objects by 45 degrees to the right. This rotation is permanent in the simulator and will be the new default rotation for the object.

Changing surface

The “pen” tool is used to draw dark lines in the environment, mainly used to interact with the infrared sensors. To draw a line, simply press the left mouse button or press down with your finger and move it where you want to draw the line. An example of an environment created with the “pen” tool can be seen in Figure 2.8.

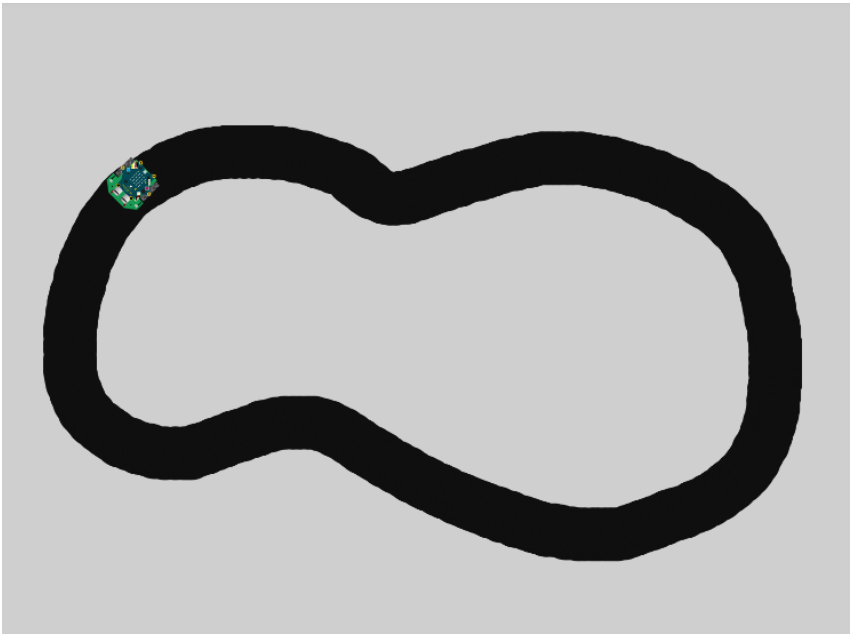


Figure 2.8. Screenshot of the Calli:bot following a dark line created with the environment editor.

Adding objects

There are three different tools to add objects to your environment, each adding objects with different properties. They can be selected depending on the desired interaction with the robot.

The “add movable block” tool adds a green block at the selected position that can be pushed by the robot and grabbed by the “hand” tool. This block can be used if you want to create an environment where the intended task of the robot is to push the object into a target area.

The “add unmovable block” tool adds a red block that has a fixed position that can not be pushed by the robot or grabbed by the “hand” tool. This block could be used to create an obstacle parkour where the robot has to drive around the blocks.

The third tool is the “build wall” tool. By selecting a start position and dragging to the wanted end position, this tool will draw a wall between the two points. To keep the walls aligned, the tool rounds the angle to the nearest 45 degrees of the drawn line. The wall is also locked in position and can not be moved by the robot or hand tool. This tool was added to enable the building of more complex environments like labyrinths, which are often used with physical robots, as this would be very tedious with just the block tools.

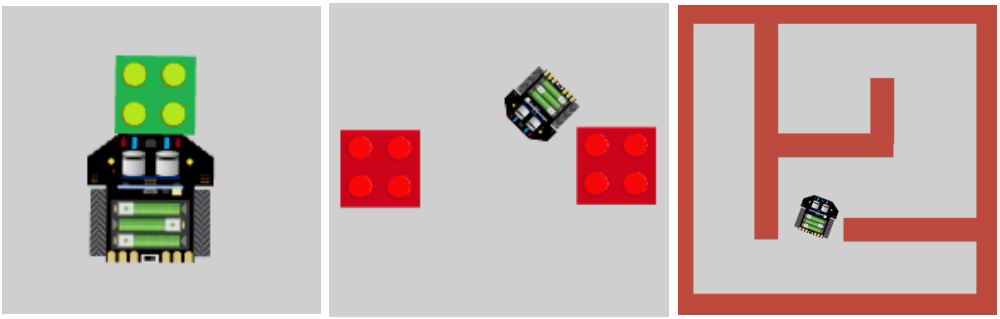


Figure 2.9. Screenshots of the Maqueen Lite in different envisionments build with different objects: movable block (left), unmovable blocks (middle) and walls (right).

Removing objects

If you can add objects to an environment, you also need to be able to remove them again. For this purpose, we have the “eraser” tool to remove individual objects or parts of lines, and to remove all objects and lines at once, the “remove all” button can be pressed.

Changing robot size

Depending on the program, a different size of the robot might be more optimal. For example, for a big labyrinth, a smaller robot is more practical. However, to demonstrate the functionality of the LEDs, a bigger robot is better suited. Thus, there exists no general optimal size for the robot. So, the option to adjust the size of the robot was added through two buttons. In Figure 2.10, the minimal and maximal sizes are compared. It is not just a visual change, as all sensors and the speed of the robot are also adjusted corresponding to its size.

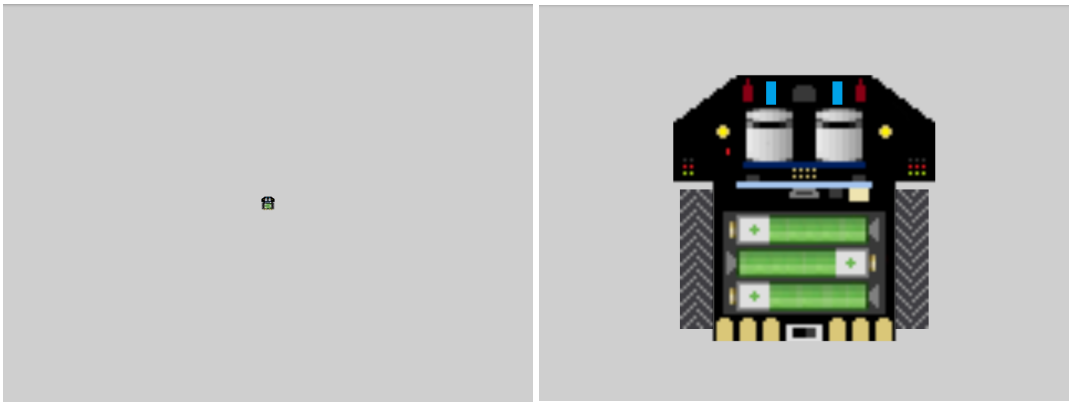


Figure 2.10. Screenshots of the minimal and maximal size of the Maqueen Lite robot in the robotics simulator.

Save and load to JSON files

To not lose all the invested time in building a cool environment or to share it with others, the functionality to save the current environment to a file was added. To do this, the "save to file" button can be pressed. After naming the file, it will be saved to the downloads folder as `<name>.json` or `mylevel.json` if no name was provided. In the file, all objects and lines, as well as the start position and size of the robot, are saved. This file can then be shared and loaded into the simulator by pressing the "load from file" button. After selecting the corresponding file, the saved environment will appear.

Table 2.1. Table of all commands from the `mbrobot`, `mbrobot_plusV2`, and `callibot` libraries. ✓ simulated, ✕ not simulated, - not part of the library.

Command	mbrobot	mbrobot_plusV2	callibot
General Movement			
forward()	✓	✓	✓
backward()	✓	✓	✓
left()	✓	✓	✓
right()	✓	✓	✓
stop()	✓	✓	✓
setSpeed(s)	✓	✓	✓
leftArc(r)	✓	✓	✓
rightArc(r)	✓	✓	✓
calibrate(o, d, a)	✓	✓	-
motL.rotate(s)	✓	✓	-
motR.rotate(s)	✓	✓	-
Servo Control			
setServo(s,a)	✕	✕	-
setMinServoDuty(x)	-	✕	-
setMaxServoDuty(x)	-	✕	-
Sensors			
getDistance()	✓	✓	✓
irLeft.read_digital()	✓	✓	-
irRight.read_digital()	✓	✓	-
irL1.read_digital()	-	✓	-
irL2.read_digital()	-	✓	-
irM.read_digital()	-	✓	-
irR1.read_digital()	-	✓	-
irR2.read_digital()	-	✓	-
irL1.read_analog()	-	✓	-
irL2.read_analog()	-	✓	-
irM.read_analog()	-	✓	-
irR1.read_analog()	-	✓	-
irR2.read_analog()	-	✓	-
irLeftValue()	-	-	✓
irRightValue()	-	-	✓
tsValue()	-	-	✓
tsLeftValue()	-	-	✓
tsRightValue()	-	-	✓
Signaling			
setLED(b)	✓	✓	✓
setLED(l,r)	✓	✓	-
setLEDLeft(x)	✓	✓	✓
setLEDRight(x)	✓	✓	✓
setAlarm(x)	✓	✓	-
beep()	✓	✓	-
fillRGB(r,g,b)	✓	✓	-
setRGB(x,r,g,b)	✓	✓	-
clearRGB()	✓	✓	-

Chapter 3

Implementation

This section will cover details about the implementation of the simulator. All code was written in TypeScript while using the Vue 3 framework for UI components. This language and framework were chosen to align with the pre-existing code of WebTigerPython and are compatible with Phaser 3.

3.1 Microcomputer simulator

Both the micro:bit simulator [14] and Calliope mini 3 simulator [3] are pre-existing simulators that were embedded as an iframe into WebTigerPython. So, the simulators are other web pages loaded into WebTigerPython as an HTML component.

Because the Calliope mini 3 simulator is based on the micro:bit simulator, as already mentioned in Subsection 1.2.2, they use the same communication interface. Thus, most of the code can be used for both simulators and does not have to be adjusted depending on the simulator.

3.1.1 Code execution

Normally, Python code written on WebTigerPython is directly executed by WebTigerPython using pyodide, but the code used for the simulator is sent to the iframe and is executed in MicroPython by the simulator. We decided to do this as they also simulate hardware-specific limitations, such as memory restrictions, thus leading to a more realistic microcomputer simulation, which in turn leads to a more realistic robotics simulation.

3.1.2 Communication

Due to the simulator being run on a different web page, communication with the simulator is restricted due to security concerns. Thus, you can't just call the functions of the simulator directly, instead, you have to send messages with the window.postMessage() function and listen for responses through an event listener. Through these messages, we send the code to be executed to the simulator, receive print statements and error messages to display in our console output, mute and unmute the simulator, and start and stop the simulator. This process is visualized in Figure 3.1.

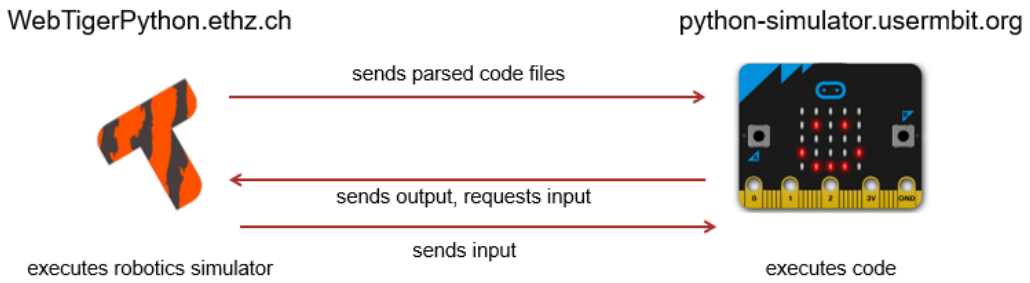


Figure 3.1. Simplified graphic displaying the code execution and communication between WebTigerPython and the micro:bit simulator.

Due to iframes being frequently used as ads, communication with them is restricted until they are interacted with. Thus, the iframe needs to be clicked once before the code can be run on it. That's why we decided to hide the control buttons for the iframe until it was started by clicking on the start button on the iframe once, as shown in Figure 3.2.

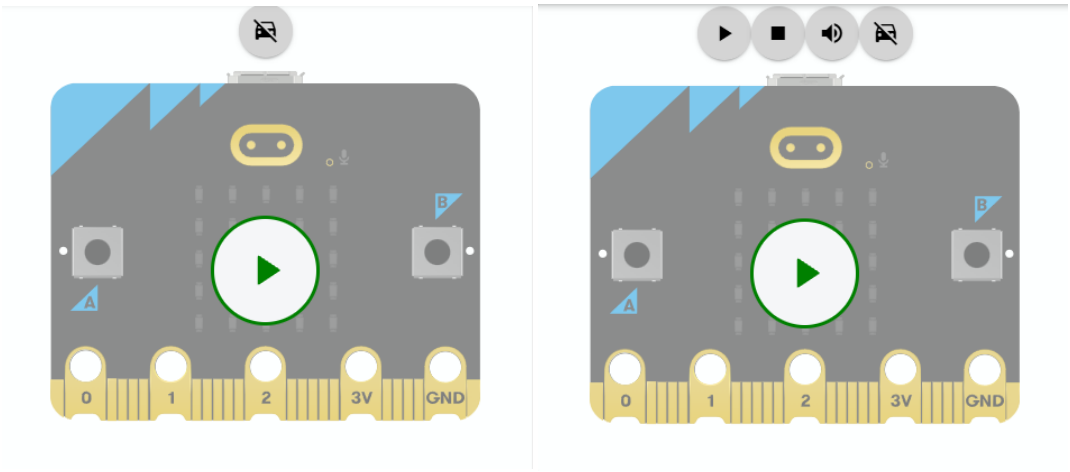


Figure 3.2. Screenshots of Visual UI of micro:bit simulator before being clicked once (left) and after being clicked once (right).

3.1.3 Pin simulation workaround

The pre-existing microcomputer simulators are not fully implemented yet. For example, the pins of the microcomputers are currently not simulated at all. So, writing and reading from or to pins does nothing in the simulation. This is quite unfortunate for this thesis, as the pins are used to control the robotic extensions and thus are needed to simulate the robots. After contacting the Micro:bit Foundation and verifying that there will be no pin simulation implemented in the near future, we briefly discussed implementing the pin simulation ourselves but quickly realized that this would heavily exceed the scope of this thesis.

So, we decided to devise a workaround for the missing pin simulation. We now parse the code before execution and replace the pin writes with print statements corresponding to the write statements and the pin reads with input statements requesting the needed information from the robotics simulator.

To distinguish between actual print statements of the code and the pin read and writes, a predetermined randomly generated string of alphanumeric characters of length 10 has been added in front of every pin-related print statement.

Figure 3.3 shows an example of the parsing process of the pin simulation workaround.

```python
def getDistance():
    pin1.write_digital(1)
    pin1.write_digital(0)
    A=machine.time_pulse_us(pin2,1,50000)
    B=(A>>6)+(A>>10)+(A>>11)+(A>>12)+1
    return max(min(B,500),0)if B>0 else 255
```

(a) `getDistance()` function form the `mbrobot.py` library before parsing.

```typescript
export function parseForMcQueenLite(code: string): string {
  //getDistance()
  code = code.replaceAll(
    "machine.time_pulse_us(pin2,1,50000)",
    'int(input("' + randomParseString + 'getDistance"))',
  );
  //parsing of other functions
  (...)
}
```

(b) parse function from `simulatorParser.ts` used to replace the pin read with an input function.

```python
def getDistance():
    pin1.write_digital(1)
    pin1.write_digital(0)
    A=int(input("IbEqa3GPv6getDistance"))
    B=(A>>6)+(A>>10)+(A>>11)+(A>>12)+1
    return max(min(B,500),0)if B>0 else 255
```

(c) resulting `getDistance()` function after being parsed be the `parseForMcQueenLite(...)` function.

Figure 3.3. An example of the parsing process used to simulate pin functionality.

Efforts were made to keep this parsing as general as possible to make the simulator work for other custom libraries that work with these robots. Unfortunately, we did not manage to achieve this with our workaround, as the commands used to read and write to the pins are not unique, and there are simply too many to efficiently parse them all. Thus, to implement a simulator working independently of the library, actual pin simulation is necessary. So we decided to only parse the three most commonly used libraries, `mbrobot`, `mbrobot_plusV2` and `callibot`, to not cause unexpected errors in other libraries.

3.2 Robotics simulator

In general, the simulator was written with adaptability and support for future robots in mind. To achieve this, typical object-oriented programming tactics were used. For example, an interface was used for the robots to keep all code interacting with the robots as generic as possible. Also, parts of the robots, like the ultrasonic sensor, were written as a separate class to be usable by all robots.

3.2.1 Simulating movement

To achieve realistic movement, the robot was split into three separate objects in the simulation: Its body and the two wheels. Through this separation, forces can be applied to the individual wheels. For example, to simulate a right turn, instead of applying a rotational force to the robot, we apply a backward force on the right wheel and a forward force on the left wheel, as visualized in Figure 3.4. This results in a more accurate rotational force, as the physical robot can also only apply forward and backward forces to its wheels. Which in turn also leads to more accurate robotics simulation.

In addition, this made the simulation of more advanced commands like `leftArc(x)` easier to simulate, as all movement commands call the internal `setMotor(...)` function which through our parsing directly tells us how much and in which direction to apply force to the wheels.

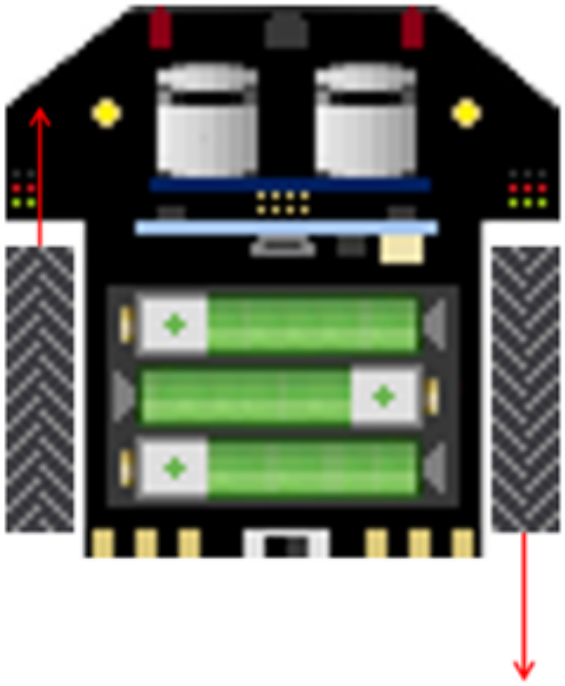


Figure 3.4. Graphic displaying the forces applied to the wheels to simulate a right turn.

3.2.2 Simulating sensors

To realize the ultrasonic sensor simulation, an additional tool was used, the phaser-raycaster plugin [18]. This plugin allows efficient object detection in front of the robot by casting rays in a cone until they collide with an object and then returns the closest object in this cone, as demonstrated in Figure 3.5.

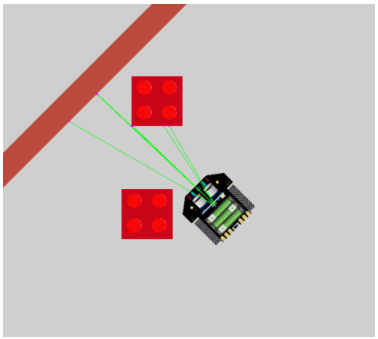


Figure 3.5. Screenshot of the ultrasonic sensor in action in the simulator visualized through the debug mode of the raycaster plugin.

This is not an exact simulation, as the actual ultrasonic sensor sends out ultrasonic waves and then calculates how much time it takes for the waves to return, thus, objects angled more than 25 degrees are not reliably detectable for the sensor, as measured by Gianluca Danieletto. To increase the accuracy of the simulation, this could also be implemented in the simulation. However, this could also be seen as an unwanted hardware limitation, and thus, one could argue to keep the sensor angle independent in the simulation.

The infrared sensors are implemented by small invisible boxes that check if they overlap with the black circles created with the "pen" tool. The boxes are visualized in green in Figure 3.6. The actual sensors do not check if the surface is bright or dark, instead, they check if the surface is reflective or not. Usually, most bright surfaces are reflective and dark ones are not. Thus, the simulation is representative of the actual sensors.

For the analogue infrared sensor reads, only supported on the Maqueen Plus V2, an expected value of the current surface is returned. This value is 230 for a bright surface and 100 for a dark surface. These values were also chosen based on measurements made by Gianluca Danieletto.

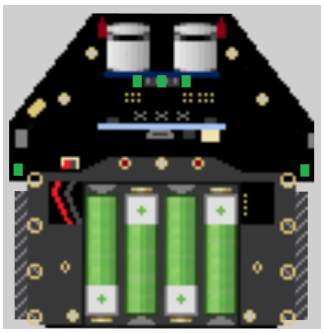


Figure 3.6. Screenshot of the five infrared sensors on the Maqueen Plus V2, visualized as green boxes.

3.3 Environment editor

To ensure no lost progress in the environment editor, the environment is saved to the session storage of the browser after every new addition or edit. This ensures that the environment does not get lost after switching the device settings or even after reloading the page, while still allowing to have multiple tabs with different environments at once.

3.3.1 Pen tool optimisations

The "pen" tool creates a line by adding many circles to the background, as shown in Figure 3.7. To avoid adding too many circles in one place, a minimal distance has to be moved with the input before the next circle gets drawn. This heavily reduces the number of circles added and thus makes saving and loading the environment more efficient.

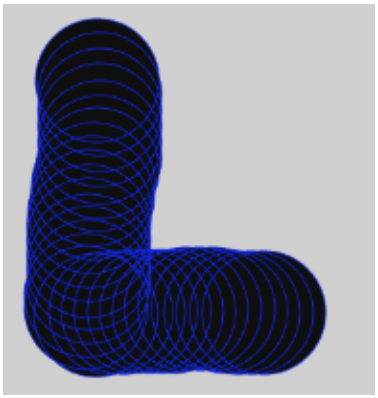


Figure 3.7. Screenshot of a drawn line in the simulator with the individual circles highlighted in blue by the Phaser debug mode.

In a previous version, if the "pen" tool was moved too fast, a non-consistent line was drawn, and individual circles were visible. This was rather visually unpleasant and inconsistent with how we expect a pen to work. To address this, more circles are now recursively added in between two points if they are too far apart. A direct comparison is visible in Figure 3.8.

Figure 3.8. Comparison of screenshots of lines drawn by the "pen" tool while doing fast scribbles with the previous version (left) and the current version (right).

3.3.2 Saving custom environments

To share or save for future use, the environments can be saved to JSON files. To make storage more compact, all objects are placed and saved only at full integer coordinates. In addition, there are only eight possible rotations for objects saved as a number from zero to seven, where the number n represents a rotation of $n * \frac{\pi}{4}$. The data was kept compact to enable future plans to share the environments per URLs, which have a rather small upper limit of data storage.

Currently, the lines of the pen are the least efficiently stored objects as we save the lines as multiple individual circle objects in storage. An optimization was added so that the tool does not create circles too close to each other, but it still requires a lot of circles for a line and thus a lot of storage. If, in the future, the data is too big for the URLs, this is certainly where the most optimization can be applied.

Chapter 4

Evaluation

4.1 Setup

An almost final build of the simulator was shown to and tested by Jeannine Marty [13], a secondary school teacher who previously worked for the ABZ team. Thus, she has great knowledge of the ABZ workflow as well as teaching in general. We would have liked to test the simulator with a class as well, but unfortunately, due to time constraints, it was not possible. However, through Jeannine Marty's many experiences of teaching students, she has a great eye for common problems that might occur for students. And thus shared her great observations with us.

After showing and explaining the functionality of the microcomputer and robotics simulator to her, she was asked to explore the environment editor on her own while speaking her thoughts out loud. She was then asked several questions about the intuitivity as well as the visual design of the simulator and its UI. She was also asked if she had already discovered some design flaws that would probably confuse students. The questions led to an open discussion about the simulator. Her thoughts and remarks were noted down on a tablet during the session. At the end, the notes were discussed together and compiled to the most important points. This entire session took about an hour and a half.

4.2 Results

This section will talk about the qualitative feedback given by Jeannine Marty by directly talking to her and taking notes.

General impression

In general, she liked the simulator and thought it was a good addition to WebTigerPython. She liked the overall visual style and thought it fitted well with the rest of WebTigerPython and was not too distracting. However, she found all the components at once to be quite overwhelming, as there was a lot of information on the screen at once. She mentioned that students would probably need a slow, step-by-step introduction, starting with the microcomputer simulator first and then later introducing the robotics simulator.

Layout

She noted that it was very beneficial to see all the components next to each other on the same screen, as this should help the students to get a deeper understanding of robotics. She elaborated that she found this to be a good visual representation of the process from left to right, first seeing the code, then the microcomputer and finally the simulated robot.

User interface

She did not have many notes about the user interface of the microcomputer simulator, and it seemed to be easy to understand. In contrast, she found the UI of the environment editor rather bloated and overwhelming. She had several suggestions to improve its readability and make it more manageable. Her concrete suggestions are listed below.

Highlight selected tool

She mentioned that it was hard to know which tool was currently selected as there was no visual indication, and it had to be kept in memory. She suggested changing the colour of the button of the currently selected tool to indicate which one is in use. Another idea that came up was to also adjust the displayed cursor to represent the selected tool.

Grouping the buttons

At the time of the interview, all buttons were placed in one line, as displayed in Figure 4.1. She brought up that grouping the buttons visually by similar functionality would ease the readability of the buttons. Furthermore, it would make it easier to learn and remember the button's functionality. Differing colours could also be used to add further distinction.

Figure 4.1. Screenshot of the old UI of the environment editor shown to and evaluated by Jeannine Marty.

Overlap environment editor buttons and simulator

In certain layouts, the buttons overlapped with the simulator, as shown in Figure 4.1. This is not optimal as it restricts the visibility as well as the functionality of the simulator environment. She suggested adjusting this to no longer overlap, as it can be quite restrictive in certain layouts.

Environment editor

She liked the small but diverse selection of tools to create and edit the environment. However, she missed different pen sizes for the tool and suggested adding them in a drop-down menu while selecting the tool to enable more specific uses for the “pen” tool.

Saving to files

She mentioned that it might be confusing for students to have to save the code and the environment on two separate files and with two different buttons. She said it could lead to students saving the wrong file and not noticing it, thus losing their code or their environment. She suggested an option to save both at once into one folder so that they could also be loaded again together, this would make it easier for students to save and organize their files.

User guidance

She liked the tooltips given as quick reminders of the function of the buttons. However, she suggested that a guide button could be a helpful addition to give a more extensive overview and explanation of the functionality of the simulator and its buttons for new users.

Chapter 5

Conclusion and future work

5.1 Conclusion

In this thesis, we successfully integrated the the micro:bit simulator [14] and the Calliope mini 3 simulator [3] into WebTigerPython. We then built a robotics simulator in Phaser 3 [10] upon these microcomputer simulators. To make this possible, we devised a workaround for the unimplemented pin simulation of the microcomputer simulators.

We added simulation for the three most commonly used driving robots, the Maqueen Lite [6], Maqueen plus V2 [5] and Calli:bot [11], as well as simulated almost all of their commands of their default libraries. Currently, no other libraries are functional. To enable library-independent simulation, pin simulation would have to be added in the future.

To meaningfully simulate and test the sensors of the robots, we added a simple but effective environment editor that enables the users to build small environments for the robots to interact with.

We also added the option to save and load the created environments to and from files. This enables the ability to share custom environments, thus enabling teachers to create and give virtual challenges to their students.

The simulator so far was only evaluated by experts and not yet used in a classroom setting. Thus, further testing in a classroom setting will be required. But we are confident that the simulator can soon be added to the public version of WebTigerPython after addressing some bugs and further improving the UI.

5.2 Future work

In almost every meeting we had, we came up with another idea for a cool addition to the simulator. We managed to implement many of these ideas, but there are still a lot of additions that can be made to the simulator. In this section, some of our most promising ideas that are not yet implemented are listed.

Improving UI of environment editor

In the evaluation with Jeannine Marty, detailed in Section 4.2, the UI of the environment editor was one of the most critiqued points. Thus, it can be greatly improved upon. Some improvements based on her feedback have already been applied. However, further improvements to the UI are still possible. This would then allow for the addition of more functionality. As with the current layout, adding more buttons would make the layout even more bloated and overwhelming.

Controlling sensors of the microcomputer

The sensors of the microcomputers don't measure actual data. However, the sensors can be set to measure a given value. An interface to set these values could be added into WebTigerPython, similar to the one implemented in the micro:bit Python Editor webpage [8]. This would allow an even further advanced simulation of the microcomputers and allow for a wider amount of programs to be testable in the simulator.

To take this a step further, certain sensors, like the accelerator, could even be set to values calculated from the robotics simulator.

Fast mode

The simulation is currently at a realistic speed of the physical robot. However, this is rather slow. A helpful improvement to the simulator could be a speed-up mode to get even faster feedback on your executed code.

Adding support for servo attachments

The Maqueen Lite and Maqueen Plus V2 both have attachments that can be added to them and controlled with the servo commands. These attachments would be a good addition to the simulator, especially the claw and loader attachments, shown in Figure 5.1, as they are frequently used by the ABZ team and would enable even more possibilities with the simulator.

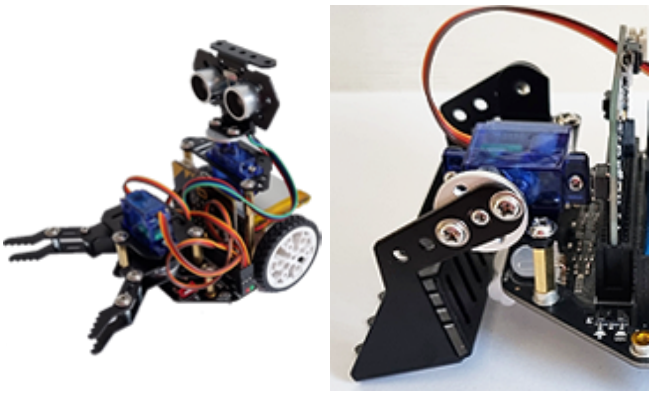


Figure 5.1. Images of two attachments for the Maqueen robots, the claw (left) and loader (right).

Adding support for other robotics

Not only driving robots are used with microcomputers, but there are also other attachments, such as the LED ring, that could be added to the simulator. An example of the LED ring can be seen in Figure 5.2.

Figure 5.2. Image of the LED ring attachment for the micro:bit.

Gamify the robotics simulator

Another idea was to add a target zone that can be placed, which the robots have to reach. When the zone is reached, a small feedback would pop up that celebrates that they successfully solved the challenge. In combination with the simulation of the attachments, as discussed above, alternative goals could be introduced, such as picking up a certain number of objects and delivering them to a target area.

Additional interactive elements could be added, such as objects with spikes that have to be avoided, otherwise, the challenge would have to be restarted. These objects could even be modified to move or chase after the robot.

This would enable challenges not possible outside of the simulator and probably increase the fun factor of using the simulator. However, this should not be taken too far to still make working with the physical hardware desirable.

Adding pin simulation

Adding pin simulation would allow library-independent simulation of the robots. It would also avoid the parsing steps and thus provide an even more direct and realistic simulation of code execution.

Bibliography

[1] ABZ. Nachhaltige Vermittlung von Wissen im Bereich Informatik. http://www.abz.inf.ethz.ch/. Accessed: 2024-08-29.

[2] Clemens Bachmann. WebTigerJython 3 – A Web-Based Python IDE Supporting Educational Robotics. Master thesis, ETH Zurich, Zurich, 2023.

[3] Calliope and Lulububu. Micropython for calliope mini v3. https://github.com/calliope-edu/micropython-calliope-mini-v3/tree/v2.1.1-cmini3. Accessed: 2025-02-26.

[4] Gianluca Danieletto. Gianluca Danieletto Contact. https://inf.ethz.ch/people/people-atoz/person-detail.MjA2NTMx.TGlzdC8zMDQsLTIxNDE4MTU0NjA=.html. Accessed: 2025-02-28.

[5] DFRobot. Maqueen Plus V2. https://learn.dfrobot.com/makelog-313320.html. Accessed: 2025-02-21.

[6] DFRobot. Maquenn Lite. https://www.dfrobot.com/product-1783.html. Accessed: 2024-08-29.

[7] Micro:bit Educational Foundation. Micro:bit. http://microbit.org/. Accessed: 2024-08-29.

[8] Micro:bit Educational Foundation. Micro:bit Python Editor. https://python.microbit.org/v/3. Accessed: 2025-02-27.

[9] Mat Groves et al. PixiJS. https://pixijs.com, 2013. Accessed 2024-02-08.

[10] Phaser Studio Inc. Phaser 3. https://phaser.io/. Accessed: 2025-02-21.

[11] Knotech. Calli:bot. https://shop.knotech.de/calli-bot/244/calli-bot-2. Accessed: 2025-02-21.

[12] Tobias Kohn. Teaching Python Programming to Novices: Addressing Misconceptions and Creating a Development Environment. Doctoral thesis, ETH Zurich, Zürich, 2017.

[13] Jeannine Marty. Jeannine Marty LinkedIn. https://ch.linkedin.com/in/jeannine-marty-61b42421a. Accessed: 2025-02-25.

[14] Damien George Natt Hillsdin, Robert Knight. Wasm-based micropython micro:bit v2 simulator. https://github.com/microbit-foundation/micropython-microbit-v2-simulator. Accessed: 2025-02-26.

[15] Calliope project. Calliope mini 3. https://calliope.cc/en/calliope-mini/calliope-mini-3. Accessed: 2025-02-21.

[16] Noe Schaller. Ts_maqueen. https://github.com/NoeSchaller/TS_Maqueen. Accessed: 2025-02-21.

[17] Nicole Trachsler. WebTigerJython – A Browser-based Programming IDE for Education. Master's thesis, ETH Zürich, 2018.

[18] Wiserim. Phaser raycaster plugin. https://github.com/wiserim/phaser-raycaster. Accessed: 2025-02-24.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten schriftlichen Arbeit. Eine der folgenden drei Optionen ist in Absprache mit der verantwortlichen Betreuungsperson verbindlich auszuwählen:

- (●) Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Es wurden keine Technologien der generativen künstlichen Intelligenz[1] verwendet.
- ( ) Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Als Hilfsmittel wurden Technologien der generativen künstlichen Intelligenz[2] verwendet und gekennzeichnet.
- ( ) Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Als Hilfsmittel wurden Technologien der generativen künstlichen Intelligenz[3] verwendet. Der Einsatz wurde, in Absprache mit der Betreuungsperson, nicht gekennzeichnet.

Titel der Arbeit:

2D Robotics Simulator for WebTigerPython

Verfasst von:
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Zürcher	Kai

Ich bestätige mit meiner Unterschrift:

- Ich habe mich an die Regeln des «Zitierleitfadens» gehalten.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu und vollständig dokumentiert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Eigenständigkeit überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Zürich, 8.03.2025	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie grundsätzlich gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

[1] z. B. ChatGPT, DALL E 2, Google Bard
[2] z. B. ChatGPT, DALL E 2, Google Bard
[3] z. B. ChatGPT, DALL E 2, Google Bard