Department of Computer Science
Algorithms and Didactics

Turtle Graphics for WebTigerJython-3

Julia Bogdan

Bachelor's Thesis

2023

Supervisors: Prof. Dr. Dennis Komm
Alexandra Maximova

Abstract

This bachelor thesis presents the development and integration of Turtle Graphics into WebTigerJython-3, an interactive web-based educational platform aimed at introducing beginners to programming and computational thinking.

Turtle Graphics is a widespread visual programming concept made popular by Seymour Papert in the 1960s. It revolutionized the way beginners learn how to write code by allowing them to create intricate drawings and designs using a simple set of commands.

Integrating this feature into WebTigerJython-3 aims to enhance the learning experience by fostering creativity, logical reasoning, and problem-solving skills among learners. The thesis outlines the design and implementation of the Turtle Graphics module with a focus on implementation and design to support the further development of the project.

Acknowledgement

I would like to extend my heartfelt gratitude to several individuals who have been instrumental in the successful completion of this thesis. First and foremost, I express my sincere appreciation to my supervisor, Alexandra Maximova, for her unwavering support, invaluable guidance, and insightful feedback throughout this journey.

I am deeply grateful to Prof. Dr. Dennis Komm for giving me the opportunity to complete this thesis within his research group. His encouragement and trust have been instrumental in shaping the direction of my work.

I would like to thank Cédric Donner, Clemens Bachmann and Andreas Aeberli for their invaluable help in answering my technical questions and sharing their expertise. Their guidance was crucial in overcoming challenges and refining the technical aspects of this thesis. I would also like to thank Prof. Dr. Tobias Kohn for laying the foundation for this thesis with his excellent work on TigerJython.

Finally, I want to express my deepest gratitude to my friends and family for their unwavering support and understanding. Their encouragement and mental support have been my pillars of strength throughout this endeavor.

Contents

Chapter 1

Introduction

Educational computing tools are constantly evolving to improve the learning experience. This chapter introduces the context and motivation behind the exploration of Turtle graphics within the WebTigerJython-3 environment.

1.1 Background

Turtle Graphics is a valuable tool for teaching computer programming to students of all ages. It provides a visual and interactive way for students to learn basic programming concepts such as loops, conditionals, and functions. Turtle Graphics allows students to see the immediate results of their code, making programming more tangible and engaging. Made popular by Seymour Papert in the 1960's [11], Turtle Graphics has a long history of use in educational contexts and has been proven to be an effective teaching method.

The Center of Computer Science Education at ETH Zurich (ABZ) and the Algorithms and Didactics Group have developed a spiral curriculum to teach computer science at all school levels. Programming starts in kindergarten with Bluebots and continues in primary school with Turtle Graphics. Turtle Graphics, robotics, and unplugged activities are repeated and deepened in each iteration. As a result, a wealth of Turtle Graphics educational materials has been developed from kindergarten to high school, including the "Einfach Informatik" series of textbooks [4, 6, 7] and the programming environments XLogoOnline [16], TigerJython [8], and WebTigerJython[19].

The programming environments were designed with two goals in mind: First, their user interfaces should be simple, intuitive, and free of unnecessary clutter, and second, they should be easy to deploy. However, TigerJython is a desktop application and as such requires schools to incur additional administrative costs in installing and updating the application on school laptops. WebTigerJython, developed by Nicole Roth (formerly Trachsler) in her Master's thesis [19], is based on Skulpt, a browser-based implementation of Python. It has all the advantages of a web version, but unfortunately it cannot reuse TigerJython libraries directly. All libraries have to be reimplemented in Skulpt [15], which compiles Python code written by students into Javascript code that can be executed in the browser.

WebTigerJython-3, developed by Clemens Bachmann in his Master's thesis [2], uses Pyodide [13] to do this translation. Pyodide aims to bring the Python scientific stack to the web using WebAssembly. It provides a full Python interpreter that runs entirely in the browser, making it more portable and easier to use than other solutions such as the aforementioned Skulpt. In addition, Pyodide includes many popular scientific libraries such as NumPy and SciPy, making it a powerful tool for data analysis and visualisation. As Pyodide uses the full Python interpreter, it can run any Python code, whereas Skulpt is based on a subset of Python and may not support certain features or libraries.

Therefore, the inclusion of Turtle Graphics support in WebTigerJython-3 is significant. We can also take advantage of the work done by Andreas Aeberli in his Master's thesis [1], which laid the essential groundwork for PixiJS and its seamless integration with Pyodide.

1.2 Main Contribution

In this thesis we design, implement, test and integrate a Turtle Graphics library in WebTigerJython-3. The prototype supports all core functionalities. This includes the basic Turtle movements such as moving forward and backward and turning left and right. Changing the speed at which the Turtle moves, making the Turtle sprite visible and invisible. Changing the width and color of the tracks left by the Turtle as it moves, as well as turning them off and on. The process of implementation as well as a documentation of the exact functionality can be found in this thesis. Additional information about how to implement further functions can also be found in the TigerJython-3 documentation.

Chapter 2

Related Work

In this chapter we will discuss some work related to this thesis. We will first discuss Turtle Graphics in general, then TigerJython and the existing WebTigerJython implementation before discussing WebTigerJython-3 and some other Turtle Graphics implementations.

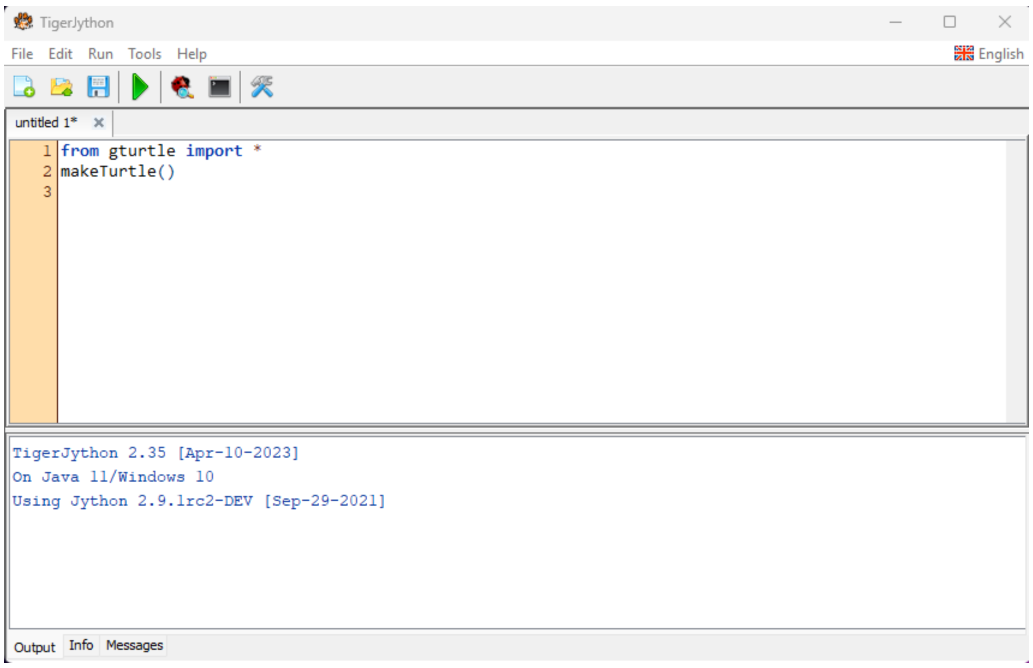
2.1 Turtle Graphics

Turtle Graphics is a graphics model that was made popular as an educational tool by Seymour Papert in the 1960s [11]. The main components are the canvas, the Turtle and the pen. The Turtle is an object that sits on a pen and has a small set of attributes that can be adjusted by a set of well-defined commands. These commands allow the Turtle to move around the canvas, drawing lines and shapes. This visual representation of the Turtle's state allows users to see the immediate effects of their programming commands. Planning their artwork allows users to develop their problem-solving skills. They have to break down complex tasks into smaller, more manageable steps to achieve the desired result. This provides a visual and interactive way for students to learn basic programming concepts such as loops, conditionals and functions.

Turtle Graphics has a long history of use in educational contexts and has been proven to be an effective teaching method. Over the years, the Centre of Computer Science Education at ETH Zurich (ABZ) has developed a wealth of Turtle Graphics educational materials, including the XLogoOnline [16], TigerJython [8] and WebTigerJython [19] programming environments. Not only the ETH sees the value of Turtle Graphics in early computer science education, the University of Oxford has also developed an educational programme called the Turtle System for the UK national computer science curriculum [10].

2.2 TigerJython

TigerJython is a programming environment developed by Tobias Kohn in 2017 [8]. It is based on Jython, a Python implementation for the Java Virtual Machine (JVM), which provides a versatile base that combines the accessibility of Python with the cross-platform functionality of Java. The TigerJython Turtle is the basis for our implementation of Turtle Graphics and also for WebTigerJython-3 as a whole. However, TigerJython is a desktop application. This adds to the administrative overhead, as the environment needs to be installed and maintained.



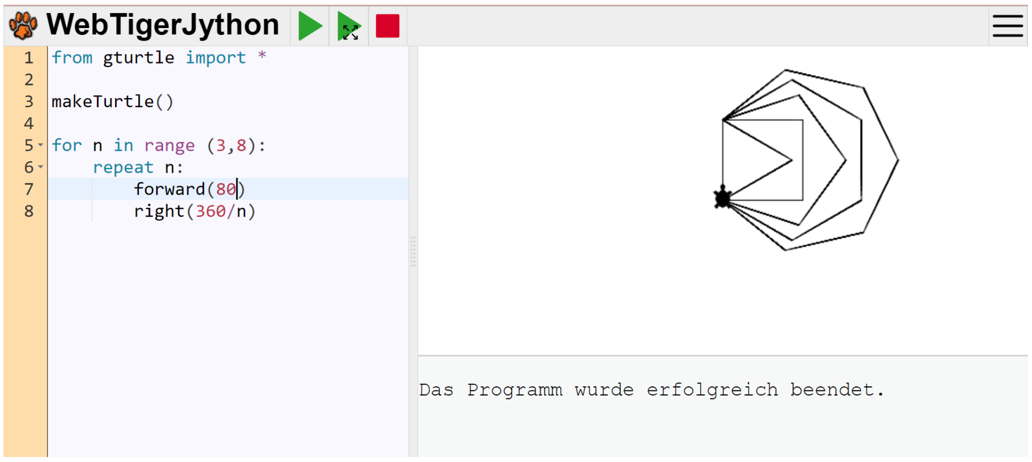
```
from gturtle import *
makeTurtle()
```

```
TigerJython 2.35 [Apr-10-2023]
On Java 11/Windows 10
Using Jython 2.9.1rc2-DEV [Sep-29-2021]
```

Figure 2.1. TigerJython IDE

2.3 WebTigerJython

WebTigerJython represents an evolution of the TigerJython programming environment adapted for web-based accessibility [19]. It is based on Skulpt, a browser-based implementation of Python [15]. With this it aims to eliminate the administrative overhead of desktop applications, but it has its own limitations. Skulpt makes it difficult to use Python libraries and impossible to import TigerJython libraries. However, there is an existing implementation of Turtle Graphics for Skulpt that could be reused. Skulpt is discussed in more detail in Section 2.5.



```
from gturtle import *

makeTurtle()

for n in range (3,8):
    repeat n:
        forward(80)
        right(360/n)
```

```
Das Programm wurde erfolgreich beendet.
```

Figure 2.2. WebTigerJython IDE

2.4 WebTigerJython-3

In his Master's thesis Clemens Bachmann laid the foundation for WebTigerJython-3, a rewrite of the WebTigerJython IDE [2]. This time it uses Pyodide, a WebAssembly based project that runs a Python interpreter in the browser. This makes it very easy to integrate popular Python packages like NumPy or SciPy. The author has also ported robotics to WebTigerJython using WebUSB. In addition, Andreas Aeberli implemented a first prototype of a Gamegrid-like module for WebTigerJython-3 as part of his Master's thesis [1]. This prototype also uses PixiJS and was the basis for our Turtle Graphics project.

2.5 Other Turtles

The concept of using an actor, such as a turtle or has gained significant popularity in programming education.

Scratch

Scratch is not based on Turtle Graphics, but is another visual programming language designed to make programming accessible and engaging for beginners. Unlike Turtle Graphics, Scratch is block-based. Users snap blocks together to create scripts that control sprites. These sprites don't typically draw but can move around perform simple animations and make sounds. Developed at the MIT Media Lab at the Massachusetts Institute of Technology in the early 2000s [17], Scratch offers an alternative entry point to programming for learners.

Python Turtle

The `turtle.py` module is part of the standard Python library and provides a very simple Turtle Graphics implementation [14]. It is based on Gregor Lingl's xturtle, which is an extended reimplementation of turtle.py from the Python 2.5 standard distribution [9]. The turtle.py module is designed to work in the context of a desktop environment and uses the graphical user interfaces provided by the Tkinter framework [18]. Tkinter was removed from the Pyodide standard library to reduce its download size and since it does not work in the WebAssembly VM [13].

Brython Turtle

Brython is a Python interpreter that implements its own Turtle module [3]. What makes it different from other Turtle Graphics implementations is that it uses resolution independent scalable vector graphics instead of a canvas to draw on. However, similar to Skulpt, Brython only supports pure Python libraries and no TigerJython libraries.

Skulpt Turtle

As mentioned above, WebTigerJython is based on the Skulpt Turtle [19]. Skulpt emulates the Python Turtle module `turtle.py` for a desktop environment by providing a Turtle graphics implementation that allows the Python commands to be drawn on an HTML canvas. While there is nothing wrong with the Skulpt Turtle itself, the Skulpt framework only allows certain Python libraries to be imported and used. Any library that is not supported would have to be ported manually [15].

XLogoOnline

XLogoOnline, as the name says, is an online IDE for XLogo. It was developed by Jacqueline Staub during her Master's thesis [16]. The XLogoOnline Turtle has 4 different stages that each offer a different level of complexity: Mini, Medi, Maxi, and Mega as seen in Figure 2.3. Starting with the commands as blocks that have to be pulled in place to control the Turtle up to a WebTigerJython integration added by Jonas Holzer in his Bachelor's thesis [5]. For this project we reuse the Turtle sprite used in XLogoOnline.

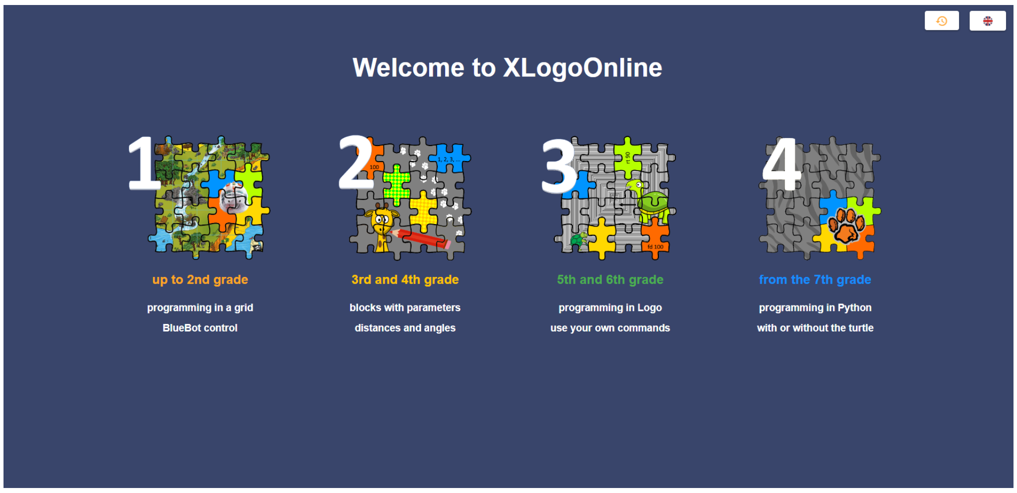


Figure 2.3. Mode selection screen XLogoOnline

Chapter 3

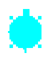
Appearance and Functionality

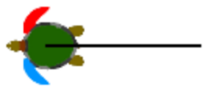
Creating our own version of Turtle Graphics in WebTigerJython-3 gave us the freedom to consider which elements of previous Turtle Graphics implementations to keep and which to reconsider. In this next chapter, we will navigate through this decision-making process and shed light on our choices.

3.1 Turtle Sprite

While it makes sense to keep all of the same functionalities of TigerJython's Turtle [8], the visual representation of the Turtle could definitely be improved. TigerJython uses a small monochrome Turtle that can be spawned in many different colors, as does WebTigerJython. XLogoOnline on the other hand uses a colorful Turtle with one blue and one red flipper [16]. For the purposes of this thesis, we chose to use the default Turtle sprite from XLogoOnline. However, PixiJS makes it very easy to add additional sprites or change the default sprite in the future.

To help users draw more precisely the decision was made to move the trace drawn by the Turtle on the layer in front of the Turtle sprite as pictured in figure 3.1b. This helps them see exactly where they are drawing without having to change to a transparent sprite.


(a) TigerJython Turtle


(b) WebTigerJython-3 Turtle and its trace

3.2 Features

We aim to include all the features supported by the TigerJython Turtle. The features should also be recognisable to users familiar with other Turtle Graphics implementations. This will allow an easy transition for users already familiar with TigerJython or Turtle Graphics in general, and ensure that any existing educational material remains relevant. Table 3.1 shows the features implemented in this thesis.

Table 3.1. Turtle Graphics functions supported in WebTigerJython-3

Function	**Description**
`makeTurtle()`	Creates a Turtle.
`forward(x: int)`, `fd(x)`	Moves the Turtle x steps forward in the direction it is facing.
`backward(x: int)`, `back(x: int)`, `bk(x: int)`	Moves the Turtle x steps in the direction opposite to the one it is facing.
`left(x: int)`, `lt(x: int)`	Rotates the Turtle x degrees to the left.
`right(x: int)`, `rt(x: int)`	Rotates the Turtle x degrees to the right.
`clearScreen()`	Erases all traces and sets the Turtle sprite back to the starting position.
`delay(x: int)`	Delays the further execution of user written code by x milliseconds.
`speed(x: int)`	Sets the speed of the Turtle's movement. The input parameter, 'x', is expected to be an integer with a valid range of 1 to 100, inclusive ($1 \leq x \leq 100$). A value of -1 for 'x' indicates the Turtle's movement is set to the maximum speed.
`penUp()`, `pu()`, `up()`	Lifts the pen up from the canvas, disabling drawing when the Turtle moves.
`penDown()`, `down()`, `pd()`	Sets the pen down on the canvas, enabling drawing when the Turtle moves.
`setPenColor(*args)`	Changes the color of the lines drawn by the Turtle. The input parameter `*args` can be in any of the formats described bellow in Section 3.2.
`setPenWidth(x: int)`	Changes the width of the pen to x pixels.
`hideTurtle()`	Hides the Turtle sprite and sets the speed to -1 (fastest speed possible)
`showTurtle()`	If the Turtle sprite is hidden, showTurtle() makes it reappear. Speed is set to 20.

Colors

The following value types are supported when referring to colors.

- Color names as defined by CSS [20]: “red”, “green”, “blue”, “white”, etc.
- RGB as three separate integers: (255,0,0), (0,0,0), (255,255,255), etc.
- RGB hex integers (0xRRGGBB): 0xff0000, 0x00ff00, 0x0000ff, etc.
- RGB(A) hex strings:
  - 6 digits (RRGGBB): “ff0000”, “#00ff00”, “0x0000ff”, etc.
  - 3 digits (RGB): “f00”, “#0f0”, “0x00f”, etc.
  - 8 digits (RRGGBBAA): “ff000080”, “#00ff0080”, “0x0000ff80”, etc.
  - 4 digits (RGBA): “f008”, “#0f08”, “0x00f8”, etc.
- RGB(A) strings: “rgb(255, 0, 0)”, “rgb(100% 0% 0%)”, “rgba(255, 0, 0, 0.5)”, “rgba(100% 0% 0% / 50%)”, etc.
- HSL(A) strings: “hsl(0, 100%, 50%)”, “hsl(0deg 100% 50%)”, “hsla(0, 100%, 50%, 0.5)”, “hsla(0deg 100% 50% / 50%)”, etc.

`hideTurtle()`

Usually, in Turtle Graphics `hideTurtle()` does two things. Firstly, it hides the Turtle sprite so that only the traces remain visible. Secondly, it sets the speed to -1. A speed of -1 usually disables any animations and all traces are drawn simultaneously unless `delay(seconds)` is called. As of now, in this implementation a speed of -1 sets the animation speed to maximum but doesn’t disable it completely. This becomes very apparent when many functions are called since even at a high speed they will visibly be drawn sequentially. More information on how this could be adapted in the future can be found in Section 5.2.

Chapter 4

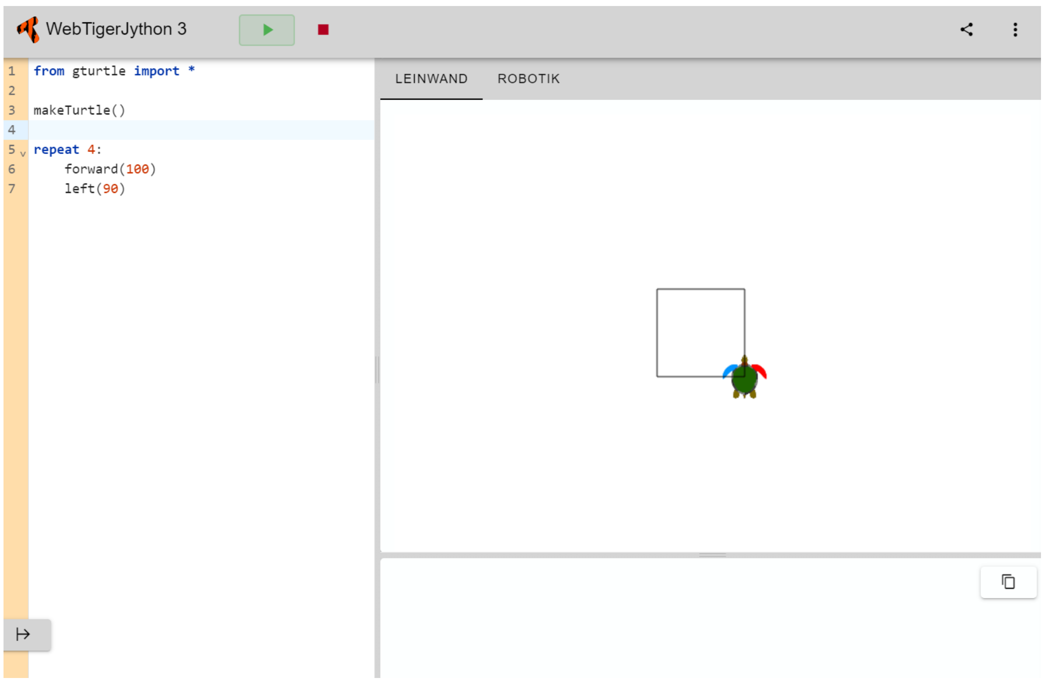
Implementation

This chapter aims to give an insight to the implementation of Turtle Graphics for WebTigerJython-3.

4.1 Overview

The user is presented with a web page containing an editor and a canvas as seen in Figure 4.1. The user can type Python statements into the editor and after pushing the "play" button, the given Python code is executed statement by statement and produces visual results on the canvas.

All our Turtle Graphics functions are implemented in a custom Python package called `gturtle.py` which has to be imported in order for the Turtle Graphics functions to be able to be called.



```python
from gturtle import *

makeTurtle()

repeat 4:
    forward(100)
    left(90)
```

Figure 4.1. Screenshot of WebTigerJython-3

As seen in Figure 4.2, WebTigerJython-3 consists of two threads, the web worker and the main thread. Web workers allow given programs to be executed in a background thread separate from the main thread of a web application [21]. In our case, the web worker runs our Pyodide instance in the background, which loads and executes the user-written Python code. The main thread takes care of everything else, especially running PixiJS, the visual rendering library we use. Both PixiJS and Pyodide are explained in more detail in Section 4.3 and Section 4.4 respectively.

The statements written by the user are parsed by Pyodide and later executed by PixiJS. The Pyodide instance runs on the Java Virtual Machine (JVM) via WebAssembly and is running within the web worker. Pyodide loads all the necessary Python packages, including our own `gturtle` package.

PixiJS is also embedded in the web page (HTML) itself. However, contrary to Pyodide it runs dirctly in the main thread with only its functions being invoked through the WebWorker interface.

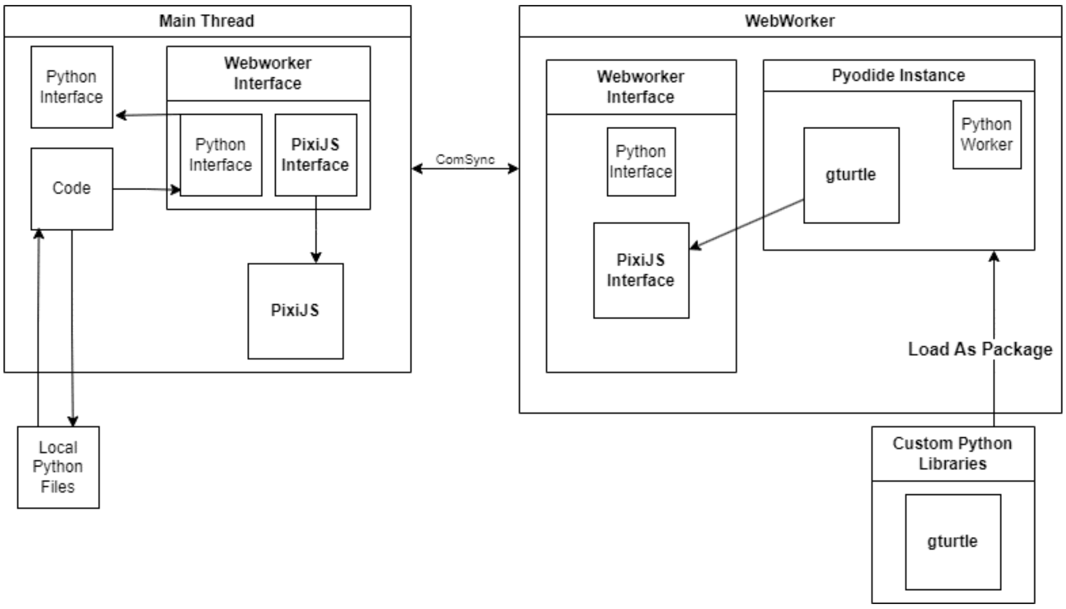


Figure 4.2. Overview with the Main Thread running PixiJS on the left and the WebWorker running Pyodide on the right.

Functions

When a function is called, it is first pre-processed by Pyodide itself, parsing and reading the command. Then the main thread takes care of the animation and visualisation.

The commands go through a multi-stage passing process and therefore need to be implemented in several different files. The first stage is implemented in `gturtle.py`, a Python library that is loaded by Pyodide and has to be imported by the user. For each of these Python functions there is a corresponding Typescript function in `pyodideworker.ts`, which is called from the Python side via the Pyodide bindings. As the name suggests, this is where the bindings are instantiated, allowing bidirectional data flow. It sets up some callbacks defined in `pyodideStore.ts` and proxies any function calls to the appropriate callbacks. This is where the individual `TurtlePixiHandler.ts` functions are called, the canvas is manipulated, turtle states are set and the turtle animations are implemented using PixiJS.

4.2 Turtle State

Internally, the turtle has several state variables that can be changed via function calls. These are kept track of in `TurtlePixiHandler.ts` to minimize the amount of information that has to be passed through from Python to PixiJS. An overview of which can be seen in Table 4.1.

Table 4.1. Turtle state variables

State Variable	**Description**
`exists: boolean`	True once `makeTurtle()` has been called and the turtle sprite has been created.
`visible: boolean`	True if the turtle sprite is currently visible.
`x: number`	Current X coordinate of the turtle on the canvas.
`y: number`	Current Y coordinate of the turtle on the canvas.
`angle: number`	Current angle in degrees of the turtle relative to the canvas.
`speed: number`	Current speed of the turtle.
`penDown: boolean`	True if pen is down and traces are being drawn.
`penColor: string | number`	Current color of the pen.
`penWidth: number`	Current width of the pen.

4.3 PixiJS

While it was possible to use an existing Turtle Graphics implementation for both TigerJython and WebTigerJython, it was not an option for WebTigerJython-3, mainly due to the lack of a Turtle Graphics implementation for Pyodide.

PixiJS, a 2D rendering engine designed for web browsers, uses WebGL and, when necessary, HTML5 canvases to render graphics. The engine provides a ticker mechanism that we use to regularly update frames and facilitate turtle sprite animations. By interfacing with our web worker, PixiJS allows WebTigerJython-3 to visually represent the execution of the user's code.

We use the concept of sprites to represent both the turtle and its traces. Sprites are visual objects that can be textured, transformed, animated and have a certain features. These features mostly correspond to the turtle states described in the section above. The canvas acts as a PixiJS container, capable of housing these visual elements. Containers in PixiJS can be assigned children, the turtle sprite and its traces are added as such children. This approach aligns with the object-oriented paradigm, enabling a structured and interactive representation of Turtle Graphics in WebTigerJython-3.

PIXI.Ticker

The turtle movements are implemented using PixiJS's Ticker class [12]. The Ticker class works by running a consistent update loop that triggers at a specified frame rate and abstracts the complexities of handling the requests of animation frames and delta time calculations while ensuring smooth animations. Each Turtle Graphics function call that triggers a turtle movement defines its own update function that is added to the Ticker. Once the movement animation has finished, the ticker isn't stopped but instead the update function is removed. This way the ticker can be reused for the execution of the next user command.

The ticker was especially useful for implementing `speed(x)` since this simply required to adjust the frame rate according to the newly set speed and not the individual update functions.

4.4 Pyodide

The Python integration in the web worker is done by Pyodide. Pyodide is a WebAssembly-based project that runs a Python interpreter in the browser. When using Pyodide, the Python interpreter and necessary standard library are loaded into the browser as a WebAssembly module. This allows the user to take advantage of Python's extensive library ecosystem. Any popular Python libraries such as NumPy or pandas can be imported and used directly in the browser. In addition, custom modules can be created and loaded in. In this case, a custom `gturtle` library was implemented, containing all the necessary Turtle commands.

Shared Array Buffer

The main challenge encountered in this project arose from the need to support multiple user prompts simultaneously. While implementing individual `forward()` or `right()` calls proved relatively straightforward, complications emerged when multiple function calls were involved. The turtle sprite's movement became unreliable, often rotating and moving simultaneously instead of sequentially.

This problem stems from Pyodide's inability to wait for the termination of one user-written command before invoking the next. As a result, subsequent commands are executed before their predecessors have completed. Ideally the end coordinates are calculated and the sprite is incrementally moved closer with each tick of the PixiJS ticker. However, this resulted in mid-movement adjustments of the end coordinates, causing the turtle to endlessly move in the wrong direction.

The underlying issue arises due to Pyodide and the web worker operating as separate processes in distinct threads with differing memory models, necessitating inter-process communication mechanisms.

The solution involves employing a shared array buffer to establish shared memory between processes. This approach differs from conventional *busy waiting*, as it utilizes interrupts to signal the completion of function calls [13].

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Turtle Graphics has proven to be an invaluable tool for teaching computer programming to learners of all ages. Its visual and interactive approach enables learners to grasp fundamental programming concepts such as loops, conditionals and functions. The immediacy of visualising the effects of their code within Turtle Graphics promotes a more tangible and engaging programming experience.

In this thesis we managed to implement Turtle Graphics for WebTigerJython-3. Our library consists of the most commonly used functions in Turtle Graphics and is ready to be tested in a classroom setting.

From a personal perspective, this journey has been enlightening. Gaining knowledge in web development and mastering the intricacies of Git in a professional setting has been valuable. Although challenging, this experience has significantly improved my skills and understanding of modern development and deployment practices.

5.2 Future Work

This thesis demonstrates the successful implementation of the core functionalities of Turtle Graphics for WebTigerJython-3. In this section we will delve into some further functionalities and improvements that could be added in the future.

Canvas

The current implementation of the canvas presents certain limitations that may hinder users' creative freedom. As observed, the canvas has a finite size, which restricts the extent to which users can draw and visualize their designs. Ideally, the canvas should be infinite and scrollable, allowing users to explore and create without boundaries. Additionally, the option to zoom in and out of the image being created would aid the user in creating bigger and more exact pictures.

Further Turtle Graphics Functions

While we were able to implement the core Turtle Graphics functions there are still some that are available in other IDEs but not in WebTigerJython-3. Table 5.1 provides an overview of some functions that should be implemented in the future and whether Python Turtle Graphics [14], TigerJython [8], WebTigerJython [19] and XLogoOnline [16] support said function.

Table 5.1. Turtle Graphics functions that have not been implemented yet.

Function	**Description**	**Python Turtle Graphics**	**TigerJython**	**WebTigerJython**	**xlogo online**
`setPosition(x, y)` and `setHeading(deg)`	Moves the turtle to the given coordinates or angle relative to the canvas without drawing the trace.	Yes	Yes	Yes	Yes
`home()`	Moves turtle to the center of the canvas with viewing direction to the north.	Yes	Yes	Yes	Yes
`circle(radius)` and `arc(radius, deg)`	Draw a circle with the given radius or an arc with given radius and angle.	Yes	Yes	Yes	Yes
`getPosition()` and `heading()`	Returns the current position or angle relative to the canvas of the turtle.	Yes	Yes	Yes	Yes
`startPath()` and `fillPath()`	Allow for shapes to be drawn and then filled.	Yes	Yes	Yes	No
`getPixelColor()`	Returns the color of the pixel (background or trace) at the current turtle position.	No	Yes	Yes	No

Animations with `hideTurtle()`

In order for a user to create animations in Turtle Graphics there has to be the possibility to switch off turtle movement and have all all traces be drawn instantaneously. This feature is currently not supported in WebTigerJython-3. The `hideTurtle()` function simply hides the turtle and speeds up the creation of traces. However, there is a significant delay between the creation of the different traces which makes the creation of smooth animations impossible in the current state.

To enable the proper functionality for `hideTurtle()` it is necessary to rethink the process traces are created when the Turtle sprite is invisible. The way traces are currently created, is that they are redrawn after each step of the Turtle from the Turtle's initial position to where it is standing after taking the step. This ensures a smooth and continues trace being drawn behind the Turtle however, also forces the traces to appear sequentially.

One solution would be to not generate the trace step by step but lay down the whole trace for each function call at once. Especially changing the Turtles orientation with `right()` and `left()` should be done in one step when the sprite is invisible.

Saving Images

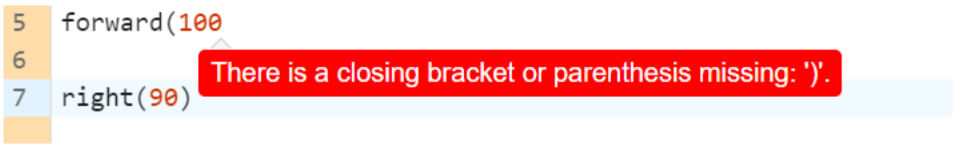
Just as users can save their code progress, it would be desirable to be able to save the pictures they create in the Turtle Graphics environment.

Additionally to static images, more advanced users can also create animations. Therefore, it is also worth exploring the possibility of saving these animations in the form of short videos or GIFs.

Allowing users to save the output images as files would give them a lasting record of their artwork and the opportunity to share it with others. This feature can instill a sense of pride and ownership in their creative journey, encouraging them to further explore and experiment with programming.

Custom Errors

To help users debug their code and improve their understanding of errors, it is highly beneficial to implement custom error messages. Ideally, an error message should be displayed when a turtle function such as `forward()` or `speed()` is called before a turtle is instantiated using `makeTurtle()`. This mechanism could be similar to the error message triggered by a missing bracket, providing a more informative and guided learning experience.



```
forward(100

right(90)
```

There is a closing bracket or parenthesis missing: ')'.

Figure 5.1. WebTigerJython-3 error for missing bracket or parenthesis

Exercises and Tutorials

Another nice feature would be to have a tab with exercises that users can solve in the browser. Ideally these would evaluate automatically allowing users to solve them completely independently. The first few exercises would function as a tutorial, introducing different concepts. A similar feature already exists in XLogoOnline [16].

Bibliography

[1] Andreas Aeberli. “Using Pyodide to Build Games in the Browser: Gamegrid”. MA thesis. ETH Zürich, 2023.

[2] Clemens Bachmann. “WebTigerJython 3 A Web-Based Python IDE Supporting Educational Robotics”. MA thesis. ETH Zürich, 2023.

[3] *Brython*. https://brython.info/. Accessed: 2023-08-02.

[4] Heinz Hofer et al. *Einfach Informatik 3/4. Programmieren und Rätsel lösen – Arbeitsheft*. de. Educational Material. Baar, 2021.

[5] Jonas Holzer. “Meet the tiger in a turtle shell: Reuniting WebTigerJython with XLogoOnline”. Bachelor’s Thesis. ETH Zürich, 2022.

[6] Juraj Hromkovič. *Einfach Informatik 5/6: Programmieren. Primarstufe*. de. Educational Material. Baar, 2018.

[7] Juraj Hromkovič and Tobias Kohn. *Einfach Informatik 7-9: Programmieren. Sekundarstufe I*. de. Educational Material. Baar, 2018.

[8] Tobias Kohn. “Teaching Python Programming to Novices: Addressing Misconceptions and Creating a Development Environment”. en. Doctoral Thesis. Zürich: ETH Zurich, 2017. ISBN: 978-3-906327-80-8. DOI: 10.3929/ethz-a-010871088.

[9] Gregor Lingl. *xturtle.py: a Tkinter based turtle graphics module for Python*. https://svn.python.org/projects/python/tags/r32/Lib/turtle.py. Accessed: 2023-08-08.

[10] Univeristy of Oxford. *Python Documentation - Turtle Graphics*. https://www.turtle.ox.ac.uk/. Accessed: 2023-08-02.

[11] Seymour Papert. “Teaching Children to be Mathematicians Versus Teaching About Mathematics”. In: *International Journal of Mathematical Education in Science and Technology* 3.3 (1972), pp. 249–262. DOI: 10.1080/0020739700030306. eprint: https://doi.org/10.1080/0020739700030306. URL: https://doi.org/10.1080/0020739700030306.

[12] *PixiJS*. https://pixijs.download/release/docs/index.html. Accessed: 2023-08-02.

[13] *Pyodide*. https://pyodide.org/. Accessed: 2023-07-27.

[14] *Python Documentation - Turtle Graphics*. https://docs.python.org/3/library/turtle.html. Accessed: 2023-07-27.

[15] *Skulpt*. https://skulpt.org/. Accessed: 2023-07-27.

[16] Jacqueline Staub. “xLogo online - a web-based programming IDE for Logo”. Master Thesis. Zürich: ETH Zurich, 2016. DOI: 10.3929/ethz-a-010725653.

[17] *tkinter — Python interface to Tcl/Tk.* https://scratch.mit.edu. Accessed: 2023-08-19.

[18] *tkinter — Python interface to Tcl/Tk.* https://docs.python.org/3/library/tkinter.html. Accessed: 2023-08-08.

[19] Nicole Trachsler. “WebTigerJython – A Browser-based Programming IDE for Education”. MA thesis. ETH Zürich, 2018.

[20] *W3C.* https://www.w3.org/TR/css-color-4/. Accessed: 2023-08-02.

[21] *Web Workers API.* https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API. Accessed: 2023-08-08.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Turtle Graphics for WebTigerJython-3

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Bogdan	Julia

Ich bestätige mit meiner Unterschrift:
- Ich habe keine im Merkblatt „Zitier-Knigge“ beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

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

Ort, Datum	**Unterschrift(en)**
Zürich,10.08.2023	[illegible]

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