Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

WebTigerPython in Code Expert

Bachelor Thesis

Nils Benz

July, 2024

Advisors: Prof. Dr. Dennis Komm, Dr. David Sichau
Department of Computer Science, ETH Zürich

Abstract

Code Expert is an online IDE that allows students to work on programming tasks without any setup on their local machines. It is planned to open up Code Expert for high schools, where programming often is taught in a playful manner. One Python library that is commonly used in this context is Turtle graphics. WebTigerPython is another online IDE developed at ETH Zurich that focuses on running Python code in the browser and displaying the graphical output.

In this thesis, support for Python programs producing visual output is added to Code Expert using WebTigerPython. The output panel of the WebTigerPython IDE is included as an iframe, while the two websites communicate via the `postMessage` API.

As graphical output is new to Code Expert, there have been no tests that compare images. In the second part of the thesis, a testing template for teachers is therefore created. In order to compare the student submission with the reference image, the Code Expert server calculates the perceptual hashes of the two input images. Perceptual hashes have the characteristic that for similar inputs, the resulting hashes are also close. The teacher can define different thresholds, which defines how far apart the hashes can be while passing the tests.

All these features were integrated into Code Expert.

Contents

Chapter 1

Introduction

Code Expert is an online IDE that allows students to work on programming tasks without any setup on their local machines. It is predominantly used at ETH Zurich, where many courses use the tool to publish exercises and do programming tasks in exams. The future plans of the Code Expert team at ETH are to open up Code Expert for high schools, as there is a large demand for good tools for computer science teaching.

High schools often teach programming in a playful manner using extensions or libraries of existing programming languages. One example of such a library is Turtle Graphics [4], which provides a Turtle that can be moved around the screen and draw various shapes. There are also mechanisms to listen for user inputs, allowing for interactive programs to be created. The goal of this thesis was to include the Turtle output into Code Expert.

1.1 Motivation

There already exist IDEs that work with Turtle graphics out of the box, for example the TigerJython IDE[1], which was developed by the Center for Computer Science Education at ETH Zurich (≪Ausbildungs- und Beratungszentrum für Informatikunterricht der ETH Zürich≫, ABZ). They also implemented a web version of the IDE, WebTigerJython[2] and it's successor WebTigerPython[3] (WTP).

These online IDEs are used in many high schools and even at schools with younger students. An ETH lecturer who frequently visits such schools observed that the workflows of the coding exercises are far from optimal. He stated that in many cases, the teacher publishes a word document with the

[1] https://www.tigerjython.ch/de, 09.04.2024
[2] https://webtigerjython.ethz.ch/, 09.04.2024
[3] https://webtigerpython.ethz.ch/, 09.04.2024

template source code on OneDrive. This file can be downloaded by the students, which then have to copy the code into the IDE where they can edit and execute the code. After finishing their assignments, the students then have to copy the edited source code to yet another word file and upload back to OneDrive. This workflow is suboptimal at best and it is clear that a simpler solution is urgently needed.

WebTigerPython also implements another form of submission. The code can be exported as a search parameter in a link to the IDE, encoded as a base64 string. This might be a slightly better solution than moving around word files, but a simple submit button directly in the IDE would clearly be superior.

Code Expert allows a teacher to create and manage coding exercises online. First, the teacher creates a course and adds exercises with code tasks. After that, they invite the students to enroll. Students can open the tasks in the online IDE without additional setup and submit their attempt. After the submission, automated tests are executed (if specified by the teacher), which provide instant feedback to both the student and the teacher.

As Code Expert solves the issues listed above, it is a natural decision to support Python execution with interactive and beginner-friendly visual output out of the box. This output should be displayed directly in the IDE as opposed to simply providing an image that can be downloaded.

1.2 Related work

We were able to collaborate with the ABZ, which agreed on providing a subset of their existing WTP page, which started with the Master Thesis of Clemens Bachmann [5]. This website can be included into the output panel of Code Expert as an iframe.

1.3 Requirements

The following requirements were developed based on discussions with the Code Expert team in order to ensure that the implemented solution fulfills all criteria. Server-side and client-side execution would both be viable solutions if they fulfill the required security, performance and infrastructure requirements as described below.

Security

There are two parties that execute the code: the students themselves and the teachers. There were no security concerns identified for the students, as they only execute their own code. However, a teacher needs to execute the submitted code, which has to be considered harmful.

There should not be any vulnerabilities due to client-side code execution. This includes cross site scripting attacks in the Python source code, which potentially is executed on a teacher's device. It should for instance not be possible to inject JavaScript code into the Python code, which can then when executed read sensitive information such as the session token from the teacher.

Performance

The visual output should be rendered smoothly. Additionally, it must be possible for several hundred students to simultaneously work on an exercise and get visual output. This is important as Code Expert can also be used for exams, in which case many students work at the same time on the exercises.

Infrastructure

As this feature will be primarily used in high schools and not at ETH, there should only be a reasonable amount of hardware requirements. The Code Expert team has no control over the infrastructure present at other schools, and therefore it is important that the features also works with a suboptimal internet connection.

Testing

As a bonus requirement, it should be possible to define automated test cases. These tests should compare the generated output to a reference image and give basic feedback on how close the submission is to the solution.

Chapter 2

Approach and Evaluation

First, different approaches to solve the problems stated before are compared. The chosen implementation is then described in detail, and finally the results are described and evaluated.

2.1 Where to Execute

As a first step, it is necessary to decide where the graphics output should be generated, on the server or on the client. Therefore, a meeting was organized with the Code Expert team, where this topic was extensively discussed.

2.1.1 On the Server

As of the time this thesis was written, every program that is run with Code Expert is executed on the server and the results are streamed back to the client. Therefore it is natural to have a closer look on what it would mean to execute on the server.

A first obvious advantage is security. On a server, the code is executed in a sandbox and cannot gain access over any critical data. Additionally, if there is a solution to stream a graphical output generated on the server to the client, this immediately opens up the possibility to support other languages such as Java. These languages are very difficult to execute in a browser, as browsers focus primarily on executing JavaScript and WebAssembly. One could possibly write an interpreter for other languages in JavaScript or even compile other languages to JavaScript or WebAssembly. This process would be quite complex, but there exist tools that could be used. Additionally, such an approach might not be as performant as native JavaScript.

On the other hand, streaming the output to the client might be very challenging. First of all, there must be enough bandwidth to serve the resulting images or videos. A separate video stream for each student might not be feasible, so

there would have to be some other solution such as sending keyframes to the client and do some animation directly in the browser. Another possibility that was brought up is to use a dedicated graphic instructions stream.

A further disadvantage of this is that the output will not feel very interactive. This might be an issue, especially since WTP supports input events from a mouse or keyboard.

2.1.2 On the Client

The advantages of executing the code locally in the browser are obvious: there is much less strain on the network and the output will be as responsive as possible.

But there are also some negative points. The implementation is much more susceptible for security issues as it is not immediately clear how to execute the potentially malicious code in a safe way. Additionally, this approach only works for programming languages that can be executed in the browser. For Python, this is the case as CPython code can be ported to WebAssembly using Pyodide[1].

2.1.3 Chosen Approach

In consideration of the above described advantages and disadvantages, it was decided that a client-side solution probably is the most sensible implementation. One of the key decision points was the performance benefit of this solution, as the internet connection will only play a minor role in the smoothness of the visual output.

Clemens Bachmann from the ABZ offered to work on a modification to WebTigerPython, such that it can be included in an iframe. With this, the security requirements could be fulfilled, as described in Section 2.3.3. Additionally, this enables the usage of an already battle-tested system that just works given some Python code. A further benefit of this approach is that WTP supports not only Turtle graphics, but also for example Matplotlib and gpanel.

2.2 Python in the Browser

For the original WebTigerJython [7] the library Skulpt was used. As alternatives, Brython and Pyodide can be utilized. After they had evaluated all solutions, the WTP team decided to switch to Pyodide. This had different reasons, some of which were:

[1] https://pyodide.org/, 30.04.2024

- Pyodide (and also Brython) supported Python version 3.11, which at the time was the most recent version. Skulpt on the other hand was still stuck on Python version 2.6.
- With Pyodide it is much easier to adapt CPython libraries. This is due to the fact that Pyodide is a port of CPython to WebAssembly/Emscripten extended with a JavaScript interface. While per default, also only pure Python packages work, it is much easier to port CPython packages.

WebAssembly is a binary instruction format that works in every modern browser. There are compiler toolchains that compile many high-level programming languages to WebAssembly. One such compiler toolchain is Emscripten, which compiles C code.

Pyodide can then be executed in a web worker and communicate via messages with the main thread, where the JavaScript code of the website is executed. The main thread can initialize, run or interrupt Pyodide in this manner.

Pyodide still has some limitations, one of them being that some packages such as Tkinter are not supported. Hence also libraries that depend on these packages, one of which is the native Python Turtle, don't work in Pyodide. In WebTigerPython, the Turtle library was reimplemented with a canvas backend, which renders the output using PixiJS.

Additionally, Pyodide runs Python code approximately 3-5 times slower than CPython. C code that is compiled to WebAssembly is only 2-2.5 times slower. This slower execution is partly caused by the web worker being single-threaded, which makes parallel execution impossible [5].

2.3 Iframe Communication and Security

An iframe is an HTML element that allows the user to embed another HTML page into the current one. The URL of the embedded page can be specified by setting the `src` attribute of the iframe. Using JavaScript, one can then obtain a reference to the window object of the embedded page:

```
<iframe src="https://webtigerpython.ethz.ch" />

<script>
  const iframe = document.querySelector("iframe").contentWindow;
</script>
```

JavaScript `Window` objects can communicate using the `window.postMessage()` method. This is possible if one page spawns a pop-up, or if a page embeds an iframe. This works in contrast to the standard where different pages are only allowed to access each other if they share the same origin.

2.3.1 Sending a Message

A window can obtain a reference to the window that opened it for example using the `window.opener`[2] or the `window.parent`[3] method. The call signature is then

```
window.postMessage(message, targetOrigin);
```

The `message` can be any JavaScript object that can be serialized. The `targetOrigin` specifies what the origin of the receiving window must be. This can either be an URI or the literal string `"*"`. If the `targetOrigin` does not match the actual origin of the target, the event will not be dispatched. This is a security measure to make sure that confidential information is only sent to the recipients that it is actually intended for.

2.3.2 Receiving a Message

If a window expects to receive messages via the `postMessage` API, it can subscribe to those events on the window object. The event that is provided in the callback function has three properties:

- `data`: the object that was passed from the sending window as payload
- `origin`: the origin of the sending window
- `source`: a reference to the sending window object

In code, this could look the following (example based on the MDN Web Docs[4]):

```
window.addEventListener(
  "message",
  (event) => {
    if (event.origin !== "https://example.com") {
      return;
    }

    console.log(`Message received: ${event.data}`)

    event.source.postMessage("Hi", "https://example.com")
  },
  false,
);
```

2.3.3 Security

There are many security concerns when considering the `postMessage` API. Since any window can send a message to any other window, it is important

[2] https://developer.mozilla.org/en-US/docs/Web/API/Window/opener, 30.04.2024
[3] https://developer.mozilla.org/en-US/docs/Web/API/Window/parent, 30.04.2024
[4] https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage, 13.06.2024

to check the received messages. If not done correctly, this enables cross-site scripting attacks.

If a website is not expected to receive any messages from other sites, the MDN Web Docs recommend simply not adding an event listener for `message` events.

However, if a message event listener is implemented, it is important to always check the origin of the event. This is demonstrated in the above code example. Additionally, the syntax of the received message should be checked to prevent malicious actions.

When sending a message, it is important to specify an exact target origin (opposed to `"*"`). Otherwise, a malicious site can intercept the data sent by changing the location of the window.

A possible attack in the context of this work would be if a malicious attacker injects JavaScript into the Python code and gets the WTP page to execute the JavaScript. Then, the attacker could read sensitive data such as session cookies. By checking the `origin` attribute and the format of the received message, such attacks can be prevented, as the WTP site is trusted [1].

2.4 Implementation

The feature of adding the WebTigerPython output to Code Expert is fairly isolated, which is why the implementation was rather straightforward. The result is shown in a new output panel that embeds the WebTigerPython iframe. Figure 2.1 shows the result of the work.

In the image, three sections are highlighted:

1. When this button is pressed, the code in the currently open file is sent via the `postMessage` API to the WebTigerPython iframe.
2. When the code is executed, a new output panel is created and can be opened using this tab.
3. The output panel contains the embedded iframe of the WTP site.

When implementing the feature, the following steps were taken:

1. Create a new button in Code Expert that when clicked opens a new output panel.
2. In this output panel, insert an iframe with `src` set to the WTP site. This includes a loading state and should adjust when the window is resized.
3. Send and receive messages via the `postMessage` API.
4. Define the TypeScript types of the different requests and responses. This makes it easier to avoid messages with incorrect types when accessing the API during development.

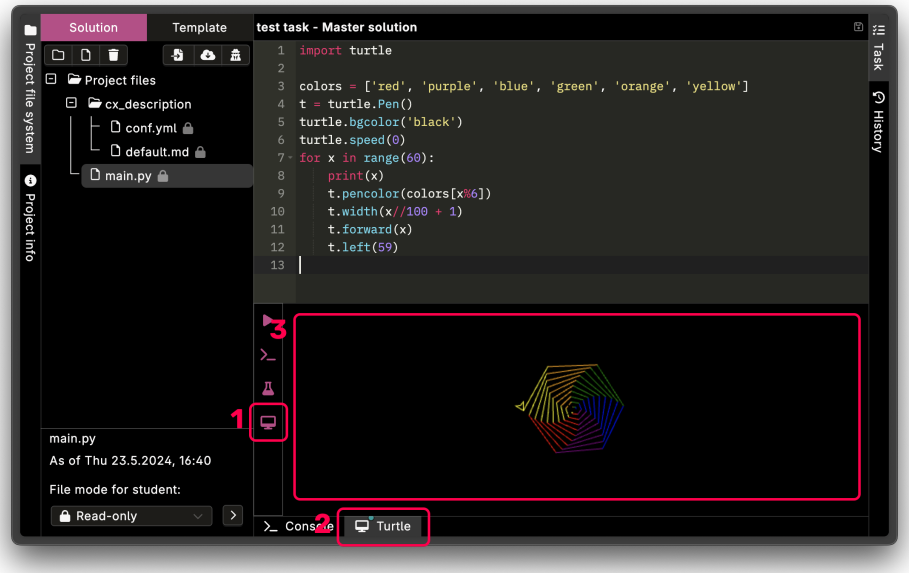


Figure 2.1: The WebTigerPython iframe in Code Expert. Highlighted are the button to execute the code, the panel tab and the output itself.

5. When the execute button is clicked, extract the code in the currently open file and send it to the iframe.
6. Listen for events from the WTP site and send the received output to the console panel. This means that non-visual outputs such as print statements can also be checked directly in Code Expert. Additionally, if an error happens it can easily be inspected.

At the core of the implementation is the synchronization of Code Expert and the WTP iframe. Figure 2.2 shows all the relevant inputs and messages that are necessary to execute a Python program.

2.5 Automated Testing

In addition to the client-side execution with visual feedback, the goal was to enable teachers to conduct automated testing of visual output. The tests should provide basic feedback on whether the generated output is similar to a reference image uploaded in advance by the teacher.

2.5.1 Perceptual Hashing

For the comparison of the images, so-called perceptual hashes were chosen. In contrast to traditional hashes, which have the property that if the input

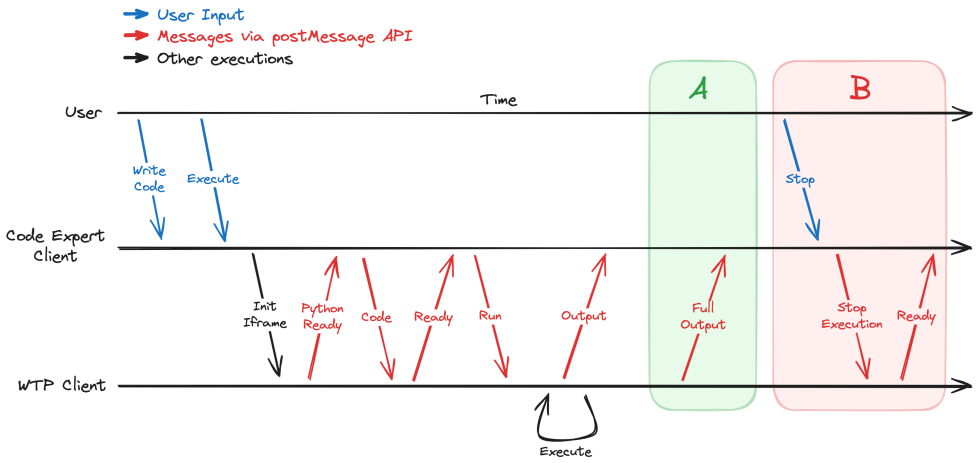


Figure 2.2: The messages sent between the Code Expert client and the WTP iframe. There are different outcomes of an execution, either the program terminates (case A) or the client interrupts the execution (case B).

changes a little, the hash is totally different, perceptual hashes try to assign similar hashes to similar inputs.

Hany Farid [6] lists five desired properties of such a perceptual hash:

- **Distinct.** No two different images should produce the same hash.
- **Resilient.** For similar inputs, the resulting hashes should also be similar. In other words, if the input changes only a little bit, the resulting hash should also only change a bit.
- **Deterministic.** The same input should always yield the same hash.
- **Efficient.** The hash should be cheap to calculate.
- **Non-reversible.** It should not be possible to construct the pre-image of the hash function when only having access to the hash.

In the paper, two different hash versions are presented.

Color-based Hashing

An image can be partitioned into 6×6 non-overlapping blocks. For each of these blocks, the average value of the three color channels (red, green and blue) is computed. These values are stored in a 108-D vector, which exactly is the resulting hash. This is illustrated in Figure 2.3.

These hashes can then be compared. Given two images, for each block i the squared distance is calculated. The r, g and b variables stand for their respective color channel:

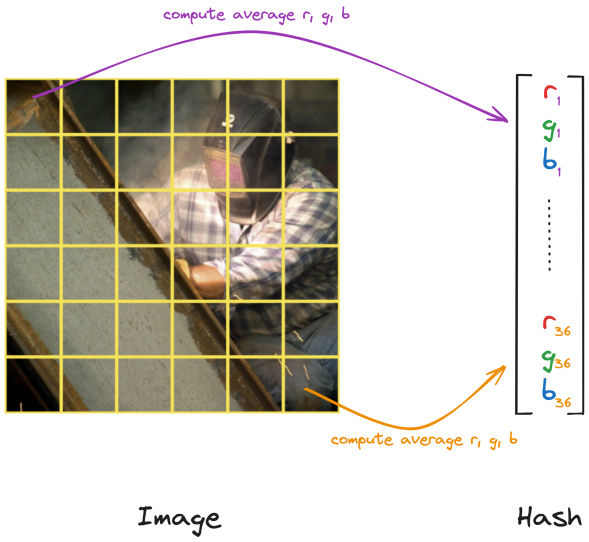


Figure 2.3: The image is partitioned into 36 blocks. For each block, the average intensity of each color channel is calculated and saved into a 108-D vector [6].

$$d_i = (r_{1,i} - r_{2,i})^2 + (g_{1,i} - g_{2,i})^2 + (b_{1,i} - b_{2,i})^2 \tag{2.1}$$

The total distance between the two images is then calculated by adding up all partial distances from (2.1) and taking the square root:

$$d = \sqrt{d_1 + d_2 + \cdots + d_{36}} \tag{2.2}$$

The images can then be compared for equality $d = 0$, or near equality $d < \tau$ for some small τ.

All of the desired properties listed above hold for this hash, except for the resiliency property. The hash is neither resilient to a simple hue shift, nor to cropping or shifting of the image.

Gradient-based Hashing

This hash is an attempt to make the hash more resilient. First, the gradient image is calculated, which results in an image like in Figure 2.4.

A perceptual hash is then extracted from this filtered image: similar to the version above, for each of the 36 blocks the average gradient is calculated (for each color channel separately). These values are again stored in a 108-D vector. To make the algorithm more efficient, it is argued that the image can first be converted into gray-scale, resulting in a 36-D vector.

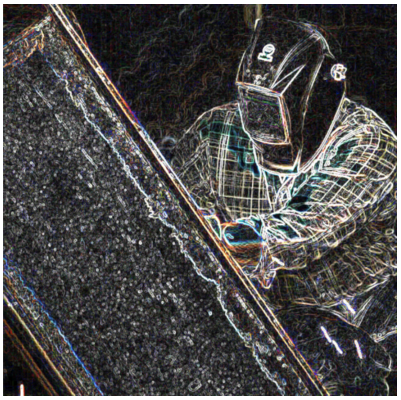


Figure 2.4: The gradient image, where a light color means a transition from a dark to a light region in the original image, and a dark color indicates the position of uniform color patch [6].

Figure 2.5 shows what results the gradient hash produces when comparing an image with distorted versions of the same image, and completely different images. As expected, the hash distance increases with more distortion, but is nonetheless smaller for distorted images than for completely different images.

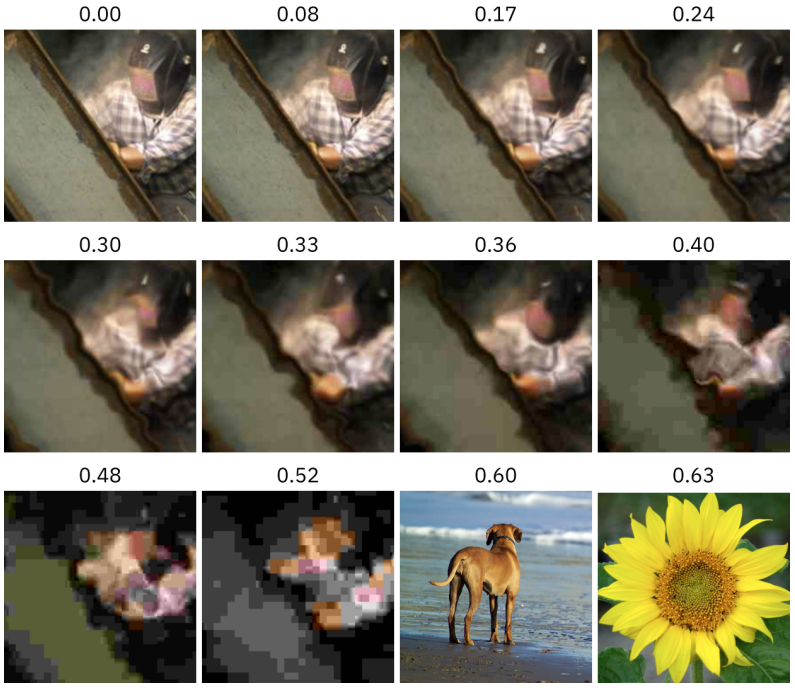


Figure 2.5: The hashes of ten versions of the same image with different amounts of distortion are compared. The resulting hash distance is indicated by a number between 0 and 1 above each image [6].

Perceptual Hashes Used in the Tests

As will be explained below, the `imagehash` Python package was used for the test cases. There, the following three variants were used:

- **aHash**: the image is reduced to 64 blocks. Each block is allocated a bit b_i, which is 1 iff. block i has a color value that is larger than the mean color value (i.e., block i is brighter than the mean brightness).
- **pHash**: similar to the average hash, but instead of comparing the color values, the frequencies obtained by a Discrete Cosine Transform (DCT) are used to calculate the resulting bit for each block.
- **dHash**: calculates the gradient direction by comparing each two neighboring pixels (or blocks) in the grayscale image. The bit b_i is set to 1 iff. the left pixel is brighter than the right pixel.

While aHash and pHash focus primarily on colors and how they are distributed across the image, dHash identifies and compares the structure of the images. The first two can be seen as variations of color-based hashes while the latter clearly is a gradient-based algorithm [2, 3, 6].

2.5.2 Executing on the Server

For the automated testcases, it is essential that the submission image has the same dimensions and resolution as the reference image. Additionally, it is important that also the teacher can see the test results of the students to quickly identify possible problems. For these reasons, it was decided that the images for the tests should be generated on the Code Expert server as opposed to the client.

Code Preprocessing

When the test button on the client is pressed, the source code is sent to the server. As the code is written in an extended syntax of Python that was introduced in TigerJython, it is not immediately possible to extract the resulting image. Hence, the source code has to be preprocessed.

First, the TigerJython-specific code is replaced by the Python counterpart. As WebTigerPython, the Code Expert server only supports `repeat n:` loops from the TigerJython extended syntax. These loops are replaced using regular expressions, the code is included in A.1

After that, the resulting code is prefixed with the necessary imports and the Turtle config such that the output has consistent dimensions. After the submitted code, functions are inserted to retrieve the difference of the hashes of the generated image and the solution. These functions can be called from the test script.

Execution

A problem arose when executing the code. The server does not have a connected monitor, but the Python `turtle` package fails if none is available. This issue can be fixed by executing the script using the `xvfb-run` package[5]. This package uses a virtual frame buffer to simulate a monitor that the Turtle library can use to "paint" the output.

2.5.3 Different Test Cases

Different test cases were prepared, which include an easy, medium and hard baseline. Additionally, it is checked if the output exactly matches the input. For each baseline, perceptual, average and difference hashing algorithms were used. As described above, the hashes have different focuses such as structure and color of the image.

Below, a demo test case is listed. It uses the `imagehash`[6] Python package.

```
import imagehash

hashLength = 12
maxDifference = 5

solutionHash = imagehash.phash(solutionImage, hashLength)
studentHash = imagehash.phash(submittedImage, hashLength)

diff = solutionHash - studentHash
assert diff <= maxDifference
```

First both hashes of the student submission and the reference image are computed. After that, the difference of the hashes is calculated, simply by subtracting the student hash from the solution hash. The test case fails if the difference of the hashes is more than 5.

The `hashLength` and `maxDifference` variables can be adjusted to match a specific project. The longer the hash and the smaller the maximum allowed difference, the harder it is to pass the test case.

2.6 Evaluation

The feature was implemented on a separate branch in the Code Expert repository, so that it is easy to merge the changes into the production branch. This already happened, with some adjustments by David Sichau to improve the user experience and make the changes fit in with the existing code.

[5] https://manpages.ubuntu.com/manpages/trusty/man1/xvfb-run.1.html, 24.05.2024
[6] https://github.com/JohannesBuchner/imagehash, 24.05.2024

2.6.1 Client

The client output works, but there are some issues with the Python execution on the WebTigerPython page. If the run button is pressed before the previous execution has finished, WTP crashes. This error occurs even though in the current implementation Code Expert first stops the execution and waits for WTP to signal being ready again before sending another execute order. Hence, this is likely not something that can be improved on the Code Expert side, and should be fixed by the WTP maintainers.

The security requirement was fulfilled, as there is no risk of cross-site scripting when embedding the iframe and checking all incoming messages.

As far as the performance is concerned, two aspects have to be considered: the start-up time and the Python execution itself. Since WTP executes the Python code using WebAssembly, there is some start-up delay that cannot be avoided. In most cases on an average computer this initialization takes around five seconds. While this is not very fast, the student only has to wait on the first execution. After the WebAssembly code is loaded, all of the code is executed directly on the client. During testing, no slow executions were detected.

While there clearly is a tradeoff between fetching all the WebAssembly code on the client and execution time, the chosen approach is reasonable since the Python execution runs smoothly. This produces a nice output without freeze frames. Additionally, the infrastructure for a webserver that hosts the WebAssembly code is much cheaper than an actual backend server that has to execute all the student submissions and send the results to the clients.

No heavy executions are necessary to display the Turtle output, and the execution and fluent output does not depend on a sufficiently fast network connection. Hence the infrastructure requirement is also fulfilled.

2.6.2 Server

The submitted code can be tested using perceptual hashes. This provides feedback on whether the generated output looks like the reference image, and if not there are different test cases to check how similar the outputs are. An example of this is shown in Figure 2.6.

The server execution is inherently secure as the code is executed in a sandbox and it should not be possible to access any user-specific data from within the sandbox. The scalability of this method was not tested as this was not in the scope of the thesis. But since the server is designed for heavy use in school environments, one can assume that the tests do not pose a risk on performance.

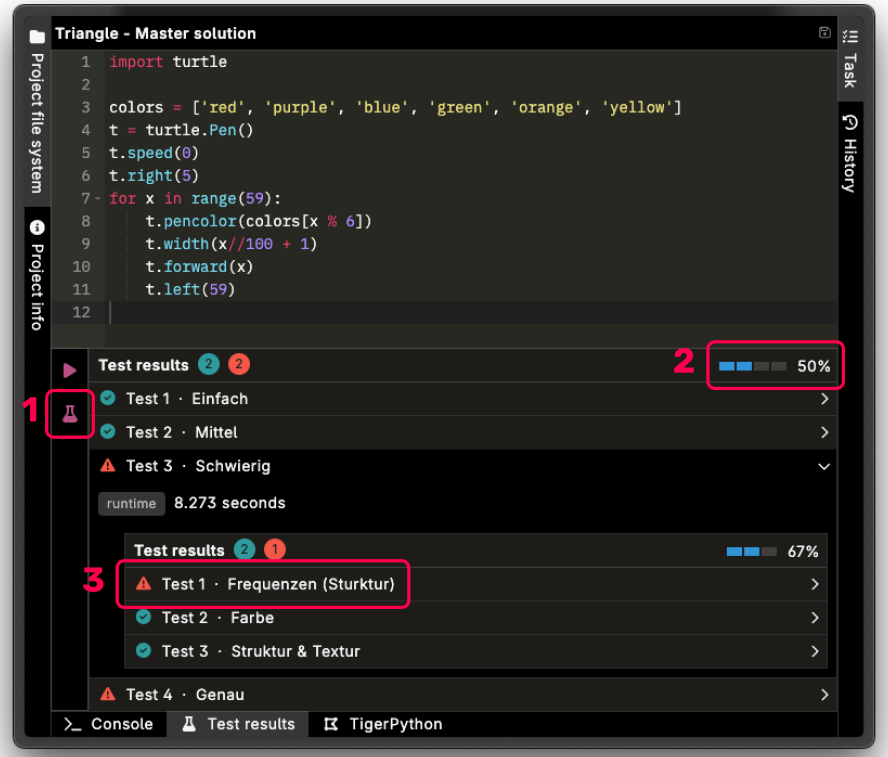


Figure 2.6: The feedback of a test in Code Expert. Highlighted are the button to run the tests, the success rate of the tests and one of the failed tests. One of the hard baseline fails because after the reference image was created, the instruction in line 6 was added. This shifts the resulting image to the right, which means that the submitted image should not pass all tests.

The tests were designed in a way that a teacher easily can modify them to fit a project. This is necessary since for example not all projects fit into a 400×400 canvas.

Chapter 3

Conclusion and future work

3.1 Conclusion

The goal of including WebTigerPython in Code Expert was reached. In the IDE, it is possible to write TigerJython code, even with basic syntax highlighting. Then, the code can be executed using the WTP iframe, while console outputs and errors are logged in the console panel.

The inclusion of WTP went very smoothly, partly due to the generous and swift cooperation of Clemens Bachmann. The execution of the TigerJython code works fairly well, but only when started the first time. There seem to be issues with Pyodide when the code is run again while the previous execution is still ongoing. This likely is an issue on the WTP side, as Code Expert already stops the execution and waits for the WTP page to send a `ready` event before starting over.

At about half time of the thesis, the WTP implementation was included in the production branch of Code Expert and shortly after that released to all teachers. Hence, there was enough time for the second part, which was the automated testing of the submissions.

In the following research phase, we decided to use perceptual hashes for the image comparisons. The goal of the remaining time was to write an example test case for teachers, such that they only have to make slight modifications. There were some troubles with executing the Python code on the server, but in the end we managed to generate the output image and compare it to the reference image.

With the provided tests, the student and teacher get instant feedback on whether the submission is totally different, fairly similar or exactly the same as the solution image. This enables the student to check whether they solved the exercise correctly, while the teacher can identify possible difficulties with the exercises quickly.

3.2 Future Work

Besides the error handling of WTP that must be improved for a smooth user experience, there are some possible steps that could be taken next.

Pyodide Error in WTP

As described above, there are some issues with Pyodide when restarting the execution in WTP. This likely is a bug in WTP, since Code Expert sends a `stop_execution` message to the iframe and starts the next execution only after it received a `ready` message back. Probably, WTP needs to wait for some condition to be met before sending the `ready` message.

Multi-file Projects

The WTP team currently works on multi-file projects. The API already exists, so the Code Expert side could be implemented independently and released as soon as WTP supports multiple files as input. At the moment, only the currently open file is sent to the iframe.

Robotics

TigerJython supports robotics, hence it would be a natural decision to also support this directly in Code Expert. WTP already supports robotics, so it could be investigated whether it can again be used directly or whether an implementation directly in Code Expert is necessary.

User Input

Simple mouse- and keyboard-based events in WebTigerPython already work when the iframe is focused. However, the native Python `input()` that reads from the console is not yet supported.

Appendix A

Appendix

A.1 Code Preprocessing

```
infinite_loop_regex = re.compile(r'repeat *:')
loop_regex = re.compile(r'repeat +(\d+) *:')
loop_regex_no_blank = re.compile(r'repeat\((\d+)\) *:')

def replaceLoops(code):
  # Replace "repeat:" with "while(True):"
  processed_code = infinite_loop_regex.sub("while(True):", code)

  # Replace "repeat <num>:" with "for _ in range(<num>):"
  processed_code = loop_regex.sub(
    lambda m: f"for _ in range({m.group(1)}):", processed_code)

  # Replace "repeat(<num>):" with "for _ in range(<num>):"
  processed_code = loop_regex_no_blank.sub(
    lambda m: f"for _ in range({m.group(1)}):", processed_code)

  return processed_code
```

Bibliography

[1] HTML standard – cross-document messaging. URL: https://html.spec.whatwg.org/multipage/web-messaging.html#web-messaging.

[2] Kind of like that - the hacker factor blog. URL: https://www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html.

[3] Looks like it - the hacker factor blog. URL: https://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html.

[4] Turtle graphics. URL: https://docs.python.org/3/library/turtle.html.

[5] Clemens Bachmann. WebTigerJython 3 – a web-based python ide supporting educational robotics. 2023. URL: https://doi.org/10.3929/ethz-b-000632758.

[6] Hany Farid. An overview of perceptual hashing. *Journal of Online Trust and Safety*, 1, 10 2021. URL: https://tsjournal.org/index.php/jots/article/view/24, doi:10.54501/jots.v1i1.24.

[7] Nicole Trachsler. WebTigerJython – a browser-based programming ide for education. 2018. URL: https://doi.org/10.3929/ethz-b-000338593.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

WebTigerPython in Code Expert

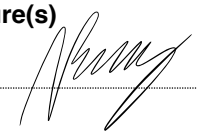
Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Benz	Nils

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
St. Gallen, 10.07.2024	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.