Manuel Menzinger

Design and Implementation of an AI Programming Playground for Schools

Diploma Thesis

to achieve the university degree of

Magister rerum naturalium

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.techn. Gerald Steinbauer

Institute for Softwaretechnology

Laßnitzhöhe, November 2020

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

Acknowledgements

First I would like to thank my colleagues from the *Autonomous Intelligent System* group of the *Institute for Software Technology*, with which I had many interesting discussions significantly influencing the direction of this thesis. Special thanks go to my supervisor Gerald Steinbauer, who provided valuable feedback and support, which in the end allowed me to finish this work in time.

Furthermore, I have to thank my colleagues from the *Klusemann Extern* (*KLEX*), as well as my students, for being understanding in times where I was not as available as usual. I am also grateful for the support and positive feedback provided, that further helped me to steer this work in the right direction.

Finally I want to thank my girlfriend for supporting me through all the stressful times as well as my family, which has always been there for me. Special thanks go my friends, who not only provided valuable feedback, but also arranged well needed distractions, by challenging me with coding problems and playing games together.

Abstract

The personal as well as the professional environment becomes increasingly digitized and the use of Artificial Intelligence (AI) approaches increases. Thus, AI had become one of the most important technologies. To be able to successfully navigate through this digital landscape, profound knowledge of AI, so called AI literacy, becomes increasingly important. While the number of tutorials and courses for teaching AI increased rapidly over the last years, the adoption in school is in its very early stages.

In this thesis we present a framework for AI education in class, which is freely available through a web browser. The goal of the framework is to provide an AI programming playground, that is easy to use and requires no installation. Further, it allows the user to program with fully featured textual programming languages, like *JavaScript* and *Prolog*, as well as to use professional libraries like *TensorFlow*.

Prior to designing the framework a survey was conducted, focusing on how teachers currently teach AI and what might be potential barriers for not including AI topics into their curriculum. Then the framework was designed, with the goal of reducing these obstacles, as well as further potential barriers, while providing engaging programming environments. Modern web technologies were used to create an easily available, reliant and powerful online development environment. Furthermore ready-to-use scenarios and example solutions were provided, helping adoption in class. Finally, a feedback survey was conducted, to review the success of the chosen approach. The result of the feedback supports the chosen approach, as all participants fully agreed that the framework is a useful tool for teaching AI.

Contents

1. Introduction

1.1. Motivation

Our daily lives increasingly depend on information technology. We are surrounded by machines, from smartphones to cleaning robots, which are continuously interacting with their environment in a meaningful way. This can take the form of taking over chores like mowing the lawn, or of helping people to communicate in a foreign language. All this technology has their roots in the field of Artificial Intelligence (AI), in order to make decisions to fullfil their tasks. AI is not only used in nearly any modern electronic device, but it also influences the way we teach, learn and communicate [1, 2]. By shifting the focus to intelligent systems like chat bots [3] and guided online courses, the way we interact while working, learning or communicating adjusts. This disruptive change increases the need for a better understanding of the underlying concepts and systems, not only for being able to successfully navigate this new electronic landscape, but also to be able to compete in a relentlessly changing world of jobs.
As a result, more and more courses and curricula, like the *European Driving License for Robotics and Intelligent Systems* (*EDLRIS*) [4], *Elements of AI* [5] or the *AI for K-12 Guidelines Initiative* [6], emerged focusing on fostering basic AI literacy in young students. Furthermore, the volume of freely available tools and tutorials is steadily increasing, helping students to test and improve their knowledge on various theoretical as well as practical problems. Many of these tasks come in the form of an agent inside a predefined environment. The students then have to create some form of intelligent decision making process for the agent to perform well inside the system.

There are many ways to teach topics of the vast field of AI, ranging from starting a discussion on where machines should be allowed to overtake

human work, to finding mathematical models to describe a problem in an effective way. Another way of teaching is a hands-on programming approach. There students have to create intelligent programs to solve problems inside a given scenario, like manoeuvering an agent safely over a frozen lake, which is one scenario of the *OpenAI Gym* [7]. These programming scenarios are currently either easily accessible online, or downloadable frameworks for offline use. While the online scenarios only require a web browser and a stable internet connection, they often use visual programming languages, which can be a limiting factor when taking on more complex problems, due to their limited support of advanced features like mathematical libraries. On the other hand, offline scenarios offer full access to textual programming languages and powerful libraries but require the installation of tools, an additional barrier which can prevent the use in a classroom setting.
Thus, the obvious next step is to provide easily accessible online tools, where one can use professional programming languages and modern libraries, while removing the need of installing additional components on a local machine.

Advances in web technology and the introduction of web applications have empowered programmers to create complex interactive websites, with access to hardware resources like the graphics cards or webcams. These advancements enabled browser programming frameworks to provide anything from playgrounds for beginners like *Scratch*[1] up to complete development environments like *repl.it*[2], which integrate the entire tool chain from programming to testing and deployment.

The work presented in this thesis contributes to the current online platform ecosystem for AI programming, by using this new technology in order to create an educational AI programming framework. The proposed website provides a powerful and easy to access AI programming environment as well as scenarios of various complexity.

The source code of the application can be found on *GitHub*[3] and is freely available under the open *MIT* license. There is also an instance of the website running on `https://ai.c4f.wtf`.

[1] `https://scratch.mit.edu`
[2] `https://repl.it`
[3] `https://github.com/c4f-wtf/ai`

1.2. Goals and Challenges

The main goal of the work presented in this thesis is to create an online, AI focused programming environment which enables users to use textual programming languages like *JavaScript*[4] and *Prolog*[5], as well as modern libraries like *TensorFlow* [8], to tackle various problems related to AI. It should be possible to solve simple and advanced problems like playing *TicTacToe*[6], manoeuvering an adventurer safely through an unknown cave or helping a bird to fly through obstacles. The initially provided scenarios, seen in Figure 1.1, focus on engaging, agent based games, which are not only appealing environments, but further allow many different approaches to create successful agents.

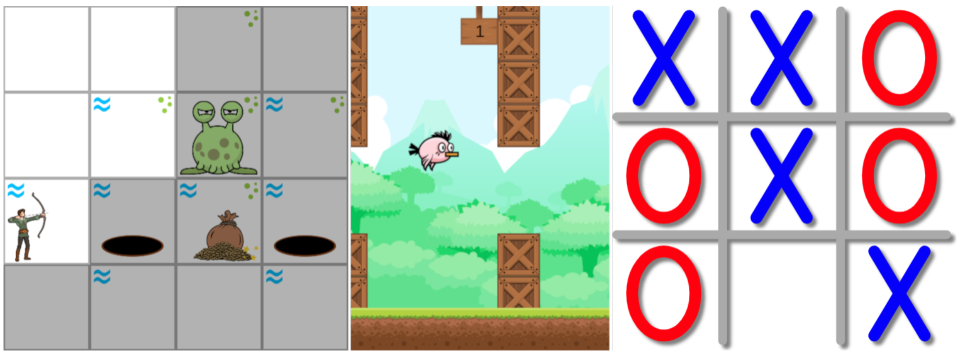


Figure 1.1.: Preview of the provided scenarios. For more information see Section 5.3.

The idea is based on the principle of scenario-based environments which is already successfully used in many different AI teaching environments like the *OpenAI Gym* [7] or *Google's AI Experiments* [9].
Although both tools provide scenarios which require some form of AI programming to be solved, they follow different approaches. On the one hand, *OpenAI Gym* provides a *Python*[7] library full of scenarios for primarily reinforcement learning approaches and requires a local *Python* programming environment. On the other hand, *Google's AI Experiments* provide web-based

[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[5] https://www.geeksforgeeks.org/prolog-an-introduction
[6] https://en.wikipedia.org/wiki/Tic-tac-toe
[7] https://www.python.org

frameworks which can be programmed by placing and modifying given blocks (like *input*, *neural-network*) to solve primarily supervised learning problems and requires nothing else but a modern web browser.

The approach proposed in this thesis combines both approaches by creating a pure web-based programming framework, which provides all the possibilities of a fully featured textual programming language inside a browser. While it is currently possible to run reduced versions of *Python* inside a browser, using *JavaScript* libraries like *Brython*[8], it is not possible to run most *Python* libraries. Despite this problem, there do exist projects which provide full online *Python* programming tools by using a backend server for running the code. An example can be found in Section 2.2.3.

In contrast, the aim of the proposed framework is to not rely on a backend server, since it creates additional dependencies and requires a constant connection to the internet, which is another barrier that can prevent adoption in schools. This leads to *JavaScript* as the programming language of choice. While not as popular in machine learning settings as *Python*, it is a fully featured multi-paradigm language which is supported by all modern browsers, including tooling support like debugging and profiling. There is also an emerging landscape of libraries like *TensorFlow.js*[9], the *JavaScript* port of *TensorFlow*, and *TauProlog*[10], a *JavaScript* based environment for the logical programming language *Prolog*, which help bringing AI to the browsers.

Apart from the technical side, reducing barriers is of huge importance for the project to achieve better acceptance in environments like schools. Therefore, the whole application has to be accessible without requiring any form of authentication or registration and should be capable to continue working even in the event of network disconnects.

8 `https://github.com/brython-dev/brython`
9 `https://www.tensorflow.org/js`
10 `http://tau-prolog.org`

1.3. Outline

In the remainder of this thesis, we start with introducing related research and available frameworks in Chapter 2, to build an understanding of the state of the art. During planning the proposed framework, a survey was conducted to identify the needs of potential users like educators and students. The design, implementation and evaluation of this survey are presented in Chapter 3. Chapter 4 introduces frequently used names and abbreviations, which are common in a web development environment. The general design of the proposed framework, including the core components as well as the provided scenarios, can be found in Chapter 5. Chapter 6 is a detailed introduction into the most important parts of the implementation, including the development tools and used libraries. In Chapter 7 the feedback questionnaire will be discussed and finally in Chapter 8 we provide a conclusion and remarks on future work.

2. Related Work

2.1. Related Research

The digitalization of the world constantly moves forward, therefore, it becomes more and more important to teach children a solid understanding of their surrounding digital environment. At the time of writing, the world is plagued by a global pandemic, forcing people even further into digital dependency. This sudden change into remote teaching highlighted the importance of *e-skills*[1] (skills required to create and interact with *Information and Communication Technologies* (*ICT*)) , as well as the field of computer science in educational settings [10]. Despite the circumstances, teaching and learning using digital tools was more often than not positively received [11]. While these studies are very recent, the positive effect of *e-skills* to perform well in a more and more digitalized society is well known [12, 13, 14].

AI literacy [15] describes the competency to not only know the term AI, but to understand how AI works and in which way it effects our society. As the shift to intelligent systems, like self driving vehicles, customer service bots and personalized advertising continues, AI literacy has become an important part of *e-skills*. The work proposed in this thesis, focuses in fostering these critical skills, by providing environments which allow to experience and create intelligent systems.

In recent years, web technologies became more prevalent, as their capabilities increased. Nowadays websites can be interactive applications offering anything from communication platforms, over interactive learning materials to complete programming environments [16]. This shift further reduces

[1] `https://ec.europa.eu/eurostat/statistics-explained/index.php/Glossary:E-skills`

barriers that could prevent usage of such tools in the past [17], by making them more easily accessible.

The website proposed in this thesis, provides an online development environment, which is easy to use and freely available for everyone. It focuses on learning and teaching different concepts of AI, by providing environments with which the users can interact, using an agent-based approach. The general concept is to provide appealing, easy to use game-like environments, in which the user can focus on improving the intelligent behavior of interacting agents. By providing engaging environments, students can more easily be motivated to learn and improve their knowledge, as similar projects have shown [18, 19].

In the following section, the proposed framework will be compared to similar projects, highlighting key similarities and differences to already available tools on the Web.

2.2. Related Frameworks

Nowadays, there are numerous of online programming frameworks available with very different concepts to address various users. They range from interactive online tutorials like the *w3schools.com*[2] and *learnpython.org*[3], through programming playgrounds like *Scratch*[4] and *OpenRoberta*[5], to complete online development tools like *repl.it*[6] and *Codeanywhere*[7].

While many of these frameworks were used as sources of inspiration and guidance, in this chapter we will discuss three of the available frameworks, which on the one hand are distinctively different to each other, while on the other hand are highly related to the proposed framework. First, each framework will be presented by its general design and functionality, then we will elaborate on which concepts will or will not be adopted in the

[2] `https://www.w3schools.com`
[3] `https://www.learnpython.org`
[4] `https://scratch.mit.edu`
[5] `https://lab.open-roberta.org`
[6] `https://repl.it`
[7] `https://codeanywhere.com`

proposed framework. While each of the selected frameworks are free to use, some do require paid accounts to access additional features.

2.2.1. OpenAI Gym

The *OpenAI Gym "is a toolkit for developing and comparing reinforcement learning algorithms"* [7]. It provides the user with different environments ranging from physical problems, like balancing a pole on a cart, to complex games like *Ms. Pacman*, where the user has to collect items inside a labyrinth while avoiding enemy ghosts. The framework is written in *Python*[8] and is available through the official package manager *PIP*. Furthermore, due to using the *MIT* license, it is completely free and open source and the code can be found on *GitHub*[9].

Design and Functionality

First and foremost, *OpenAI Gym* was created as a framework for implementing and testing reinforcement learning algorithms [20]. In contrast to other forms of machine learning, like supervised learning, reinforcement learning evolves around exploring the environment and adapting the behavior based on given feedback. The *OpenAI Gym* uses an agent-based system (Figure 2.1), where the agent has to decide its next action based on observations regarding the environment and rewards, which indicate progress given a specific task [21, Ch. 2].

The framework itself provides fully functional environments with predefined actions and easily accessible observations and rewards. All environments are implemented using a class with a common interface, making it easy to move from one to the next. As the core of each environment, the *step* method updates the internal state given an action and returns the new observations and a reward, as well as some additional debugging information. Given this information, the agent has to decide which action to take next until the goal is reached or the run is aborted.

[8] `https://www.python.org`

[9] `https://github.com/openai/gym`

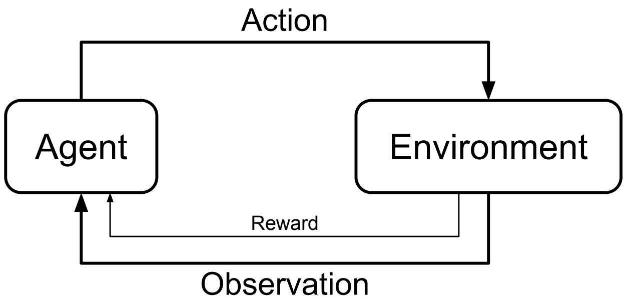


Figure 2.1.: Agent-Environment-Loop

OpenAI Gym comes with dozens of environments, ranging from simple control problems like swinging a pendulum, to complex robotic simulations like pick and place an object using a 5-joint robotics arm. It is even possible to interact with old *Atari 2600*[10] video games like *Space Invaders* or *Ms. Pacman*. In addition, it has many user-created environments which can be installed manually.

While the *OpenAI Gym* itself can easily be installed on any platform using the *Python* package manager *PIP*, some environments require the installation of further components like an *Atari* simulator as well as the *Atari* games themselves.

Similarities and Differences

The *OpenAI Gym* and the ideas of the proposed framework have a lot in common. Both focus on providing environments (scenarios) for the user to enable focusing on the agents decision making using approaches like reinforcement learning. Furthermore, both target a common interface across their scenarios to ease the transition to a new environment.

However, while the *OpenAI Gym* focuses on *Python* and reduces the entry barrier by integrating into the native package manager, with the proposed framework we like to go a step further by requiring no installation at all. This can be achieved by moving the local development environment to a

[10] The Atari 2600 is a gaming console released in the EU in 1978 by Atari, Inc.

browser-based system. This also changes the programming language to *JavaScript*, as it is the only language that works natively inside a browser. On the other side, every modern operating system includes a web browser, therefore no additional installation is needed. Furthermore, due to modern browser technology, the proposed framework is able to function even when offline, which removes the requirement of a constant internet connection.

The switch from *Python* to *JavaScript*, however, has its own drawbacks, as *Python* arguably has more libraries like the *Atari* simulator or *TensorFlow*, a machine learning library. On the other side, *JavaScript* is rapidly evolving and more and more libraries arrive in a *JavaScript* version themselves, like *TensorFlow.js*, which provides a more and more complete version of *TensorFlow* available inside web browsers.

2.2.2. Web Maker

Web Maker[11] is a coding playground, which allows the user to create snippets including *Hypertext Markup Language* (*HTML*) for the layout, *Cascading Style Sheets* (*CSS*) for the style and the programming language *JavaScript* (as well as other comparable web languages) and simultaneously see the result all in one page. Such coding playgrounds are commonly used to test and share code snippets inside programming communities like *Stack Overflow*[12]. They further can be used as simple code editors, as they are easily available using just a web browser. Therefore, they can be used on every modern operating system without installation. *Web Maker* is published under the *MIT* licence and is freely available on *GitHub*[13].

Design and Functionality

The framework provides four windows, one for each of the available languages (*HTML*, *CSS*, *JavaScript*) and one for the resulting page. It also supports templating- and metalanguages like *Markdown* or the *Syntactically*

[11] `https://webmaker.app`
[12] `https://stackoverflow.com`
[13] `https://github.com/chinchang/web-maker`

Awesome Style Sheets (*Sass*) as well as many *JavaScript* libraries. When using any of these higher-level languages, the code gets compiled to basic *HTML*, *CSS* and *JavaScript* whenever the page is updated. Each of these languages can be used in their respective window inside a browser-based editor including modern features like *auto completion*, *syntax highlighting* and *code formatting*. Finally, the resulting page is embedded into its window using an *iframe*, which allows the inclusion of different websites inside a part of the page.

The website generally works without registration, but provides the possibility for users to create a free account to synchronize their projects across multiple computers. Even without an account all projects can be saved locally inside the browser and exported to and imported from a file. In version 4.0, released in march 2019, a *files* mode was introduced to provide the possibility to work with multiple files of any kind, instead of one file per type.

Web Maker was designed from ground up to be resilient towards unreliable network connections. As a result, once loaded the website can be used even when there is no internet connection. While offline, all features work normally except the possibility to login and save and load files to an online profile.

Similarities and Differences

Both *Web Maker* and the concept presented in this thesis focus on an offline first, no installation required and easy to use programming playground. Since the introduction to *files* mode, the experiences are even more similar.

The differences become apparent with the objectives. *Web Maker* focuses on providing users with a web programming editor to play around with small website projects. The aim of the proposed work, however, is to provide the user with working scenarios in the field of AI where the user can program their own intelligent system using *JavaScript* inside a browser. As a result, while *Web Maker* supports multiple web-related languages like *HTML* and *CSS* as well as libraries, the proposed work focuses on *JavaScript* and a *HTML5* canvas for graphical output, which allows real-time rendering of complex scenes like games. Therefore, *Web Maker* does not provide any templates, as it is meant to be a general playground, while the proposed framework centers around ready to use scenarios in addition for users to being able to create their own scenarios from scratch. This differences become more apparent when one is looking into the details. For instance *Web Maker* does not provide the ability to use images other than external hosted ones (which require an internet connection and a place to host the images), which makes it impossible to create graphical scenarios in the same manner as in the proposed work.

More differences can be found in the technical implementation. As the executional context of *JavaScript* inside a tab of a web browser is single threaded, whenever *JavaScript* code is running, the webpage becomes unresponsive as the browser is not able to run the update process. When programming complex algorithms, that have to run for multiple seconds or even minutes, this can become a serious issue. *Web Maker* solves this by using external libraries to parse and manipulate the user code to prevent it from becoming stuck for too long, which makes it impossible to write more complex programs. The solution selected for the proposed framework is to move the user code to a different thread, which can be controlled and aborted externally, but suffering from the consequence that the code loses access to the structure of the website itself and has to communicate over a messaging *API*. A more in depth explanation of the workings of *JavaScript* and the specific implementation details can be found in Chapter 6.

2.2.3. Google Colaboratory

Google Colaboratory[14], or *Colab* for short, is a product from *Google Research*. *Colab* allows anybody to write and execute arbitrary *Python* code through the browser, and is well suited for machine learning, data analysis and education [22]. The framework is built on *Jupyter Notebook*[15], an open-source project for creating documents with included executable code samples, which is used in many areas of data science [23, 24]. Everyone owning a

[14] `https://colab.research.google.com`
[15] `https://jupyter.org`

Google account can create and run *Colabs* for free with the option to upgrade to a *Colab Pro* account for more server resources.

Design and Functionality

The framework is directly integrated into the *Google Drive* environment, with the result that anyone who owns a *Google* account can easily create and share *Colab* documents. A *Colab* document comprises individual cells which can either contain *Python* code or text, formatted using *Markdown*[16], a simple formatting syntax.

Code cells can be run individually, but they share the same scope. This means that later executed cells can access all global variables and functions from previously executed cells. Therefore, code can be split into multiple cells which then have to be executed in order. Each code cell has its own output which can contain text and images like graphs. Web browsers are unable to run *Python* code, therefore the code is sent to a server in the background for execution. As a result, it is not possible to create real-time applications, in particular as the only way to interact with a program is to change its code.

Text cells use *Markdown* to format the content which is converted to *HTML* and *CSS*, which are native structuring and styling languages inside web browsers. The syntax is rich enough to create all common forms of formatting like hyperlinks, tables, images, sections and more. Furthermore, it is a well known syntax used in many applications like *readme* files in source code repositories as well as in some online forums.

Similarities and Differences

While technically quite different, the fundamental idea of creating an easily available web platform with fully featured programming environments out of the box is the same. The same idea applies to the use *Markdown* to incorporate descriptions and information into projects.

[16] `https://daringfireball.net/projects/markdown`

The fundamental differences start with the selection of the used programming language. *Colab* uses *Python*, which in contrast to *JavaScript* is not available inside the browser. Therefore, the code has to be executed on a dedicated server, which results in the requirement of constant internet connection and depends on the availability of the server. This also results in resource shortages in the event of many users executing their code simultaneously. To compensate for the increasing cost of server infrastructure, paid accounts were introduced to finance the background infrastructure as well as to assure users availability. On the other hand, the approach selected for our framework is local execution using *JavaScript*, a language which can be executed natively inside the browser. As a result there is no scaling problem, as every user uses their own hardware. Moreover, the local approach enables direct user input (mouse, keyboard, camera, ...) and real-time as well as offline applications.

The structure and workflow in general are different as well, while *Colab* focuses on a single file which includes the code as well as text for further explanations. The proposed framework uses a directory structure with separate files for code, text and assets. It is worth mentioning, that *Colab* does support a folder structure for asset files, which can also be included using the google drive infrastructure. While the former approach allows to create a very descriptive document explaining a concept while letting the user play around with relevant parts of the code, the latter approach, which was taken in this work, enables the user to create complex programs across multiple files of different types, similar to a traditional local programming environment.

3. Survey - Teaching AI in School

Prior to the development of the proposed framework, a preliminary survey was conducted to collect potential requirements for the framework. Because of the specific target audience for the project, the survey was conducted with teachers who participated in either the *EDLRIS AI Basic* or *EDLRIS AI Advanced* [4] courses before. Therefore, all participants had a basic knowledge about topics in AI and the motivation to teach AI to students. The detailed questionnaire can be found in Appendix A.

3.1. Design and Implementation

The survey was designed as a quantitative online questionnaire, targeting teachers which already have been showing interest in teaching topics of AI. At the end of the survey, there was an option to register with an email address to get notified when an early version of the framework would be available.

Questions were divided in to two categories: (1) teaching AI at schools and (2) technical circumstances. The former comprises questions targeting the topics, approach to and reasons not to teach AI in class as well as tutorials, courses, frameworks and programming languages used at school. To keep the questionnaire short and easy to complete, all questions in this category were multiple choice with the option to add additional remarks. Questions of the latter category were about browser availability and the frequency of updates for programs at school, which is directly related to the available feature set that can be used by the framework.

The questions were designed to get a better understanding of the contents of, and the requirements for teaching AI in class. While there can be many reasons for deciding what to teach in school, the survey focused on identifying barriers, which then could be considered while designing the framework.

For ease of creation and distribution, the questionnaire was created using *Google Forms*[1]. The resulting link was posted on message boards of multiple *EDLRIS AI* [4] courses conducted over the last two years as well as send to specific individuals who stated interest in the project in prior discussions. In order to increase user participation, the posting also mentioned the possibility to get early access to the tool by submitting the questionnaire and providing an email address.

3.2. Evaluation

Over the span of a month, 14 people participated, out of which 12 stated interest in testing the framework by providing a contact email address. In the reminder of this chapter we will present the evaluation of each individual question and conclusions that can be drawn from them.

For readability, the longer labels inside the charts were shortened. The complete questionnaire can be found in Appendix A.

Q1: Which general AI topics are you teaching or planning to teach in class?

While for 86% of all participants the *ethical and social aspects of AI* were the most selected topics, the same number of people stated interest in traditional topics of AI like *search* and *knowledge representation and reasoning*. These topics represent the more classical approaches like finding the shortest path on a road and using logic to make deductions from given facts. With 57% over half of the participants were teaching or planning to teach supervised learning in class. *Reinforcement learning* was the least checked topic with only

[1] https://www.google.com/forms/about

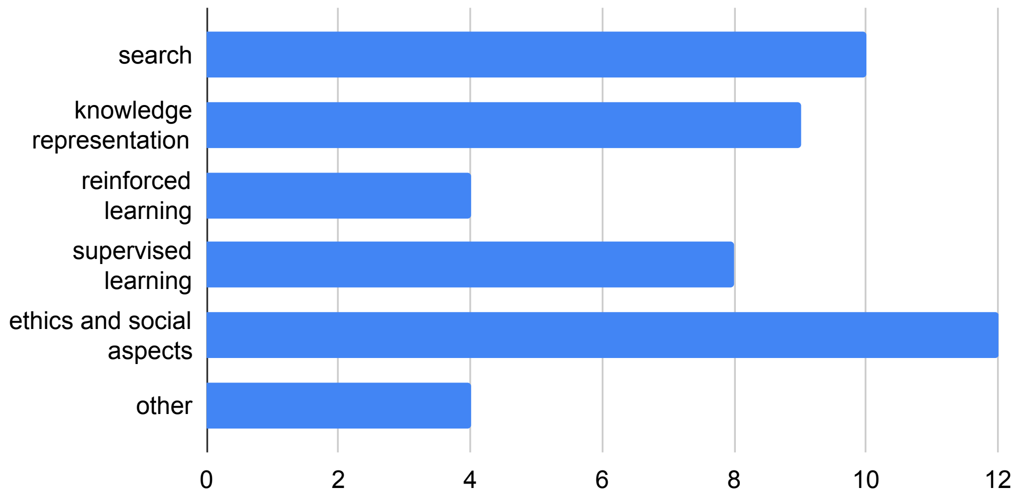


Figure 3.1.: Results of Question 1: Which general AI topics are you teaching or planning to teach in class?

29%. The category *other* contained only clarifications about projects around these topics, like *donkey cars* (autonomous driving cars) and the focus on the difference between artificial and biological neuronal networks.

Q2: How do you approach teaching AI?

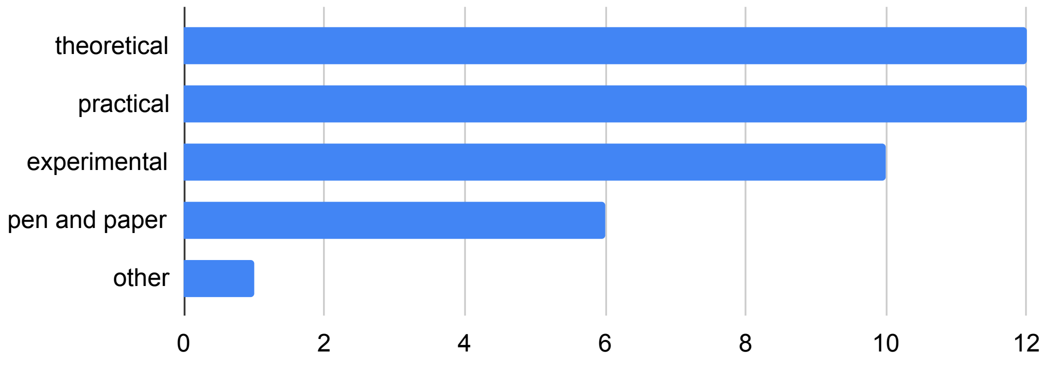


Figure 3.2.: Results of Question 2: How do you approach teaching AI?

With 79% the majority of the participants chose a mixed theoretical and practical approach, with 8 of them also incorporating experimental elements. Two participants focused exclusively on an experimental and practical approach, while 43% of the participants also included pen and paper exercises. One participant also mentioned a project-based approach.

Q3: What are your reasons (if any) not to teach AI in class?

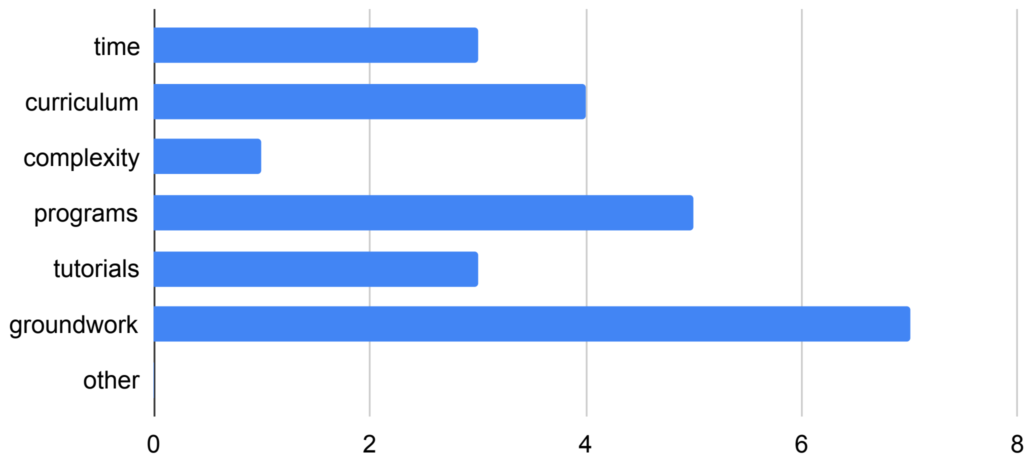


Figure 3.3.: Results of Question 3: What are your reasons (if any) not to teach AI in class?

While, as shown in the questions before, all participants had been teaching or planning to teach some topics of AI, there were still some reasons not to teach (more) AI topics. The main reasons were the required groundwork of creating frameworks to be able to work with more complex examples (50% of participants) and the requirement of installing specific software (36%). Another 36% of the teachers mentioned the lack of time or space to fit more topics inside the curriculum. Finally, four participants checked missing tutorials and courses or the general complexity as reasons not to teach AI in class.

Q4: Are there any AI tutorials/courses or frameworks you have participated in or used yourself?

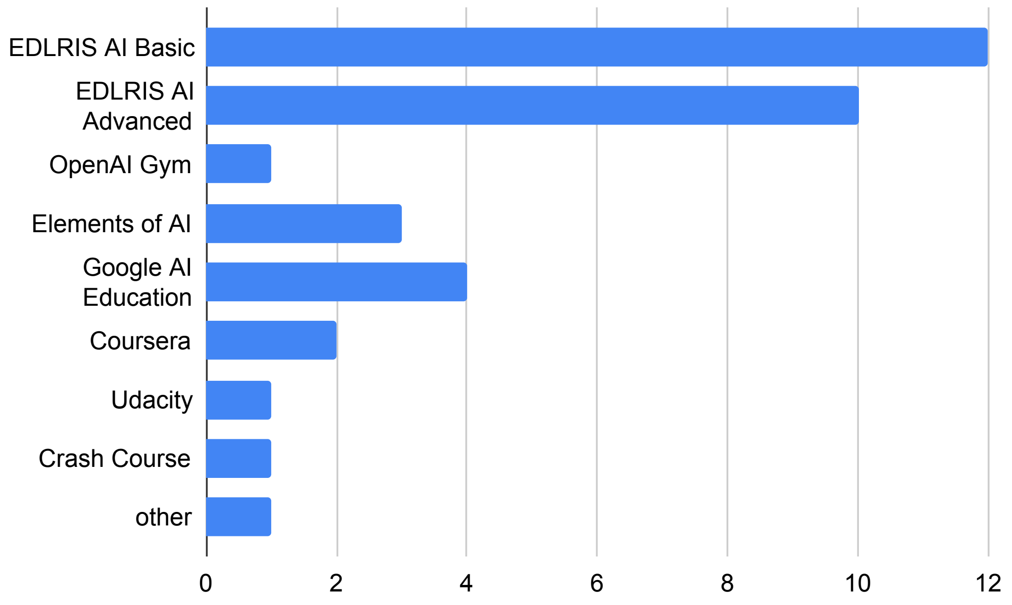


Figure 3.4.: Results of Question 4: Are there any AI tutorials/courses or frameworks you have participated in or used yourself?

Due to the selection of participants, everyone participated in either the *EDLRIS AI Basic* or *AI Advanced* course[2]. The next popular choice with 29% was *Google AI Education*[3] followed by *Elements of AI*[4] with 21%. Other courses on popular platforms like *Coursera*[5], *Udacity*[6] and the machine learning lectures of the *Johannes Kepler University Linz*[7] as well as the *OpenAI Gym*[8] framework were all together only used by 21% of the participants.

[2] `https://edlris.ist.tugraz.at/modules`
[3] `https://ai.google/education`
[4] `https://www.elementsofai.com`
[5] `https://www.coursera.org`
[6] `https://www.udacity.com`
[7] `https://www.jku.at/en/institute-for-machine-learning`
[8] `https://gym.openai.com`

Setting aside the *EDLRIS* courses, the net two popular choices are *Google AI Education* and *Elements of AI*, both online courses with a low bar of entry due to full availability inside the browser. While neither course has a dependency on external programs, both require user registration to be fully available as well as a constant internet connection.

Q5: Are you using or planning to use parts of any AI tutorial/course or framework in class?

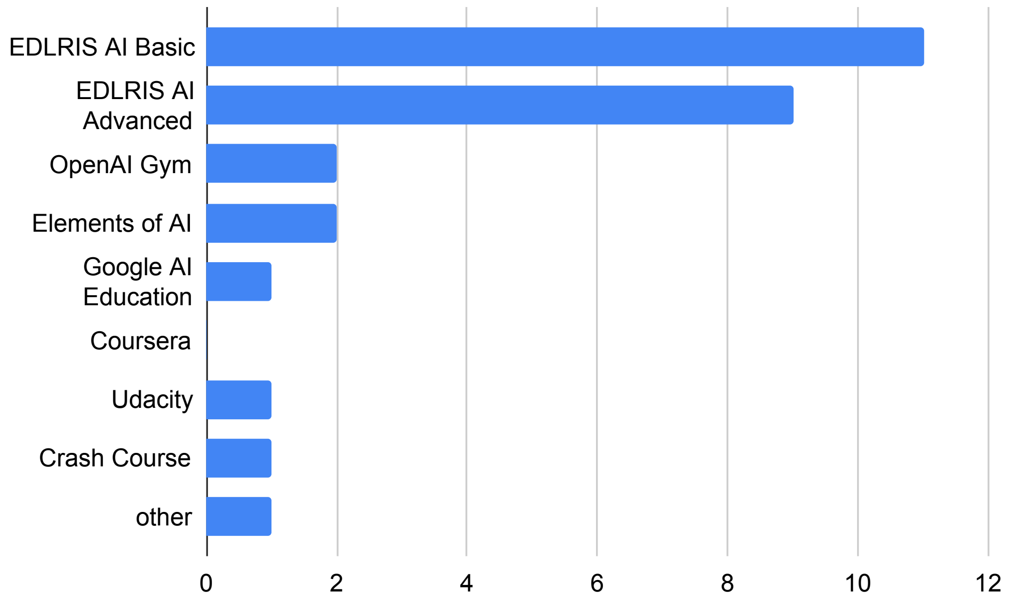


Figure 3.5.: Results of Question 5: Are you using or planning to use parts of any AI tutorial/course or framework in class?

It is not surprising that 86% of the participants used or were planning to use parts of the *EDLRIS* courses, as all participants are teachers who already took part in at least one of these courses. While 43% of the participants chose one or more of the other available courses and frameworks, one mentioned heavy reliance on a custom made *OneNote* course. Only one participant selected no answer signaling he or she would not plan to use AI frameworks or tutorials in class.

Q6: Which programming languages and libraries are you teaching in class?

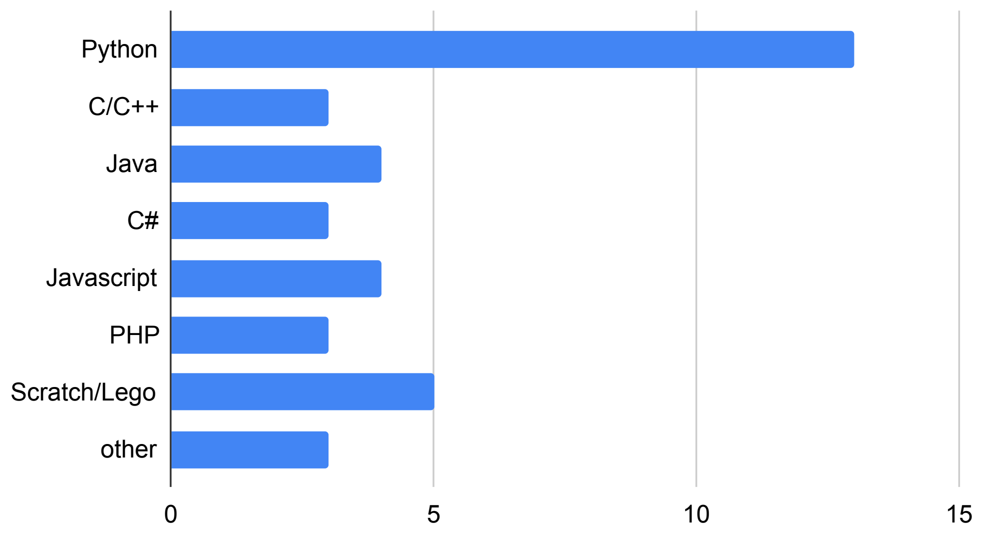


Figure 3.6.: Results of Question 6: Which programming languages and libraries are you teaching in class?

The most popular programming language is *Python*[9], which is used by 93% of the participants. It is followed by graphical programming languages like *Scratch*[10] and *Lego Mindstorms*[11] with are used by 36%. 79% of the participants selected at least one of the other languages including *Orange*[12], *OpenRoberta*[13], *Keras*[14] and *PyTorch*[15].

It is evident, that even if *Python* is the most used language, there are still many other programming languages in use, as each proposed language was selected by at least three participants. The prevalence of *Python* comes

[9] `https://www.python.org`
[10] `https://scratch.mit.edu`
[11] `https://www.lego.com/en-at/themes/mindstorms/about`
[12] `https://orange.biolab.si`
[13] `https://lab.open-roberta.org`
[14] `https://keras.io`
[15] `https://www.pytorch.org`

naturally due to being the language of *TensorFlow* [8], one of the most popular machine learning libraries as well as the beginner's language of many programming courses.

Q7: Which web browsers are available in your school?

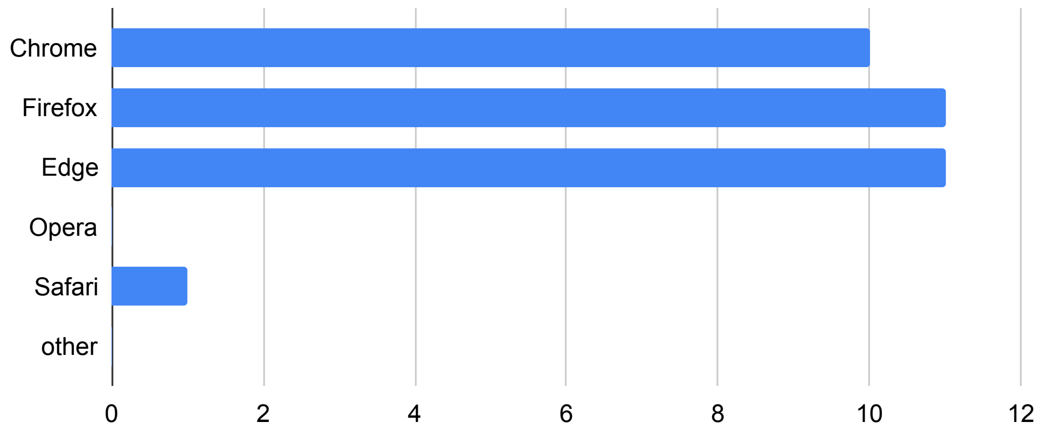


Figure 3.7.: Results of Question 7: Which web browsers are available in your school?

While all participants answered that one or more modern browsers are available in class rooms, two stated that only *Firefox*[16] is available. This is worth mentioning, as at the present the proposed framework does not work inside *Firefox*, due to the lack of support for some modern features. For more information, see Section 6.4.

Q8: How often are your programs (especially web browsers) updated in school?

In at least 71% of the participants schools, the browsers get updated at least once per year, 21% did not know their update cycle. Only one stated that the browsers would get updated every other year. Due to the frequent

[16] `https://www.mozilla.org/en-US/firefox/`

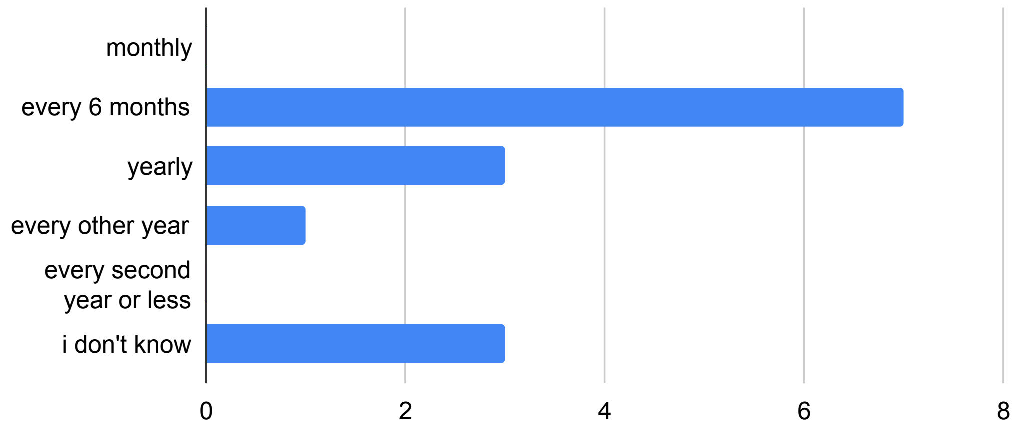


Figure 3.8.: Results of Question 8: How often are your programs (especially web browsers) updated in school?

update cycles in the participants schools, new features can be expected to be available at the latest after two years.

3.3. Conclusion

The survey highlights, that a practical approach is generally considered important, but also harder to integrate in to a class setting. One of the main reasons for not taking the practical route is the requirement to install specific programs and the absence of frameworks, which provide ready-to-use environments, as can be seen in Figure 3.3. These frameworks help with the groundwork of creating scenarios, where the user can test their ideas without writing all the surrounding mechanics themselves. While there are projects like the *OpenAI Gym* [7], they still require the installation and configuration of tools, which can explain the low adaptation rate as seen in Figure 3.5.

In Figure 3.1 it can be seen that from all AI topics, *reinforcement learning* is the least taught one. While *reinforcement learning* in general is a more advanced topic, the requirement of a framework providing the environment for a system to learn in, is an additional diminishing factor.

In summary, the survey illustrates that although there are a lot of resources already available, there still is need for more. Specifically in the area of *reinforcement learning*, new tools with fewer barriers can enable the integration of the subject in to a class setting.

4. Prerequisites

In this chapter we introduce the basic terms required when working with web technologies. It contains a list with short descriptions of frequently used names and abbreviations. The goal of this chapter is to familiarize the reader with the terminology used in the next chapters. This is especially important as some of the terms often get mixed up like the *World Wide Web* and the *internet*.

If not stated otherwise, the terms are either common knowledge or standardized by the *World Wide Web Consortium*[1] (*W3C*), an international community that develops open web standards. For more detailed information we direct the reader to the *MDN Web Docs* [25], an excellent resource of guides, references and general documentation about web technologies.

4.1. Frequently used Names and Abbreviations

API

An *Application Programming Interface* (*API*) defines the interaction possibilities between multiple entities. It specifies the exact data formats that have to be used for communication.

Application

An *application*, or *app*, is commonly known as a software program that provides some functionality to the user, like email clients and web browsers. In recent years the term *web application*, or *web app*, became popular for websites that provide the same functionality while being accessible using a web browser. As websites become more and more

[1] `https://www.w3.org`

complex, the distinction between *websites* and *web apps* becomes blurry und the terms are often used interchangeably.

Browser
A *browser*, or *web browser*, is a program for accessing the *World Wide Web*.

CSS
Cascading Style Sheets (*CSS*) are used to apply styles to *HTML elements* and are therefore used to produce the design of a webpage.

DOM
The *Document Object Model* (*DOM*) is a interface to programmatically modify the structure and style of a document. In this thesis it always refers to the *HTML DOM*, which enables *JavaScript* to modify the content and design of a webpage.

HTML
The *TyperText Markup Language* (*HTML*) is used to structure the content of a webpage using descriptive *elemets* like *paragraph*, *button* and *hyperlink*. It is the basic building block of any webpage.

HTTP
The *HyperText Transfer Protocol* (*HTTP*) is an application layer network protocol and the foundation of the *World Wide Web*. It is a stateless protocol and communication takes place using *request* and *response* messages, which include all required information. In current times, mostly the encrypted variant *HTTPS* is used.

Hyperlink
A *hyperlink*, or just *link*, is an *HTML element* that uses *URLs* to connect to resources on the *World Wide Web*. It is commonly used to create a connection between different webpages.

Internet
The *internet* is a global network connecting devises all over the world. It is used for many purposes like video conferences and messaging as well as the *World Wide Web*.

JavaScript
JavaScript is a multi-paradigm, dynamically typed programming language, which is available in all modern web browsers. It supports many imperative as well as declarative concepts and is therefore the programming language of the Web. While the *W3C* specifies the *APIs* that are available inside a browser, *JavaScript* in general is based on

the *ECMAScript* specification from *ECMA International*[2], a standards organization for information and communication systems.

Page
: A *page*, *web page* or *webpage*, contains all information to display content associated with an *URL*.

Server
: A *server*, or in the context of this thesis a *web server* is a computer, connected to the internet, that provides different functionality to other devices called *clients*. This includes handling *HTTP requests* and providing webpages as well as storing user information and user authentication.

Site A *site*, also *web site*, or *website*, is a collection webpages which are usually connected using *hyperlinks*.

State / Store
: A *state* commonly describes the present condition of a system. The term *store* is used to describe objects that store information about the *state* of a system.

URI / URL
: *Uniform Resource Identifiers* (*URI*), commonly also known as *Uniform Resource Locators* (*URL*) and *web addresses*, specify the location of resources on a network. They come in the form of a single line string and contain the scheme, the domain as well as a path and additional queries (*scheme://domain/path?query*).
For example: *https://www.google.com/search?q=URI*

WWW / Web
: The *World Wide Web*, also known as *the Web*, is the biggest information system on the internet. It allows to share resources like webpages using URLs which are transferred using the *HyperText Transfer Protocol* (*HTTP*). The Web can be accessed using software called *web browsers* and content can be published by using *web servers*.

[2] `https://www.ecma-international.org`

5. Design

This chapter describes the general design philosophy and principals used for the core framework, as well as provides an overview of the provided scenarios. Implementation details can be found in Chapter 6.

5.1. Core Framework

5.1.1. General Structure

In general, web applications can be categorized into two categories: (1) *multi-page* and (2) *single-page*. Traditionally, in web development, the *multi-page* approach was used to structure the content. In this approach all the information is split over multiple pages. The user can then use hyperlinks to navigate from one page to another, which is requested from the server and rendered in the browser by reloading the window. While this approach has existed since the beginning of the *World Wide Web* [26], it is still widely in use, as can be seen in popular examples like *Amazon*[1], the website of the *Graz University of Technology*[2] and all the websites build on frameworks like *Wordpress*[3] and *Moodle*[4].

However, there are some drawbacks when using the *multi-page* approach. Whenever a new page is requested, the whole page has to be loaded from the server which likely contains code duplication in the form of elements that have not changed. Furthermore, after each request the browser reloads the

[1] https://amazon.com
[2] https://tugraz.at
[3] https://wordpress.com
[4] https://moodle.com

window to display the new content, throwing away any state information that was not explicitly saved (for instance by using cookies or a local database).

A newer approach is to use a single page. While visually indistinguishable, the approach is quite different internally. Whenever the user requests a new page, only the relevant content is sent over the network. Then *JavaScript* is used to replace the corresponding parts of the website without reloading the window. This approach not only reduces the data required to be transferred over the network, but it can also sustain state information, as the window is not automatically reloaded. The drawback of this approach is the dependency on *JavaScript* to be able to load and interject the new content. When a user disables *JavaScript* for any reason, this approach can not work and fallbacks have to be provided. Still, *single-page* applications rise in popularity as can be seen in popular examples like *Youtube*[5], *Twitter*[6] and *Facebook*[7].

As the proposed framework requires the execution of user code, which has to be loaded by script anyway, this capability is extended to the entire website and a *single-page* approach is used. Furthermore, this framework makes heavy use of *JavaScript* to store complex states, like the multiple files the user is currently editing, which would be thrown away and had to be loaded again whenever the user switches to a different section of the site temporarily. The state still has to be stored locally however, as the user can always decide to reload the window manually, after which it has to be restored.

Even though the website technically just has a single page, the term *page* is used from here on forward to describe a unit of content used to structure the website like *project-page* and *welcome-page*.

5 `https://youtube.com`

6 `https://twitter.com`

7 `https://facebook.com`

5.1.2. Code Execution

By default, web applications run only in a single thread called the *main thread*. It has access to all available features like the *Document Object Model* (*DOM*), local databases (*localStorage*, *indexedDB*), *WebGL* and more. Although this approach can be quite convenient, it has some severe drawbacks:

- There is only one execution unit. When any part of the application performs anything computational intensive or uses a blocking function, the entire window becomes unresponsive.
- All resources can be accessed from anywhere inside the code without restrictions.

While both drawbacks can be avoided by programming carefully and by making use of asynchronous methods, they are fatal in the context of this thesis, where the goal is to execute arbitrary code generated by the user. For instance, if the user adds an infinite loop by accident, the entire page should not become unresponsive.

To solve this problems, web workers were introduced. They are separate execution units with their own *JavaScript* scope and no access to the *DOM*. While they still have access to the local database called *indexedDB*, the only way of communication with other threads is through messages. Web workers can execute independent code and are directly controlled by their creating thread, which can start and stop them at any point. Nowadays, web workers are supported by all modern browsers[8].

In this application, a web worker named *scenario-worker* is used for the execution of user generated code and the scenario itself. It is created and controlled by the *main thread* and communicates with it through well defined messages. Still, the main part of this application resides on the *main thread*, including the entire *Graphical User Interface* (*GUI*) as well as the control over the different workers and the *indexedDB*. It contains the state of the website and stores user generated code inside the database.

The connection to the network is controlled by the *service-worker*, which is a special kind of worker that runs in the background and can even persist

[8] `https://caniuse.com/#search=web%20worker`

when the website is not loaded. It can intercept all network requests (*HTTP requests*) and creates caches from network responses for speedup and offline usage. Furthermore, the *service-worker* enables access to user generated code by loading it from the local database and providing it as a network response to the *scenario-worker*. This approach has the additional advantage that the user code is treated like any other code, which allows the debugging tools, which every modern browser provides, to function properly.
Figure 5.1 illustrates the connection between the main components.

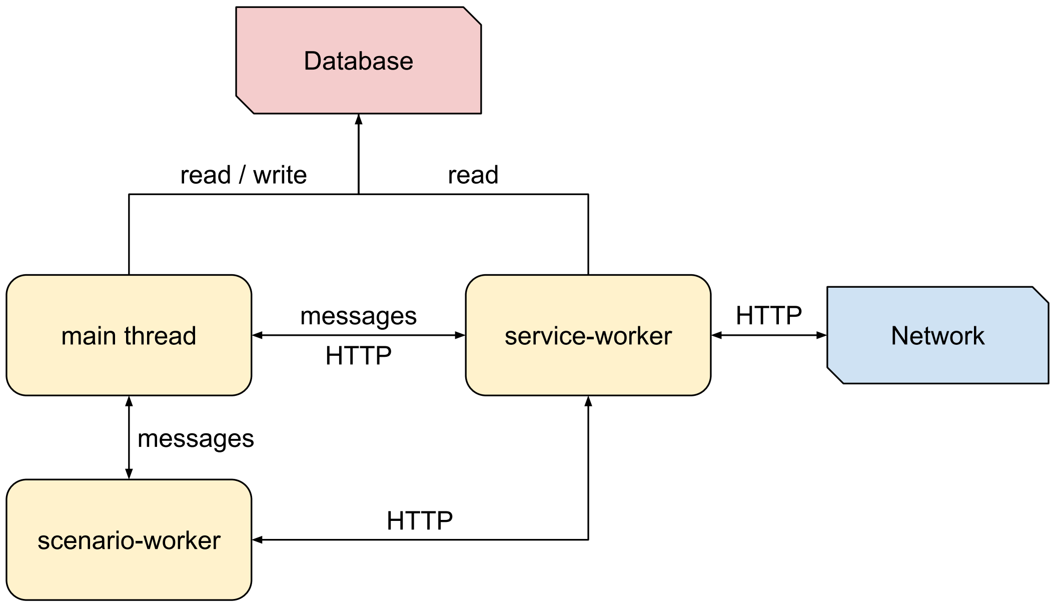


Figure 5.1.: Overview of the main components. The yellow components are separate execution units, which communicate using messages. Furthermore, all *HTTP requests* are intercepted by the *service-worker*, which has access to the database to provide user generated code as *HTTP responses*, as well as for caching. The *main thread* acts as the main controller and is the only component that can modify the database.

5.2. Components

This section discusses the design decisions made for each of the three main components. Further implementation details can be found in Chapter 6.

5.2.1. Main Thread

The *main thread* represents the basic entry point to the application. Its purpose is to control the *User Interface* (*UI*), manage the general state of the website as well as to start, stop and communicate with worker threads.

User Interface

The general structure of this framework is build on the *Polymer Project* [27], which provides open-source libraries for the use of a new web specification called *Web Components*. It was created and released by *Google* in 2015 under a permissive *BSD* license. Furthermore, it is used in many of their most prominent products like *Youtube* and *Google Maps* as well as in other popular sites like *GitHub*[9]. *Web Components* allow the programmer to not only group code (structure, style and behavior) by page but also by element inside a page. Normally, in a *single-page* application style definitions and behaviors are applied over the entire page and require careful coding to avoid overlapping. With *Web Components*, styles and behaviors can be limited to the component itself without generating undesired side effects.

In this application, *Web Components* in the form of *LitElements* from the *Polymer Project*, are used for all major elements as well as some smaller reusable parts and all the different pages. *LitElements* are classes to easily create and manage *Web Components*. The following list gives an overview of all components and how they are used:

tab-group
A reusable component which provides the ability to use tabs to display multiple elements inside the same space. Only one tab can be active at any time.

dynamic-split
A reusable component that splits the available space in two using horizontal or vertical separators, which can be dragged within given limits. The current position for each separator is stored locally inside

[9] `https://github.com/Polymer/polymer/wiki/Who's-using-Polymer%3F`

the browser which allows it to persist after reload. The component can be nested to create any kind of grid layout.

simulator
The *simulator* controls the interaction with the user code. It therefore has buttons to start or stop scenarios. Furthermore, it provides canvases for graphical output.

bug-tracker
The *bug-tracker* makes it easy for the user to report bugs by providing a simple form which can sent the bug description, in addition to some information about the current state of the website, to a server.

console
The *console* consists of a basic output console which also includes file and line number information of where to find the corresponding call as well as a link to directly open the file.

editor
The *editor* is build on the open-source editor *Monaco* [28], which is created by *Microsoft* and released under the *MIT* license. It provides the user with a powerful code editor, including many features like *syntax highlighting*, *code completion* across multiple files and many more.

modal
The *modal* component is a generic element to display popup-boxes for user interaction. It is used whenever the user has to provide some information, like the name for a new file.

file-tree
The *file-tree* lists all files corresponding to the active project in an extendable tree form. Furthermore, it includes additional documentation files, explaining the available libraries.

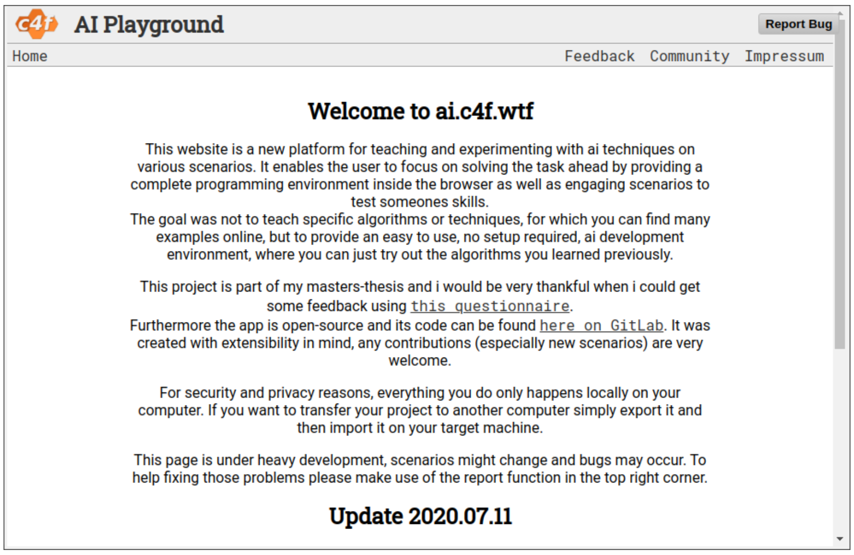
app
The *app* component groups the entire application together and represents the main entry point for the website.

header
The *header* component represents the top bar of the webpage and contains basic navigation links as well as the *bug-tracker*.

Additionally, there are four components representing different pages. All pages contain the header component for basic navigation:

news

The page containing the latest news and updates. This component is shown when the user first visits the website or when there was an update since the last visit. See Figure 5.2.



Welcome to ai.c4f.wtf

This website is a new platform for teaching and experimenting with ai techniques on various scenarios. It enables the user to focus on solving the task ahead by providing a complete programming environment inside the browser as well as engaging scenarios to test someones skills.
The goal was not to teach specific algorithms or techniques, for which you can find many examples online, but to provide an easy to use, no setup required, ai development environment, where you can just try out the algorithms you learned previously.

This project is part of my masters-thesis and i would be very thankful when i could get some feedback using this questionnaire.
Furthermore the app is open-source and its code can be found here on GitLab. It was created with extensibility in mind, any contributions (especially new scenarios) are very welcome.

For security and privacy reasons, everything you do only happens locally on your computer. If you want to transfer your project to another computer simply export it and then import it on your target machine.

This page is under heavy development, scenarios might change and bugs may occur. To help fixing those problems please make use of the report function in the top right corner.

Update 2020.07.11

Figure 5.2.: Welcome and news page

project-index

The page displaying an overview over all projects. Each project is represented by its title and logo, as shown in Figure 5.3. Furthermore, there are options for exporting/importing each project as well as creating new and deleting existing projects.

project

The page displaying all elements required to work on a project. To the left there is a *file-tree*, which contains not only project files but global files, which can be shared across multiple projects. Additionally, documentation files can be found in the *docs* folder at the top of the *file-tree*. In the center there is a fully featured code editor as well as a *Markdown* viewer to display documentation files in a well formatted way. The simulator is to the right including the controls for starting and stopping a scenario as well as all the visual output from the scenario. In the bottom left the console output can be found, including

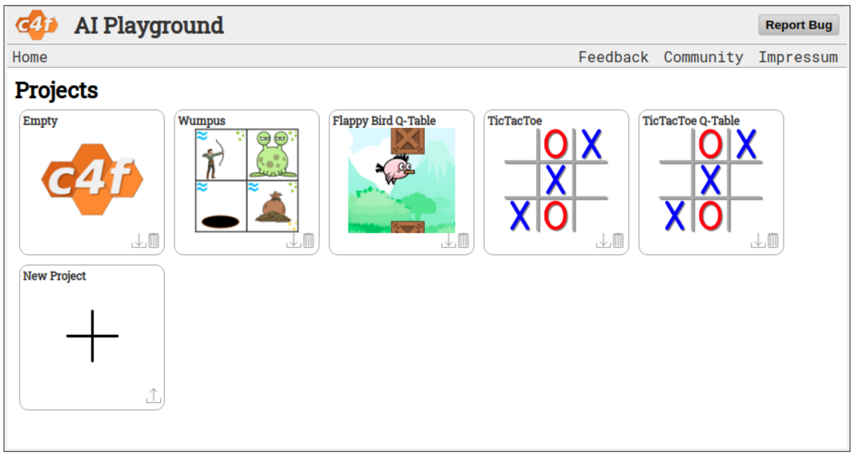
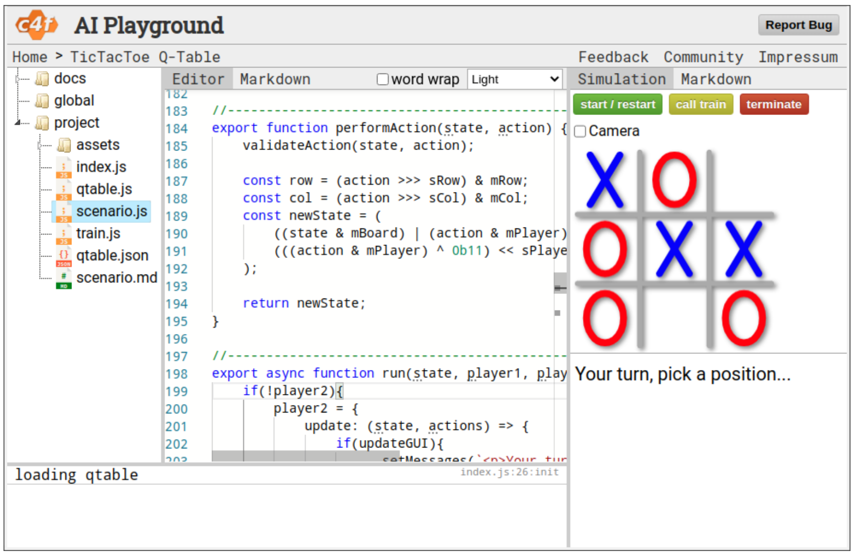


Figure 5.3.: Project index page

links to the corresponding files.
All components can be resized by dragging their separator. Figure 5.4 shows the project page with the opened *TicTacToe* scenario (see Section 5.3.2).



```
//----------------------------------------------
export function performAction(state, action) {
    validateAction(state, action);

    const row = (action >>> sRow) & mRow;
    const col = (action >>> sCol) & mCol;
    const newState = (
        ((state & mBoard) | (action & mPlayer)
        (((action & mPlayer) ^ 0b11) << sPlaye
    );

    return newState;
}

//----------------------------------------------
export async function run(state, player1, play
    if(!player2){
        player2 = {
            update: (state, actions) => {
                if(updateGUI){
```

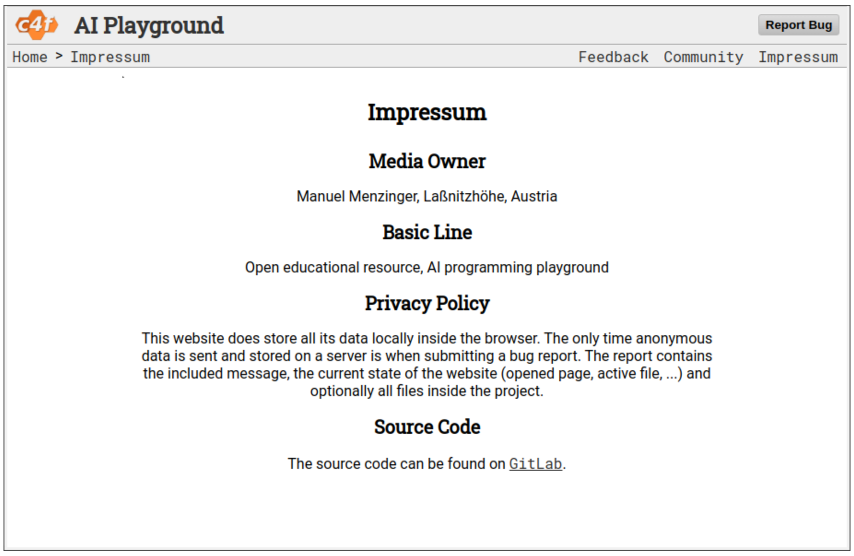
```
loading qtable                                          index.js:26:init
```

Your turn, pick a position...

Figure 5.4.: Project page

impressum

The page containing the impressum which includes contact information and the privacy policy, legally required for each website, as shown in Figure 5.5.



Impressum

Media Owner

Manuel Menzinger, Laßnitzhöhe, Austria

Basic Line

Open educational resource, AI programming playground

Privacy Policy

This website does store all its data locally inside the browser. The only time anonymous data is sent and stored on a server is when submitting a bug report. The report contains the included message, the current state of the website (opened page, active file, ...) and optionally all files inside the project.

Source Code

The source code can be found on GitLab.

Figure 5.5.: Impressum page

The design itself is focused on simplicity, leaning towards the *less is more* principle [29] and inspired by apps like *Google Drive*[10] and *GitHub*[11]. The main goal was to reduce visual clutter to a minimum, shifting the focus to the few present elements. Therefore, very few colors are used and the only images directly correlate to the scenarios themselves. Additional focus was laid on spatial proximity and familiarity. Thus, the *header* component was introduced as a fixed element on all pages, which includes all basic navigation elements. Furthermore, all icons for manipulating an element, like exporting or deleting a scenario, are close to the element itself, visually grouped together and separated from other elements by thin lines as can be seen in Figure 5.3.
All icons used for this work are either created by the author or from the

[10] `https://drive.google.com`
[11] `https://github.com`

Basic User Interface Elements icon set[12] by Stefan Taubert released under the *Creative Commons By 3.0* license.

State Management

As an application grows in size and complexity, robust state management is required. The approach used in this work is a centralized store system, based on the *MobX* project, an open source state management solution released under the *MIT* licence. In this system there can be multiple stores, each managing their own state using predefined functions. Furthermore, it is possible to subscribe to parts of the state by providing callback functions, which get executed whenever the relevant part of the state has changed. These stores are just objects in memory, which get lost whenever the window is reloaded. Therefore, some parts of the state are additionally stored in either the *localStorage*, a simple *key-value* store for persisting data inside the browser, the asynchronous local database called *indexedDB* or by modifying the Uniform Resource Locator (URL) in the address bar of the browser. However, to ensure data consistency, every modification of the state has to be done using the stores, while read access can also be done directly but is usually triggered by a callback from a state modification.

We use three distinct stores, all residing in and controlled by the *main thread*. As each thread has a separate scope with no sharing, whenever another thread needs access to the state, it has to communicate with the *main thread* using messages or read the data from one of the persistent storages.

app-store
: The *app-store* contains global information like the currently visible page and modal as well as network availability. Most of this data is dynamic, therefore only the opened page is persisted by modifying the *URL*. As a result, like in *multi-page* applications, a specific page can be linked to using the *URL*.

project-store
: The *project-store* contains information about the currently opened project like opened files as well as log and *file-tree* changes. While

[12] `https://www.iconfinder.com/iconsets/basic-user-interface-elements`

all user files are stored inside the *indexedDB*, the opened project is again stored inside the *URL*.

settings-store
: The *settings-store* is a simple *key-value* store containing various user settings like the color theme of the editor or the position of the space separators. It is meant to be a flexible store with no predefined fields, therefore whenever a new element needs to persist some user specific information, this store can be used. All data is persisted inside the *localStorage* which, compared to the *indexedDB*, has the advantage of being synchronous which allows fast and easy access.

Worker Management

It is the responsibility of the *main thread* to manage additional workers. In the context of a web browser, the *main thread* controls everything, which includes rendering and input handling. As a result, a blocking operation can freeze the entire application and make it completely unresponsive. For this reason some of the work is transferred to separate threads using *Web Workers* and *Service Workers*. While *Web Workers* are reliant on the *main thread* and exist only when the website is visited (active), *Service Workers* are independent and prevail in the background. Furthermore, all network calls can be intercepted by a Service Worker which enables powerful caching capabilities. In this work, a *Service Worker* is also used to provide user generated files as if they were requested over the network, although they are actually stored locally. This approach enables all available browser tools for debugging, as the files are treated like regular code.

The two workers used in this work are:

scenario-worker
: The *scenario-worker* is responsible for running user generated code in a separate context. Additionally, it provides an enhanced programming context by passing events like logging and mouse input to and from the *main thread*.

service-worker
: The *service-worker* provides access to user generated files and is used for offline caching.

5.2.2. Scenario Worker

The *scenario-worker* provides a separate thread and context for the user code to be executed. One of the main benefits of separating the user code from the rest of the application is to not interfere with the general behavior of the website. Even if the user creates an infinite loop by accident, the *main thread* continues to work normally and is able to stop the user code at will. The separation into a different context means the user code has in general no access to any other part of the application. Therefore the user can not easily make the website unusable by accident.
Furthermore, the *scenario-worker* handles sending and receiving messages from and to the *main thread* as well as modifies the globally available console object to intercepting logging commands. Using messages, these are then sent to the *main thread* to be displayed in the *console*.

The *scenario-worker* listens to three types of messages:

call-messages
Call-messages allow the *main thread* to call arbitrary functions inside the user's code which are used to start the scenario itself or to run some training function.

event-messages
Event-messages are responsible for providing access to mouse events which allow scenarios to be interactive. It makes it possible for the player to take over control of the game or play against his or her own created AI.

video-messages
Video-messages provide the possibility to receive video frame updates when a camera is active which can be used for object or face tracking.

Additionally, there are three types of messages the *scenario-worker* can send to the *main thread*:

log-messages
Log-messages are used to make log commands not only available in the browsers developer console, which in general is not visible by default, but to display them in the *console* component on the *project* page.

json-messages
: *Json-messages* allow the user to store any information inside a local file using the *JavaScript Object Notation* (*JSON*) format [30]. This can be used to persist any kind of data during program execution, like storing the values of a trained neural network for future use.

html-messages
: *Html-messages* are used as a simple way to output small junks of text or other data like tables, formatted using the *Hypertext Markup Language* (*HTML*) [31]. The *main thread* then displays the result below the graphical output, as can be seen with the text "Your turn, pick a position..." in Figure 5.4.

5.2.3. Service Worker

Service Workers are workers that behave differently than any other code, as the browser runs them in the background. They therefore provide features otherwise unaccessible like background synchronization and push notifications, which can be used to inform users about changes on the server like new messages for the user. For this work, the important new feature is the ability to intercept network requests and to provide a custom result.

In this application, an offline first caching approach is used. At the beginning of each network request, the cache is checked. If there is already a cached version of the request available it is immediately returned as a result. Additionally, to be able to receive updates, the network request is further send to the server and the result is written to the cache. If there was no cached version before, the new result is returned to the user. Therefore, the cache is always updated for further requests, while the user gets an immediate result whenever possible. This drastically improves the responsiveness of the application, as the user does not have to wait for the network response, which can take up to a few seconds, depending on the current connection.
To improve things even further, the first time the website is visited, it downloads all required data, like pages, scenarios and libraries, in the background and stores them inside the browser cache. Therefore, after a few seconds

the whole application is locally available and even continues to work if the network connection is lost.

There are many websites like *JSFiddle*[13] or *JSBin*[14] which enable users to write *JavaScript* code and execute it inside the browser. The main method these sites are using is to store the user created files directly on the server and then providing the files like normal using a personalized *URL*. On the upside users are able to share the code with others the internet, but on the downside it requires constant network access. Another approach is to store the code locally and use functions like *eval* or *function objects* to execute the code. This approach has the downside that not all language features are available especially newer ones like *modules*.
Therefore, in this work a *Service Worker* is used to intercept specific network requests for user generated code and to serve the local files as network files. To be distinguishable from normal requests, the *URL* for these files start with either *project/* or *global/*. As a result it is possible to write any code and use any feature the browser supports. Additionally, all build in browser tools like the debugger and performance profiler are fully functional and can be used like normal.

5.2.4. Adaptations for Classrooms

While the proposed framework can be used in many different contexts, its main goal is to provide a useful AI development environment for schools. Therefore, there is a clear focus on removing or reducing any barrier that can prevent adoption in class like privacy concerns or installation requirements. This section takes a look at some of the barriers outlined in the preliminary survey in Chapter 3 as well as some general barriers.

Availability and Usability

One of the main barriers for tools to be used in class is the system administration. Not every teacher has the privileges to install his or her own

[13] `https://jsfiddle.net`
[14] `https://jsbin.com`

programs and system administrators can, for various reasons like security concerns, reject or delay the installation of tools. More problems arise when students have to use the program at home, where different operating systems or lacking system permissions add to the number of roadblocks. Therefore, the framework was planned from ground up to be accessible without installation using only tools every modern system already provides: a web-browser. While this means there is an initial network connection required, this app is also designed to work completely offline once loaded. Even in the case of complete network failure the program can be used without limitations as long as it was opened once before.

While by the nature of the approach a basic knowledge in programming is required, the general *UI* was designed to be as lean as possible, providing just the most relevant functionality by default. This makes it easier for new users to navigate through and find the relevant parts in the application. Nevertheless the site provides powerful utilities inside the *editor*, as it is based on the popular editor *Monaco* [28], for advanced users who are willing to learn some shortcuts or dig into the depths of the *command palette*.

Security and Privacy

Another barrier can be the requirement to create an account to use a tool, due to privacy or security concerns. On the one hand student data can be quite sensitive, on the other hand most people reuse the same or similar passwords on multiple sites, reducing the security with each new account [32][33].

To prevent all these problems, there does not exist user registration in this application. All the data is stored locally inside the browser and only leaves the machine when manually exported into a file by the user. Furthermore, the website uses no tracking and does not store any data at all (with the exception of optional submitted bug reports) on the server.

Another concern when using user generated code is security and the possibility of writing malicious code. Due to the local nature of this approach users only have access to their own code which prevents attacks on others. While the code can be manually exported and imported, which makes it possible to run malicious code on different machines, this attack requires manual access to the targeted machine. At this point the attacker can already do everything he or she wants. As a point of reference, an attacker can already do the exact same thing (executing arbitrary *JavaScript* code) with the default developer tools available in every browser, which can even be executed on any website.

Adaptability

While there are ready-to-use scenarios already on the website, it can be beneficial to create new scenarios for specific purposes. This application allows easy creation of new or adapted scenarios as well as the possibility to share these scenarios as templates by exporting them and providing the resulting file to the students. All features can be accessed directly on the website, as can be seen due to the fact that all provided scenarios were programmed on the website itself.

5.3. Projects and Scenarios

In this section, the general structure of all projects and scenarios is presented as well as the three provided scenarios and the rationale behind them. This only are the scenarios which are currently available on the website by default. More scenarios will come in the future (see Section 8.2). Furthermore, each user can easily create their own scenarios directly inside the application and share them with their friends or students using the import and export functionality.

The following chapter will include some algorithms used for creating intelligent programs, like *reinforcement learning*, *Q-learning* and *minimax*. Knowledge about these algorithms is not necessary to understand this thesis, but if one wants to understand them more, a good reference is the book *Artificial Intelligence - A Modern Approach* [21], which was used by the author himself as a work of reference while creating these scenarios.

All images are, unless otherwise stated, either created by the author or released under the *Creative Commons Zero* license, making them available without restriction to the public domain. They where found on sites like `https://publicdomainvectors.org` and `https://opengameart.org`.

5.3.1. General Structure

A project is basically a collection of files. It is required to have a unique name and at least one file called *index.js*. This file is the main entry point and has to contain an function named *start*. Additionally, a function named *train* can be provided. While both functions can be manually triggered by the user using the corresponding buttons in the *simulator* component, the *start* function is called additionally whenever the project is opened. This ensures that the user can immediately see and interact with the scenario. While both functions can contain arbitrary code, *start* is meant to start a new run of the scenario while *train* could be used for a learning AI to play training games against itself.
In addition, in all default projects a file called *scenario.js* is present, implementing all the functionality of the scenario. Most scenarios also include a *scenario.md* file containing documentation on how to use its provided functions.

A scenario contains all the code required to simulate an environment. In the provided scenarios the environment is some kind of game, but it could be anything the user is willing to program. Additionally all provided environments are modelled around an agent, which uses callback functions for interaction, as can be seen in Figure 5.6. The most important callback is *update*, which has to be present in every agent. *Update* receives at least the current state or the agent's observations as an input and has to return the action the agent wants to take. Every time the agent has to make a decision inside the scenario, the *update* function is called and the returned action is used.
All other callback-functions are optional and correspond to specific events. The *init* and *finish* functions are called at the beginning and end of a scenario, the latter also provides information about the final state and some kind of scoring the agent reached. Finally, the *result* function is called after each

action took place. It provides information about the previous state, the new state, the chosen action and optionally additional information for calculating the reward. This function can be used for *reinforcement learning* to train an agent by updating its decision process given the success or failure of its action.

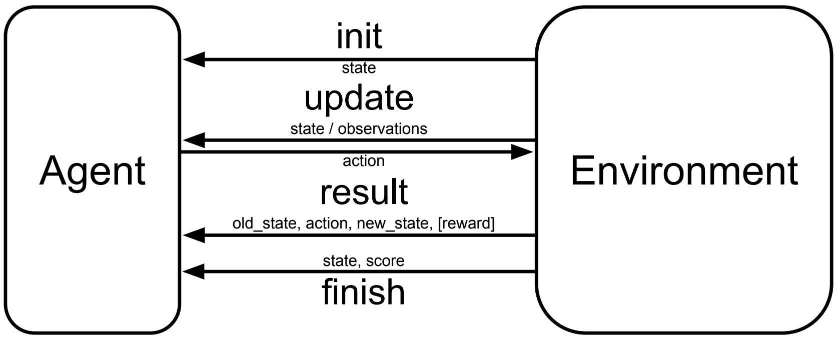


Figure 5.6.: In this figure, all four agent callbacks can be seen. These functions are executed under specific circumstances and provide some information using arguments. Only the *update* function has to be provided by the agent, the others can be omitted. *Update* is further the only callback that has an *action* as a return value.

The entire functionality of the scenario resides inside the file *scenario.js*. In the provided scenarios, this file includes at least one function called *run* which takes one or multiple agents as well as some settings and starts a new run of the game. While this file in general is not meant to be modified, a skilled user can change it to provide game variants or increase/reduce the difficulty of the scenario.

While this structure is currently used in all provided scenarios, it is not strictly required. A user can simply create a scenario with a completely different structure, as long as there is an *index.js* file containing a *start* function.

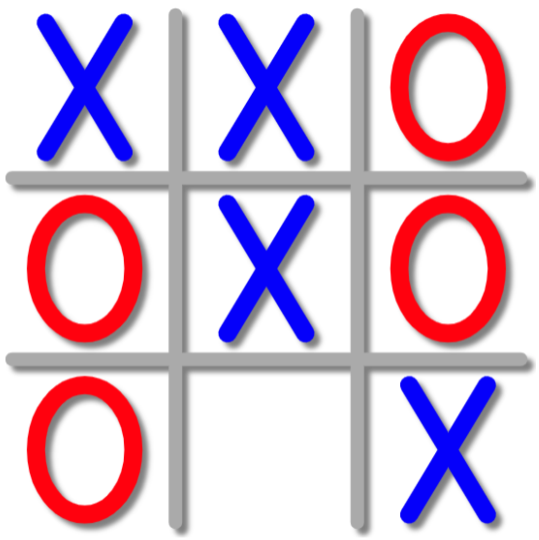


Figure 5.7.: Scenario TicTacToe

5.3.2. TicTacToe

TicTacToe, also known as *Noughts and Crosses*, is a two player game, played on a three by three board. The players take alternating turns and check one of the fields as theirs, one using the symbol X, the other the symbol O. The first player who has three of their symbols in a row (horizontally, vertically or diagonally) wins.
The game itself is quite simple and as long as no player makes a mistake neither can win. Additionally, the number of possible states is quite limited, and therefore every possible board state can be calculated and an unbeatable AI can be created.

The provided *scenario.js* file does not only provide the *run* function, but also intermediate functions like *getScore* and *performAction*. This has the advantage that the user has more control over the game which makes it easier, for instance, to calculate all possible states in advance. Furthermore, the *run*-function can be used with either one or two agents. If only one agent is present, the second agent is controlled by the user, who can then play the game against the computer.

Inside the default template there is already a simple agent present, which choses random actions to play against. Additionally, further examples are provided, one creating an unbeatable AI using a *minimax* algorithm and another using *Q-learning*.

5.3.3. Wumpus

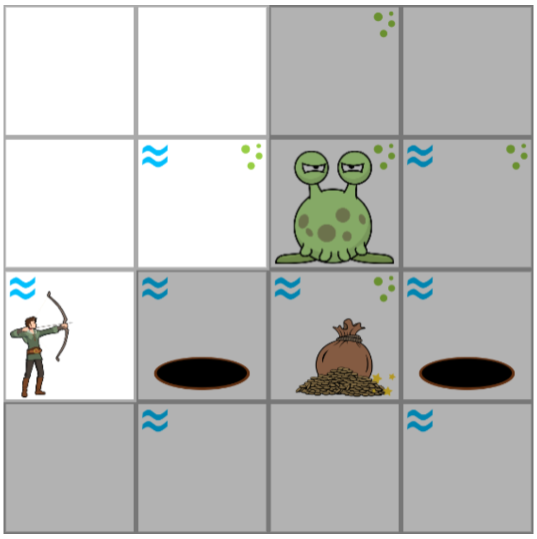


Figure 5.8.: Scenario Wumpus

The *Wumpus* scenario is a variation of the classical *Wumpus World*, which is a classical environment for agent-based AIs [21, Ch. 7]. It consists of a cave and an explorer who has to find a treasure without dying. The map consists of square tiles, with the default being four tiles wide and tall. Inside the cave there are two kinds of deadly obstacles: pits where the adventurer can fall to his death and the *Wumpus*, a big green monster that eats the explorer. Furthermore, there is a treasure which he has to reach to win. The agent has no information about the map and only has his perceptions to make a decision for his next move. There are three different perceptions: either a breeze, when he is adjacent to a pit, a stench, when he is adjacent to the *Wumpus* or some glitter when he has reached the treasure.
The map in this scenario is randomly generated, using a given seed. While it is ensured, that every map has a solution, there can be situations where the

given information is not sufficient to come to a clear conclusion. Therefore, even a perfect AI has to sometimes make decisions with uncertain outcome. Furthermore, the map can be modified, as the explorer has a bow and one single arrow which allows him to kill the *Wumpus*, signaled by a deep scream. This makes the previously blocked tile passable.

As in the *TicTacToe* scenario, the *scenario.js* file does not only provide the *run* function, but intermediate functions like *getPercepts* and *getTile* which can help in the decision making process. Furthermore, the *run*-function can be modified using settings which include the size of the cave, the used seed as well as the complexity. The complexity decides the type of movement that is available. When set to *simple*, the adventurer can instantly move to any unexplored tile, as long as it is adjacent to an already visited tile. If set to *advanced*, the adventurer can only move one tile at a time, which increases the difficulty as additional pathfinding is required.

The default template includes an agent taking random actions. There is an additional example available using a logic-based approach, highlighting the use of the programming language *Prolog*, which can be used using the *JavaScript* library *TauProlog* [34].

5.3.4. Flappy Bird

Flappy Bird is a single player game with the goal to navigate a bird through a series of obstacles. It is a 2D game where the bird constantly flies with the same speed from left to right while obstacles occur periodically with only a small gap for the bird to fly through. At any time the player only has two possible actions, either to accelerate upwards (jump) or to do nothing, which results in downwards acceleration due to gravity. Every time the bird successfully navigates through an obstacle, it gets awarded one point and a new obstacle, with a randomized gap, appears.

In this scenario, the obstacles are randomly generated with the possibility to use a fixed seed to be able to recreate specific rounds and compare different approaches. In contrast to the *Wumpus* and *TicTacToe* scenarios which are turn-based, *Flappy Bird* is a real-time game. This results in frequent updates

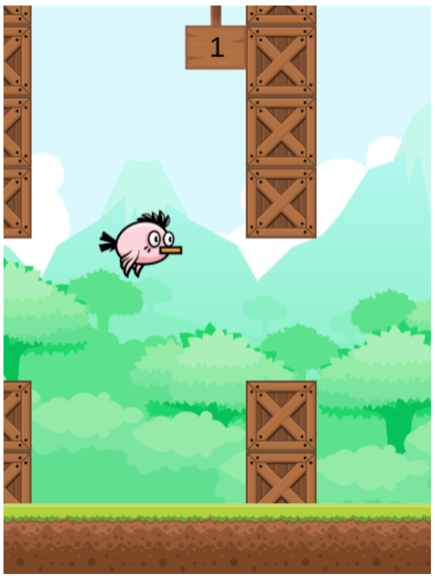


Figure 5.9.: Scenario Flappy Bird

with only small changes, which makes some approaches like a *Q-table* harder to implement.

While the *scenario.js*-file provides a *run*-function, only a few helper functions like *createAgent* and *createSettings* are present. Additionally there is a difference for the *update*-callback, as it provides observations instead of a complete state. These observations include the horizontal and vertical distance to the next gap, the current velocities, the score and the distance to the ground. For learning approaches requiring a state, the user himself has to find an appropriate mapping from observation to state.

While the default template includes the functionality for the user to play the game using the mouse, an additional template for *Q-learning* is provided. This template already has a working datastructure for the *Q-table* as well as a basic framework for a *Q-learning* algorithm. This can help novice programmers to focus on the learning algorithm and tweaking the basic parameters to understand their purpose and effect. Furthermore, there is a function to visualize the content of the *Q-table* to get a better understanding of how the inner workings change during each learning step, by watching it change in realtime.

6. Implementation

This chapter presents the technical details regarding the implementation of the application. The following sections explain in more detail the structure and the functions of the core elements. More details can be found on *GitHub*[1], where the entire source code is available.

6.1. Prerequisites

Web development has changed a lot in recent years, therefore this chapter provides an overview of the used tools and technologies as well as their interaction.

6.1.1. Development Environment

For development, the editor *Visual Studio Code* (or *VSCode*) [35] from *Microsoft* was used. It is a free editor, based on the *Monaco* [28] editor, which also is used as the online editor in the application. *VSCode* is a feature-rich *Integrated Development Environment* (*IDE*) and provides full support for all other used tools and libraries.

To provide access to all required tools and libraries, a local *Node.js* [36] environment was used. *Node.js* not only allows local execution of *JavaScript* code, but is also required for the *Node Package Manager* (*npm*) [37]. *Npm* provides access to over a million packages, including tools and libraries, and is considered to be the largest open-source package manager in the

1 https://github.com/c4f-wtf/ai

world[2]. For this work, all tools and libraries, with the exception of *VSCode*, were installed using *npm*. This has the additional advantage, that the entire project can be installed using the single command `npm install`, as long as *npm* is present. Furthermore, to test the website using a local server one simply has to use the command `npm start`.

The code itself is managed by *webpack*, a "static module bundler for modern *JavaScript* applications" [38]. *Webpack* is used to merge all project files together into a few *JavaScript* files called *bundles*, grouping connected parts together using *loaders*. *Loaders* allow *webpack* to process different kinds of files and to create valid *bundles* out of them. These *loaders* also enable the use of different programming languages by converting them into valid *JavaScript*, a process commonly known as *transpiling*. A *transpiler* is a compiler, that only converts between languages of similar abstraction levels. Therefore, users can program in any high-level language they want, as long as a suitable *loader* exists.

For the proposed framework, the programming language *TypeScript* [39] is used. *TypeScript* is based on *JavaScript* and enhances the language by adding static type information. In normal *JavaScript*, variables can be of any type and even change their type at will, which makes it harder in larger projects to ensure the correctness of the code. *TypeScript* removes this problem by requiring type annotations on all variables and objects as well as checking them for validity whenever it is *transpiled* to *JavaScript*. This process is done using the *loader Babel* [40], which is not only able to generate *JavaScript* out of *TypeScript*, but also to perform additional post-processing optimizations like *code minification* and *feature transpiling*. *Code minification* reduces the size of the resulting code to a minimum by removing all unnecessary whitespace and replacing names with shorter ones. This results in a significant lower file size, reducing the required network bandwidth to load the website. *Feature transpiling* enables the use of new features, even when they are not readily available in most browsers. It replaces parts of the code with different instructions that provide the same behavior, while being more readily available.

[2] `https://snyk.io/blog/npm-passes-the-1-millionth-package-milestone-what-can-we-learn`

A more comprehensive guide to this approach can be found in the *Front-end Developer Handbook 2019* by Cody Lindley [41].

6.1.2. Libraries

At the current state, this work uses 21 libraries during runtime, not counting dependencies and plugins for these libraries. All these packages can be found in the *Node Package Manager*[3] using the provided names.

deepcopy
: *Deepcopy* is a library for creating deep copies of objects. It works recursively and is capable of handling special cases like objects referring to themselves.

dexie
: *Dexie* provides easy access to the asynchronous local database named *indexedDB*, which is available in every modern browser.

file-saver
: *File-saver* allows the download of arbitrary binary data in the form of a file. In this work, this is used to create exported scenario files.

jquery
: *jQuery* is a well known *JavaScript* library, providing hundreds of utility functions. As it is heavily *DOM*-based, it is only used as a dependency for *jstree*, which in its current version does not work without it.

json-stringify-pretty-compact
: This small library is used to create well formatted strings out of objects using the *JavaScript Object Notation* (*JSON*). It is mainly used for the *console* component.

jstree
: *jsTree* is a library for creating interactive file-trees. It requires *jQuery* and is not able to run inside *web components*.

jszip
: *JSZip* provides functions to create and unpack zip-files.

[3] `https://www.npmjs.com`

lit-element
LitElement is the basic building block of the *Polymer Project*, which is an efficient framework for using *web components*.

lodash
Lodash is a library containing hundreds of utility functions like *throttle* or *flatten*. *Throttle* for instance is able to group frequent function calls together, which are then reduced to only a single function call in a given time frame. This is used while logging, as too many calls can slow down the system. Using *throttle*, the logging calls get queued up and then displayed collectively once every 100 milliseconds.

lz-string
lz-string provides access to text compression functions.

mobx
MobX is a lightweight but scalable state management library. It provides objects called *stores*, that provide controlled access to state information like the currently active page.

monaco-editor
Monaco Editor is a powerful code editor created by *Microsoft*. It is also the core of their popular local code editor *Visual Studio Code*, which was used to create this project.

prismjs
Prism is a simple code highlighting library, used in this work for formatting code examples inside *markdown* files.

pwa-helpers
pwa-helpers provide a few functions useful for programming web applications. It is mainly used for the functions *installRouter* and *installOfflineWatcher* to handle *URL* changes as well as changes in network connectivity.

resize-observer
resize-observer is a small library providing objects to register and handle changes in the size of *DOM* elements.

seedrandom
seedrandom.js is a small library providing a seeded random number generator. It is used to create deterministic randomness.

showdown
Showdown is a library to convert *markdown* into *HTML* code. It is used for all documentation files.

stacktrace-js
: *stacktrace.js* creates browser independent stacktraces from error objects.

tau-prolog
: *TauProlog* allows to use the programming language *Prolog* inside the browser.

tensorflowjs
: *TensorFlow.js* is a machine learning library, providing many machine learning models and algorithms inside the browser. It is the official *JavaScript* version of the popular *TensorFlow* library.

workbox-expiration, workbox-routing, workbox-strategies
: *Workbox* is library for *Service Workers*, making it easy to intercept routes and handle caching.

In addition to the runtime libraries, development libraries are used to compile all code into a optimized bundles. This can easily be done by using the command `npm run build:prod`.

babel
: *Babel* is a *JavaScript* compiler that can transform modern code into optimized, better supported code. This can allow the use of new language features, that are not not currently supported by many browsers. However, not all features can be transformed that way, especially more substantial ones like *Service Workers* and *offscreen rendering*. In this work *Babel* is mainly used to compile *TypeScript* into a minimized and optimized *JavaScript* code.

glob
: *Glob* is a small library allowing to search and list files inside a directory using patterns like stars.

typescript
: *TypeScript* is a programming language that enhances *JavaScript* by providing type safety.

webpack
: *WebPack* is a web bundling library that splits code into optimized chunks, which then can be loaded individually. It also includes additional capabilities like running a development server, while also updating changes in real time. Furthermore, it is able to run different compilers, like *Babel*, on different files.

6.1.3. Event Loop and Promises

Inside modern browsers, each tab has its own thread. As *JavaScript* is a single-threaded language, all operations are further performed in the same thread. Therefore, as long as any part of the website uses more computation time, all other parts, including rendering and input handling, have to wait for it to finish. To prevent constant freezing, *JavaScript* is built on an *event loop*. All operations are triggered by specific events, like when the website has finished loading, the user clicks on an element or a requested network file is available. Whenever an event occurs, it is pushed to the *event queue*. The *event loop* continuously performs predefined tasks like rendering the website and handling user input as well as processing the *event queue*. Each item on the queue gets processed completely, until the queue is empty and the loop continues.

There are two major ways of dealing with this-event base approach, one is to use *callbacks*, the other is to use *promises*. When using *callbacks*, one simply has to provide a function that will be called whenever the event is triggered. As an example, when requesting data over the network, the request is send and a callback function provided. Finally, when the file is available, the callback function is called. While this approach is easy to use, it becomes more cumbersome when there are multiple events which have to be fulfilled simultaneously (for instance multiple files have to be loaded before the scenario can start).

The more modern approach is to use *promises*. A *promise* is an object that acts like a placeholder for a value. At any point after creation it can get *resolved* or *rejected*. When a *promise* is *resolved*, the value is returned to a registered function. When a *promise* is *rejected*, an error is returned to a different registered function. The main difference to *callbacks* is, that it is possible to wait for multiple *promises* to resolve and then continue with all results being available.
To improve asynchronous programming even further, *async/await* was introduced. These new keywords allow a function to be asynchronous, by automatically changing the result into a *promise*. Furthermore, *await* can now be used inside *async*-functions to pause the execution until a *promise* is resolved while also directly returning the value without the need of any

callbacks.

A more comprehensive explanation of *promises* and *callbacks* can be found in chapter 7 of the book *JavaScript (ES2015+) Enlightenment* by Cody Lindley [42].

6.2. Core Structure

This section discusses how the central parts of the framework were implemented. It focuses on the most important functionalities and provides explanations on how they work as well as why they are used in this way. The implementation details for the provided scenarios can be found in the next chapter.

6.2.1. Main Components

Web Components

The core framework in this work is build on the *Polymer Project* [27]. Therefore all *UI* components are made of *litElements*, which is the basic building block of *Polymer*. *LitElements* handle the creation of *web components*, which encapsulate structure, design and functionality in one area. Furthermore, they use a technology called *Shadow DOM*, for strict encapsulation. As a result all structure, style and behavior definitions can only manipulate *DOM* elements inside the component itself, and no side effects can occur.

This approach is very different to traditional web development, where all elements can access everything and separation has to be done manually by carefully choosing names and ids that are unlikely to overlap. For instance if one would use the name *console* for an element that outputs some information, one would have to be certain, that no other element on the current page has the same name. If name duplication occurs, it becomes hard to differentiate between two elements when trying to access them. Furthermore, ids are required to be unique by the *HTML* standard. This can become especially problematic when multiple libraries are used which can create new elements by themselves.

Web components were only standardized in recent years, therefore most of the available libraries and frameworks use the traditional system, which by design is incompatible with the strict encapsulation of *litElements*. While it is possible to use *web components* in normal web development, it is impossible to use normal web development inside *web components*. Therefore, most of the available libraries for manipulating the *DOM* do not work inside of *web components*. This also affects the libraries *jsTree* and *monaco-editor* in this work. Although, by February 2020, *monaco-editor* added support for *web components*, it still has a few problems, preventing it from being used in this form for this application.
To still be able to use these libraries, *iframes* are used. *Iframes* in general enable different websites to be embedded inside an element. To solve the previously mentioned problem, the *file-tree* and *editor* components, both *web components*, contain *iframes* which load custom pages. These pages do not need to contain *web components* and therefore are able to use the libraries through traditional web pages. Using *JavaScript*, it is still possible to directly interact with the embedded libraries, as long as they reside on the same domain.

Messaging

To communicate between the *main thread* and its *workers*, messages are used. In *JavaScript*, messages can send a variety of data from one place to another. This data is automatically serialized and deserialized using the structured clone algorithm[4], making messages easy to use. However, not all data types can be cloned, like function or error objects. This is especially problematic as this data is frequently used in logging, to provide the user with more details regarding error messages. Therefore, a custom serialization function is used, which uses the *JavaScript Object Notation* (*JSON*). The advantage of this approach is that *JSON* is a human readable format, which further can be

[4] `https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Structured_clone_algorithm`

formatted using the *json-stringify-pretty-compact* library to create appealing logging entries.

In this work it is often required not only to send a message, but also to wait for a returning message. Therefore a utility function *messageWithResult* was created. This function makes use of *message channels*, which allow it to send messages from one point to another, without them being broadcasted to any other element. By using *promises*, this function makes it easy to send a message and to *await* the result.

6.2.2. State Management

States in this application are managed using the library *MobX* [43]. In *MobX*, states are encapsulated in objects, which in this project are called *stores*. Each *store* consists of *observables*, which are attributes containing some state information as well as *actions*, which are methods for modifying the stored information. In addition, there can be *computed values*, which are not used in this work.
Whenever any event modifies the *store* using an *action*, *side effects* can be triggered. *Side effects* can be anything from updating the *UI*, saving a file to the *indexedDB* to modifying the *URL*. The condition for triggering these *side effects* is a change in value of one or more specified *observables*. Furthermore, *side effects* themselves can trigger new *actions* which continues until all events are handled. See Figure 6.1.

This approach was chosen as it provides a simple but effective way of storing and modifying states, while also enabling relevant elements to get updated when necessary.

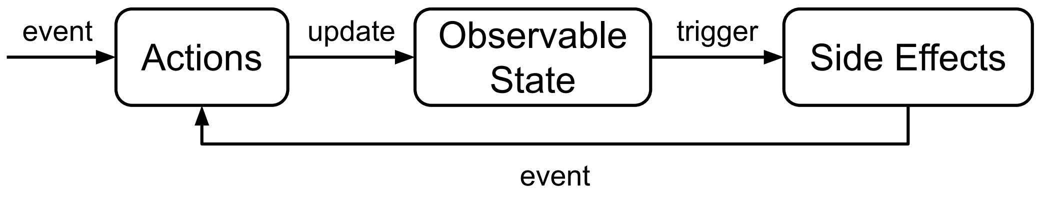


Figure 6.1.: Illustration of the basic MobX event loop. Adapted from `https://mobx.js.org`

App Store

The most central *store* is the *app-store* which controls the general website behavior. It uses the *observables page* and *params*, to store the currently visited *page* and additional information like the opened project. For controlling these *observables* the *action navigate* is used. Using the *pwa-helpers* library, *navigate* is triggered whenever the *URL* changes. It then parses all the provided information after the symbol # and stores them inside the *observables*. Furthermore, additional *actions* on different *stores* are triggered, depending on the *page*.
For example, if the *URL* ends with `#project=1`, the *app-store* stores the string `'project'` inside *page* and the pair `[project, 1]` inside *params*. Furthermore, it calls the *action openProject* from the *project-store* which then takes care of loading all required files. As a *side effect*, the *app* component reacts to the change and loads the *project* page which in itself reacts to the changes in the *project-store* and updates the *file-tree*, *editor*, *console* and *simulator* components. The whole process can be seen in Figure 6.2 and Figure 6.3.

This approach has the advantage, that the store directly correlates to the content of the *URL*, which can not only be easily modified using hyperlinks, but also be stored using bookmarks and shared with other users. It basically behaves the same way as a traditional *multi-page* website, which is what most users are used to.
The *app-store* also has *observables* for storing the network status, the currently opened modal as well as for when a *bug-report* is open or not. All these *observables* are not saved but always recalculated.

Project Store

The *project-store* handles all the information related to the currently active project. Its *observables* contain an object of the active project, an object of the active file as well as log messages and a timestamp for the last *file-tree* changes. *Actions* provided are *openProject*, *createProject*, *openFile* and similar functions. As a *side effect* it synchronizes all changes regarding the current file and project to the *indexedDB* and ensures data consistency, as it is the only place that is allowed to modify the database.

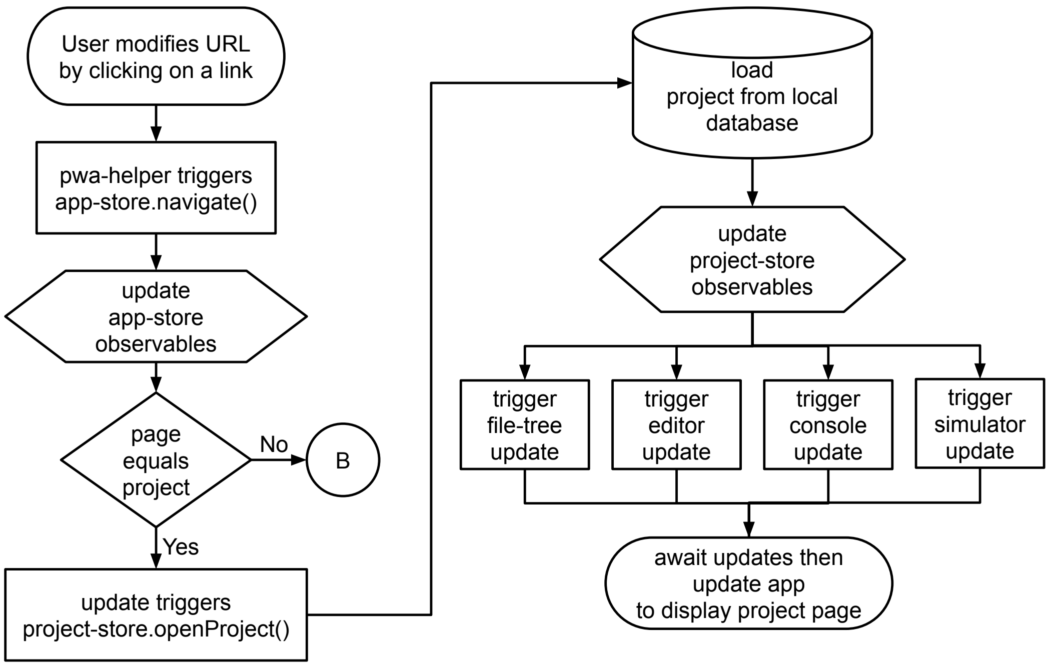

User modifies URL by clicking on a link
pwa-helper triggers app-store.navigate()
update app-store observables
page equals project
No
B
Yes
update triggers project-store.openProject()
load project from local database
update project-store observables
trigger file-tree update
trigger editor update
trigger console update
trigger simulator update
await updates then update app to display project page

Figure 6.2.: This navigation flowchart illustrates the process when a link is clicked. Generally, all triggers are only executed, when the corresponding *observable* actually changed. For better readability these conditions were omitted. The chart continues in Figure 6.3

Settings Store

The *settings-store* is the simplest of the *stores* and contains only one *observable* called *data*. This *data* object can store any key-value pair and can be modified using the *actions set* and *get*. As a *side effect* this data is synchronized to the *localStorage* of the browser, which allows synchronous storage of data. This *store* is used by many components to save arbitrary user data like the currently used theme of the editor or the position of a space separator. The synchronous nature makes it easy to use in comparison to the *indexedDB*, which requires asynchronous access using *promises*.

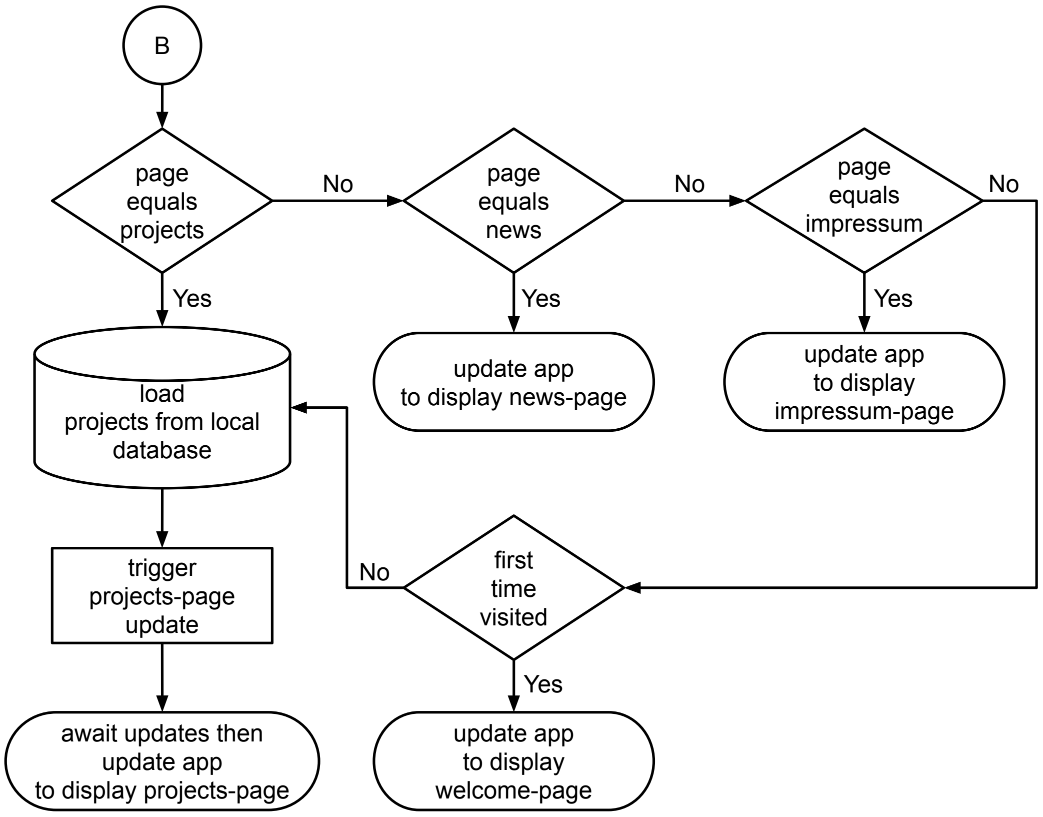

B
page equals projects
No
page equals news
No
page equals impressum
No
Yes
load projects from local database
Yes
update app to display news-page
Yes
update app to display impressum-page
trigger projects-page update
No
first time visited
Yes
await updates then update app to display projects-page
update app to display welcome-page

Figure 6.3.: Continuation of the navigation flowchart, showing additional possible outcomes. The chart further shows, that when no match fot the page can be found, either the projects-page or the welcome-page will be displayed.

6.2.3. Handling User Code

The main functionality of this project is the possibility to edit and run *JavaScript* code directly in the website, without using a server. This section explains how the code is locally stored and executed.

Storing User Code

All user data is stored inside the local database *indexedDB*. It consists of two tables, one for files and one for projects.

Files are objects containing a unique number as identifier (*id*), which is incremented automatically for each new file. Furthermore, it stores information about its name, content, the state of the editor, the timestamp of its latest change as well as the *id* of the project it belongs to. Inside the database, a combination of the project *id* and the name is used as the primary key, meaning every file inside a project needs a unique name. This is required as the import functionality of *JavaScript* is based on the name of files, therefore it would not be possible to distinguish between two different files with the same name. The content of each file can either be a string for text files, or an object containing binary data (called *Blob*) for images. Finally, the state is an object containing information about the cursor position and viewport scrolling, which enables users to continue working directly from where they left off.

Projects are objects containing also a numerical unique *id*, which is automatically incremented. Additionally, a name, the corresponding scenario as well as the *id* of the currently opened file are stored. While technically not strictly necessary, as the projects are always handled using their *id*, the name must be unique to prevent confusion.

Executing User Code

All files are stored inside the *indexedDB*, therefore they can be accessed by all *workers*. The connection between all components can be seen in Figure 5.1 in the Section 5.1.2.

When a scenario is started, at first an object containing the name and id from the current project is sent to the *service-worker*, which stores it as its active project. Then a new *scenario-worker* is generated. Finally a *call* message is sent to the *scenario-worker* instructing it to load the file `/project/index.js`. *Index.js* is a locally stored file, as indicated by the path starting with `project`, therefore the request is intercepted by the *service-worker*. It uses its active project *id* and the name of the file to load it from the *indexedDB* and then returns it as a *HTTPResult* to the *scenario-worker*. Finally the *scenario-worker* calls the function *start* which executes the user generated code.

This approach has the additional advantage, that the users themselves can import their files by just preceding the filename with `project/`. As a result, the user generated code is executed like any other code, therefore it has full access to all features available in the browser.

6.3. Projects and Scenarios

6.3.1. General Structure

All the code a user can interact with is grouped into *projects*. To be able to handle many different use cases, the general structure of each *project* is quite simple. The only requirement is that it has to provide a file named *index.js*, containing an *asynchronous* function called *start*. The function has to be marked as *exported*, as the *scenario-worker* uses a dynamic *JavaScript import* to access the function. While it is not strictly required to be *asynchronous*, it enables the user to use the keywords *async* and *await*, which make dealing with *promises* very intuitive. Many scenarios will use *promise*-based functions, for instance to react to user input or to create local files, which would require callback functions otherwise. The user can freely create and delete files inside each *project*, with the exception of the file *index.js* which can not deleted.

A *project* can provide a *scenario*, which simply means it includes code that provides all the functionalities the scenario requires. As an example, a *project* using the *TicTacToe* scenario simply contains files that provide functions to display as well as play a game of *TicTacToe*. All provided scenarios contain a single file with the name *scenario.js*, that includes the required code. This also means, users can modify scenarios inside their projects in whatever way they want.
To create custom scenarios, the user can either create an empty *project* and start from scratch or use an existing project and modify the scenario code. The *project* can then easily shared by using the export / import functionality on the page.

As the main entry point is just the single function *start*, the structure of the *project* is very flexible. Furthermore, a library called *utils.js* is available, providing general functions like *storeJson* for saving data, *getCanvas* for graphical output and *onMouseDown* for user interaction. Generally it is possible to use any library inside *projects* as long as they are based on *ECMAScript 6 modules*, using valid *export* statements. They either can be imported from any *URL*, or they can be uploaded to the *project* as a file. Additionally, the work currently provides two libraries *tensorflow.js* and *prolog.js* containing module-based versions of the *TensorFlow.js* and *TauProlog* libraries.
While it is possible to load some libraries using other formats, for instance by using the function *includeUrl* provided in the *utils.js* library, not all formats are compatible. Some libraries automatically try to inject their code inside the local *DOM*, which is not available inside the *scenario-worker*. The only solution for such libraries is to manually rewrite their code to use a different export format.

6.3.2. Provided Scenarios

Scenarios that are included in the source code of this project, and therefore available to all users, are called *provided scenarios*. These scenarios act as templates for creating new projects, which already include files providing the full functionality of the scenario.

Inspired by the *OpenAi Gym* [44], all *provided scenarios* use a uniform, agent-based structure. Therefore, they export an *asynchronous* function called *run*, which takes one or multiple *agents* as well as some scenario specific settings and then runs the entire scenario. *Agents* are objects, containing at least the required function *update* which resembles the decision making of the *agent*. It is called whenever the *agent* has a decision to make and provides either the current *state* or the current *observations* as a parameter. The function then has to return an *action* the *agent* is going to take. Optionally an *agent* can provide additional callback functions as described in Section 5.3.1.

Figure 6.4 illustrates how the *TicTacToe* scenario can be used for creating a simple agent taking random actions. In line 1 everything from the file

scenario.js is included, providing access the the whole functionality using the symbol *$*. Next, a settings-object is created defining the computer to be the starting player. The *start* function, is called when the scenario begins and only does three things. First it creates the initial board state using the previously defined settings. Then it creates an agent-object that contains the functions *update* and *finish*. Finally the game is started by using the *run* function and providing the initial state as well as the agent.
The *update* function simply generates a list of available actions using the provided function *getActions* and the current state. It then choses and returns a randomly selected action. In the end the *finish* function is called which logs the final result to the console.

```javascript
import * as $ from 'project/scenario.js';

const SETTINGS = {
    startingPlayer: $.EPlayer.Computer,
};

export async function start() {
    const state = $.createState(SETTINGS);
    const agent = { update, finish }
    await $.run(state, agent);
}

async function update(state) {
    // take a random action
    const actions = $.getActions(state);
    const action = Math.round(Math.random() * (actions.length - 1));
    return actions[action];
}

async function finish(state, score) {
    console.log('score: ', score);
}
```

Figure 6.4.: Code example of a simple TicTacToe project.

To be able to easily create new *provided scenarios*, they are individually bundled using *webpack* and reside in a separate folder inside the *src* directory. Each scenario has its own folder containing general files like the *scenario.js* and images, as well as subfolders for templates and examples. During compilation a custom script parses the folder structure and creates a list of available scenarios, templates and examples which then is used inside the *scenario-index* page to list and create new scenarios. As a result, a new *provided scenarios* can be created by simply providing its files inside a folder and rebuilding the framework.

6.4. Technical Limitations

This work makes use of very modern technology, which is not yet available in all browsers. However, all used concepts are either in the process of being standardized by the *World Wide Web Consortium* (*W3C*), or already standardized. Furthermore, they are either on the bug- or todo-list of most major browsers, resulting in a high likelihood of being widely adopted in the next few years.

6.4.1. Module Workers

While classic web workers are readily available, they don't work with modules. To be able to allow users to use modern *ECMAScript 6 modules*, a new type of workers, called *module workers*, have to be used. Unfortunately, the support for these workers is quite low. At the time of writing, only recent versions of *Chromium*-based browsers (*Google Chrome*, *Opera* and as of 2020 also *Microsoft Edge*) have working implementations available.

6.4.2. Service Workers

In this work, a *Service Worker* is used to provide access to user-generated code. While they are available on all modern browsers[5], some browsers like *Firefox* currently disable them in contexts like private mode. Therefore, even if a browser does support the technology, depending on the context, settings or extensions can disable the required functionality.

[5] `https://caniuse.com/#feat=serviceworkers`

6.4.3. Offscreen Canvas

HTML5 Canvases are used for rendering, but they have to be created within the *main thread*. The scenarios themselves, however, reside on the *scenario-worker*. To provide access to the *canvas* this work relies on a technology called *Offscreen Canvas*, which is not available in all browsers at the moment[6].

[6] `https://caniuse.com/offscreencanvas`

7. Evaluation

To be able to evaluate the proposed framework, formal feedback in the form of a questionnaire, as well as informal feedback during talks with some users was collected. The goal of this evaluation was to determine which parts of the framework were well received and which led to problems. These information will then be used to steer the future direction of the project.

7.1. Feedback Questionnaire

To ensure an easy and consistent way for providing feedback, a feedback questionnaire was created and published on the project website itself. Every user of the site can provide feedback by clicking on a link inside the header portion of the page and submitting the feedback.

The entire questionnaire can be found in Appendix B.

7.1.1. Design and Implementation

The feedback questionnaire was designed as a quantitative online survey, directly targeting users of the project. To increase user participation, the general feedback was designed to be short and concise, while still allowing users to elaborate on specific topics when needed. Therefore, the first part of the questionnaire consists of three single and multiple choice questions, in which the users can state how much they agree or disagree with certain statements as well as a short question about how many hours and for what purpose the framework has been used. The 11 statements are centered around the usability and reliability of the website itself as well as the practicality of provided scenarios, documentation and examples. Questions in the latter part can be answered by text and cover topics like what new scenarios the users would like to see or what features they felt were missing.

For ease of creation and distribution, the feedback questionnaire was created using *Google Forms*[1]. The resulting link was incorporated into the header of the website, easy to find for all users. Additionally, a posting on message boards of multiple *EDLRIS AI* [4] courses conducted over the last two years was created, to thank people for their participation and to make them aware of the feedback questionnaire, which would help to improve the project. Information about the project and the questionnaire was also sent to individuals in the field of computer science education and posted on the *RoboCupJunior Austria* channel on *Facebook*[2]. Furthermore, a short paper [45] about this work was written and presented at the *2nd International Workshop on Education in Artificial Intelligence K-12*[3] which was held online in conjunction with the *21st International Conference on Artificial intelligence in Education*[4]. Finally, to help users understand how the project can be used, a exemplary 40 minute lesson about teaching *reinforcement learning* with the *Flappy Bird* scenario (Section 5.3.4) was recorded and published on *Youtube*[5]. The language of the video is German, to make it easier for local teachers to directly use it in class.

The response rate was rather low, because of the required expertise to use the proposed framework for teaching and because of the current *Covid19* pandemic. The latter resulted in a lot of additional work for virtual teaching, as informal talks with individuals from the targeted audience revealed. Nevertheless, some conclusions can still be drawn despite the small sample size.

[1] `https://www.google.com/forms/about`
[2] `https://www.facebook.com/RoboCupJuniorAustria`
[3] `http://eduai.ist.tugraz.at`
[4] `https://aied2020.nees.com.br`
[5] `https://www.youtube.com/watch?v=H939JwHIy0s`

7.1.2. Evaluation Results

Over the course of five months, from the first release of the project in June 2020 until the time of writing of this thesis, seven people provided their feedback. In the reminder of this chapter we present the evaluation of the individual questions and discuss the results in detail.

For readability, the descriptions of the individual questions have been shortened inside the figures. The complete questionnaire can be found in Appendix B.

Q1: How many hours have you been using the framework for?

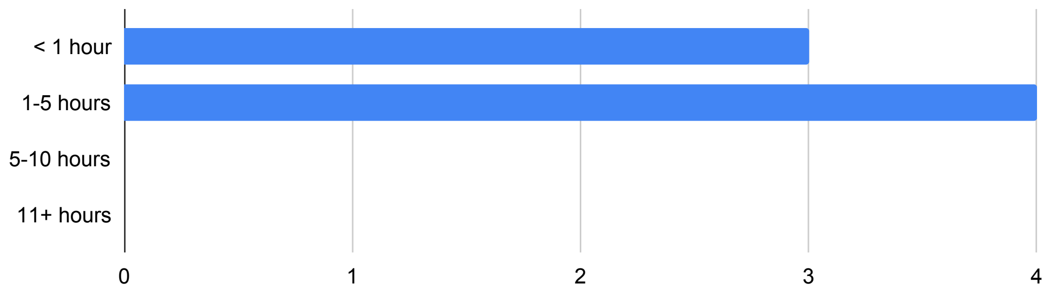


Figure 7.1.: Result of Question 1: How many hours have you been using the framework for?

Four of the participants used the website between 1 to 5 hours, while three stated they used it for less than an hour, as seen in Figure 7.1. Therefore, while most participants took at least a fair look at the project, the time periods were too short to use this project for anything substantial. As a result, the significance on the provided feedback is quite limited.

Q2: For what purpose have you been using the framework?

Figure 7.2 shows, that while most of the participants reported they used the framework for learning AI themselves, two also stated that they have used this work for teaching AI in class. One mentioned the reason was to get to

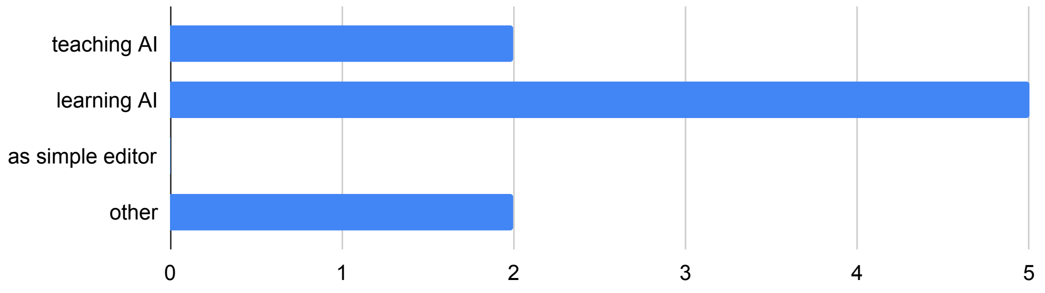


Figure 7.2.: Result of Question 2: For what purpose have you been using the framework?

know the website to use it for subsequently teaching of AI, while another one stated to just want to support the development of the framework.

Q3: How much do you agree with the following statements?

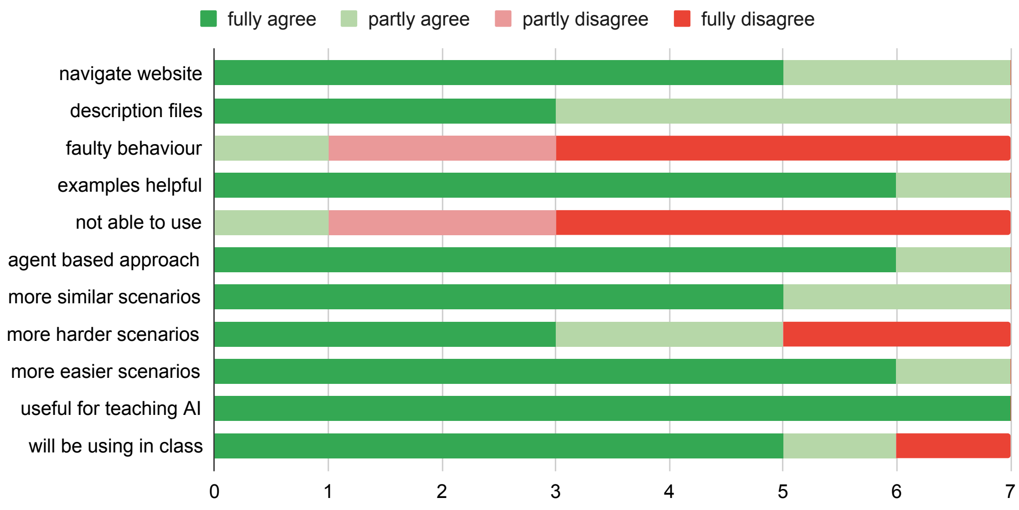


Figure 7.3.: Result of Question 3: How much do you agree with the following statements?

The consensus of all participants was, that in general the website was easy to use and included useful scenario descriptions. They liked the agent-based approach very much and they would be looking forward to see more scenarios of similar or lower complexity. Futhermore, Figure 7.3 shows that all participants found the framework useful for teaching AI in class and five out of seven stated a high likelihood of using it in class in the future.

While one of the participants partly agreed and two only partly disagreed, with the statement of encountering bugs or faulty behavior, no bug reports were filed. Therefore, and due to the generally positive feedback, it can be assumed that the encountered problems were minor and did not disrupt the usage of the framework.

Q4: What new scenarios would you like to see?

Half of the participants replied with new scenario ideas, mentioning classical games like *Go*, *Snake*, *Tetris*, *Super Mario*, *Battleship* and *2048*. One also suggested to provide simple games like *cat - mouse - cheese*, while another mentioned games with more complexity like *Polytopia* and popular board games.

Q5: What general features are you missing?

The only mentions of missing features were "maybe some more statistics" and that there were "none to speak of" as well as "n.a.".

Q6: Final thoughts and additional remarks

Four of the participants added additional remarks. One stating "it is a super tool" while the other also mentioned that "it will be interesting to see how pupils and teachers adopt/apply the framework". The third one mentioned he had been reliant on the additional video, which allowed him to work in parallel, due to the appropriate pace of the video. The last one also liked the "good tutorial video on YouTube" as well as the "very clear UI".

7.1.3. Conclusion

While no conclusion of scientific significance can be drawn from such a small sample size, the general feedback was still quite positive. All participants found it to be a useful tool for AI education and most stated that they would like to use it in class. Furthermore, the agent-based approach used for the scenarios was well received and further scenarios are looked forward to.

From a technical point of view, the website worked decently well, as half of the participants fully disagreed with the statement of experiencing bugs, while using the framework. The others partly disagreed and partly agreed, which probably means they encountered some problems, but nothing that kept them from using the website.

While the documentation was generally received in a positive way, there is still room for improvement, as two thirds only partly agreed with the usefulness of the provided scenario descriptions.

Usage in Workshops and Courses

Since the release of the website in June 2020, it was further used in two online school workshops, as well as a lecture for students in the teacher training program of the *Graz University of Technology* (*TU Graz*).
The one hour long workshops were held at the *WKÖ CodingDay 2020*[6] in October and used the framework to familiarize two classes with the concept of *Q-learning*, using the scenario *Flappy Bird*. While the time was short, the lessons and the framework were well received by the participating teachers and students.

During the lecture *"Einführung in das Studium für das Lehramt Informatik"*, a first year lecture at the *TU Graz*, the framework was used to familiarize the students with the possibilities of online tools. The lecture was held online and a 30 minute section was used for presenting the website. Multiple questions from the students suggest that the subject was interesting and engaging. Furthermore, at least some students stated during evaluation that the reinforcement learning part of the lecture was to technical, which

[6] `https://www.wko.at/site/codingday/start.html`

suggests that the topics were not taught to them in school. This further emphasizes the need for better AI education in class.

Personal Usage in Class

In addition to usage in workshops, the framework is used by the author himself in his three year computer science curriculum in school. It is used in class to educate the students in the field of reinforcement learning and is further used in the upcoming school leaving examination called *Matura*. Due to the ability to continue working even when offline, the framework was a reliable tool in an environment that lacks reliable network infrastructure. The website was also very well received by the students as an engaging and fun project. One student even stated that it will be the base of his pre-scientific work in the next year, which is also part of the *Matura*.

During all these practical uses, the framework stayed reliable and no major bugs occurred. This shows that the chosen approach and the design decisions made, were appropriate and the implementation in general a success.

8. Conclusion

8.1. Discussion

Artificial Intelligence is the one of the most important technologies at the moment. Therefore, educating young people on the underlying principles, the used methods as well as the risks and potential, not only using theoretical but also practical approaches, becomes more and more important. The AI education framework presented in this thesis aims in this direction and tries to fill the gap of easy-to-use, low-barrier tools to help educators in their journey.

The work was developed using an agile approach, which enabled frequent adaptations and early working prototypes. Therefore, it was possible to try different libraries and approaches without major time investment, which was very helpful for finding the chosen path.

While there is always room for improvement, the feedback provided during the evaluation, but also in informal discussions, reveals that this project in general is well received as a useful addition to the other available resources for teaching AI. In Figure 7.3 it can be seen that all of the participants fully agreed that the framework is a useful tool for teaching AI and most of them are likely to use it in class in the future. This, and the fact that the website was used without major issues on multiple occasions, supports the design decisions made before and during development.

In conclusion, the presented work succeeded in its goal to provide a useful, low-barrier tool for teaching more complex AI methodologies in school. It still can be further improved and will be worked on more in the future, as briefly discussed in the next section.

8.2. Future Work

The web application presented in this thesis already provides a useful and well working AI programming environment, as it can be concluded from the formal and informal feedback. Nevertheless, there is still quite some room for improvement. One major problem for adoption, and probably the reason for the low response rate in the evaluation, was the complexity of the subject, which requires more introductory and guiding resources. This can be provided in the form of further video tutorials and more detailed written explanations. Moreover, it is also planned to use the framework directly for teacher education workshops in the upcoming successor project to *EDLRIS* entitled *"Education and Awareness for Intelligent Systems"* (*ENARIS*).

While well received, the *UI* and usability of the website can still be improved. This includes small functionalities like renaming files and projects, which were omitted due to time constraints. Furthermore, all users mentioned that they wanted more scenarios, especially some with lower complexity. As this was anticipated from the beginning, it is easy for others to contribute new scenarios, which then can easily be directly integrated into the framework.

Finally, a personal note from the author about the ongoing work for this project: As a teacher myself, I already use this website in class and it helped me a lot to motivate students to take a deeper look into the complex field of AI. Therefore, I will continue working on this project, not only to keep it up to date, but to improve it and make it more usable, to help my colleagues and fellow teachers bringing AI to their students as well.

List of Figures

Bibliography

[1] Liu Shuguang, Li Zheng, and Ba Lin. "Impact of Artificial Intelligence 2.0 on Teaching and Learning." In: *Proceedings of the 2020 9th International Conference on Educational and Information Technology*. ICEIT 2020: 2020 9th International Conference on Educational and Information Technology. Oxford United Kingdom: ACM, Feb. 11, 2020, pp. 128–133. ISBN: 978-1-4503-7508-5. DOI: 10.1145/3383923.3383928 (cit. on p. 1).

[2] Stefania Druga et al. "Inclusive AI Literacy for Kids around the World." In: *Proceedings of FabLearn 2019*. FL2019: FabLearn 2019. New York NY USA: ACM, Mar. 9, 2019, pp. 104–111. ISBN: 978-1-4503-6244-3. DOI: 10.1145/3311890.3311904 (cit. on p. 1).

[3] Aggeliki Androutsopoulou et al. "Transforming the Communication between Citizens and Government through AI-Guided Chatbots." In: *Government Information Quarterly* 36.2 (Apr. 1, 2019), pp. 358–367. ISSN: 0740-624X. DOI: 10.1016/j.giq.2018.10.001 (cit. on p. 1).

[4] Martin Kandlhofer and Gerald Steinbauer. "A Driving License for Intelligent Systems." In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 7954–7955. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16110 (cit. on pp. 1, 17, 18, 72).

[5] *A Free Online Course - Elements of AI*. URL: https://course.elementsofai.com/ (visited on 06/02/2020) (cit. on p. 1).

[6] David Touretzky et al. “Special Session: AI for K-12 Guidelines Initiative.” In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE ’19. New York, NY, USA: Association for Computing Machinery, Feb. 22, 2019, pp. 492–493. ISBN: 978-1-4503-5890-3. DOI: 10.1145/3287324.3287525 (cit. on p. 1).

[7] Greg Brockman et al. *OpenAI Gym*. June 5, 2016. arXiv: 1606.01540 [cs]. URL: http://arxiv.org/abs/1606.01540 (visited on 12/02/2020) (cit. on pp. 2, 3, 9, 25).

[8] *TensorFlow*. TensorFlow. URL: https://www.tensorflow.org/ (visited on 08/17/2020) (cit. on pp. 3, 24).

[9] *AI Experiments — Experiments with Google*. URL: https://experiments.withgoogle.com/collection/ai (visited on 02/24/2020) (cit. on p. 3).

[10] Tom Crick et al. “The Impact of COVID-19 and 'Emergency Remote Teaching' on the UK Computer Science Education Community.” In: *United Kingdom & Ireland Computing Education Research Conference.* New York, NY, USA: Association for Computing Machinery, Sept. 3, 2020, pp. 31–37. ISBN: 978-1-4503-8849-8. URL: https://doi.org/10.1145/3416465.3416472 (visited on 12/04/2020) (cit. on p. 7).

[11] J.-L. Gaudiot and H. Kasahara. “Computer Education in the Age of COVID-19.” In: *Computer* 53.10 (Oct. 2020), pp. 114–118. ISSN: 1558-0814. DOI: 10.1109/MC.2020.3011277 (cit. on p. 7).

[12] Fulvio Castellacci, Davide Consoli, and Artur Santoalha. “Technological Diversification in European Regions: The Role of E-Skills.” In: (), p. 35 (cit. on p. 7).

[13] Elena Fleaca and Radu D. Stanciu. “Digital-Age Learning and Business Engineering Education – a Pilot Study on Students’ E-Skills.” In: *Procedia Manufacturing*. 12th International Conference Interdisciplinarity in Engineering, INTER-ENG 2018, 4–5 October 2018, Tirgu Mures, Romania 32 (Jan. 1, 2019), pp. 1051–1057. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2019.02.320 (cit. on p. 7).

[14] Katarína Gašová, Tomáš Mišík, and Zuzana Štofková. "EMPLOYERS DEMANDS ON E-SKILLS OF UNIVERSITY STUDENTS IN CONDITIONS OF DIGITAL ECONOMY." In: *CBU International Conference Proceedings* 6 (Sept. 24, 2018), pp. 146–151. ISSN: 1805-9961. DOI: `10.12955/cbup.v6.1147` (cit. on p. 7).

[15] Duri Long and Brian Magerko. "What Is AI Literacy? Competencies and Design Considerations." In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20. New York, NY, USA: Association for Computing Machinery, Apr. 21, 2020, pp. 1–16. ISBN: 978-1-4503-6708-0. DOI: `10.1145/3313831.3376727` (cit. on p. 7).

[16] David Weintrop, David Bau, and Uri Wilensky. "The Cloud Is the Limit: A Case Study of Programming on the Web, with the Web." In: *International Journal of Child-Computer Interaction* 20 (June 1, 2019), pp. 1–8. ISSN: 2212-8689. DOI: `10.1016/j.ijcci.2019.01.001` (cit. on p. 7).

[17] Caitlin Kelleher and Randy Pausch. "Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers." In: *ACM Computing Surveys* 37.2 (June 1, 2005), pp. 83–137. ISSN: 0360-0300. DOI: `10.1145/1089733.1089734` (cit. on p. 8).

[18] Stefan Friese and Kristian Rother. "Teaching Artificial Intelligence Using a Web-Based Game Server." In: *Proceedings of the 13th Koli Calling International Conference on Computing Education Research - Koli Calling '13*. The 13th Koli Calling International Conference. Koli, Finland: ACM Press, 2013, pp. 193–194. ISBN: 978-1-4503-2482-3. DOI: `10.1145/2526968.2526992` (cit. on p. 8).

[19] Sébastien Combéfis, Gytautas Beresnevičius, and Valentina Dagienė. "Learning Programming through Games and Contests: Overview, Characterisation and Discussion." In: *Olympiads in Informatics* 10.1 (July 10, 2016), pp. 39–60. ISSN: 18227732, 23358955. DOI: `10.15388/ioi.2016.03` (cit. on p. 8).

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning An Introduction*. 2nd ed. MIT Press, 2018 (cit. on p. 9).

[21] Stuard Russell and Peter Norvik. *Artificial Intelligence - A Modern Approach*. 3rd. Pearson, 2010 (cit. on pp. 9, 46, 50).

[22] Ekaba Bisong. "Google Colaboratory." In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Ed. by Ekaba Bisong. Berkeley, CA: Apress, 2019, pp. 59–64. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_7 (cit. on p. 13).

[23] B. M. Randles et al. "Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study." In: *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL). June 2017, pp. 1–2. DOI: 10.1109/JCDL.2017.7991618 (cit. on p. 13).

[24] Dan Toomey. *Jupyter for Data Science: Exploratory Analysis, Statistical Modeling, Machine Learning, and Data Visualization with Jupyter*. Packt Publishing Ltd, Oct. 20, 2017. 236 pp. ISBN: 978-1-78588-329-3. Google Books: sRhKDwAAQBAJ (cit. on p. 13).

[25] *Web Technology for Developers*. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web (visited on 11/30/2020) (cit. on p. 27).

[26] *History of the Web*. World Wide Web Foundation. URL: https://webfoundation.org/about/vision/history-of-the-web/ (visited on 11/15/2020) (cit. on p. 31).

[27] *Polymer Project*. URL: https://www.polymer-project.org/ (visited on 05/11/2020) (cit. on pp. 35, 59).

[28] *Monaco Editor*. URL: https://microsoft.github.io/monaco-editor/ (visited on 11/11/2020) (cit. on pp. 36, 45, 53).

[29] *Less Is Still More: The Importance Of The Minimalist Approach To Web Design*. Usability Geek. Apr. 13, 2016. URL: https://usabilitygeek.com/less-is-more-importance-minimalist-web-design/ (visited on 11/12/2020) (cit. on p. 39).

[30] *JSON*. URL: https://www.json.org/json-en.html (visited on 11/12/2020) (cit. on p. 43).

[31] *HTML Standard*. URL: https://html.spec.whatwg.org/multipage/ (visited on 11/12/2020) (cit. on p. 43).

[32] Elizabeth Stobert and Robert Biddle. "The Password Life Cycle: User Behaviour in Managing Passwords." In: *10th Symposium On Usable Privacy and Security* (SOUPS 2014), pp. 243–255 (cit. on p. 45).

[33] Dinei Florencio and Cormac Herley. "A Large-Scale Study of Web Password Habits." In: *Proceedings of the 16th International Conference on World Wide Web - WWW '07*. The 16th International Conference. Banff, Alberta, Canada: ACM Press, 2007, pp. 657–666 (cit. on p. 45).

[34] *Tau Prolog: A Prolog Interpreter in JavaScript*. URL: `http://tau-prolog.org/` (visited on 11/12/2020) (cit. on p. 51).

[35] *Visual Studio Code - Code Editing. Redefined*. URL: `https://code.visualstudio.com/` (visited on 11/16/2020) (cit. on p. 53).

[36] Node.js. *Node.Js*. Node.js. URL: `https://nodejs.org/en/` (visited on 11/16/2020) (cit. on p. 53).

[37] *Npm — Build Amazing Things*. URL: `https://www.npmjs.com/` (visited on 11/16/2020) (cit. on p. 53).

[38] *Webpack*. webpack. URL: `https://webpack.js.org/` (visited on 11/16/2020) (cit. on p. 54).

[39] *Typed JavaScript at Any Scale*. URL: `https://www.typescriptlang.org/` (visited on 11/16/2020) (cit. on p. 54).

[40] *Babel · The Compiler for next Generation JavaScript*. URL: `https://babeljs.io/` (visited on 11/16/2020) (cit. on p. 54).

[41] Cody Lindley. *Front-End Developer Handbook 2019*. URL: `https://frontendmasters.com/books/front-end-handbook/2019/` (visited on 11/16/2020) (cit. on p. 55).

[42] Cody Lindley. *JavaScript (ES2015+) Enlightenment*. URL: `https://frontendmasters.com/books/javascript-enlightenment/` (visited on 11/16/2020) (cit. on p. 59).

[43] *MobX*. URL: `https://mobx.js.org/index.html` (visited on 11/12/2020) (cit. on p. 61).

[44] *Openai/Gym*. OpenAI, Aug. 18, 2020. URL: `https://github.com/openai/gym` (visited on 08/18/2020) (cit. on p. 67).

[45] Manuel Menzinger and Gerald Steinbauer. “Design and Implementation of an AI Programming Playground for Schools.” In: *2nd International Workshop on Education in Artificial Intelligence K-12 in conjunction with the 21st International Conference on Artificial intelligence in Education* (2020) (cit. on p. 72).

Appendix A.

Survey

Teaching AI at School

This survey is about the general use of AI in computer science education. It is part of a diploma thesis about an online AI programming tool for education.
While this first part is about the general use of available AI teaching resources for k12-students, the second part is all about technical limitations at school concerning modern web tools.

In general the survey is anonymous, however it is possible to state interest to get early access to the tool.

Which general AI topics are you teaching or planning to teach in class?

- ☐ search (e.g. pathfinding, searching for the best action ...)
- ☐ knowledge representation and reasoning (e.g. structuring information to base decisions of [logic, chatbot, ...])
- ☐ reinforced learning (e.g. make an agent learn by trial and error)
- ☐ supervised learning (e.g. categorizing pictures of cats and dogs)
- ☐ ethics and social aspects (e.g. how is AI affecting us)
- ☐ Other:

How do you approach teaching AI?

- ☐ theoretical (discussing topics and their use/relevance)
- ☐ practical (implementing known algorithms to solve problems)
- ☐ experimental (trying to create own solutions to solve problems)
- ☐ using pen and paper exercises
- ☐ Other:

What are your reasons (if any) not to teach AI in class?

- [ ] not enough time
- [ ] can't fit into curriculum
- [ ] the content is too complex
- [ ] the requirement to install specific programs
- [ ] missing tutorials / courses
- [ ] more complex examples need more groundwork (e.g. to programm an AI for a game you need the game first)
- [ ] Other:

Are there any AI tutorials/courses or frameworks you have participated in or used yourself?

- [ ] EDLRIS AI Basic
- [ ] EDLRIS AI Advanced
- [ ] OpenAI Gym
- [ ] Elements of AI
- [ ] Google AI Education
- [ ] Coursera
- [ ] Udacity
- [ ] Crash Course
- [ ] Other:

Are you using or planning to use parts of any AI tutorial/course or framework in class?

- ☐ EDLRIS AI Basic
- ☐ EDLRIS AI Advanced
- ☐ OpenAI Gym
- ☐ Elements of AI
- ☐ Google AI Education
- ☐ Coursera
- ☐ Udacity
- ☐ Crash Course
- ☐ Other:

Which programming languages and libraries are you teaching in class?

- ☐ Python
- ☐ C/C++
- ☐ Java
- ☐ C#
- ☐ Javascript
- ☐ PHP
- ☐ Scratch/Lego Mindstorms or other graphical programming languages
- ☐ Other:

Next

Teaching AI at School

AI Online Platform

The upcoming online platform is all about providing an easily accessible tool to practice AI programming on practical examples. Therefore it provides different scenarios (like Tic-Tac-Toe, Wumpus, ...) and tools (keras [tensorflow], prolog, ...) to master them. While there are documentation and more detailed examples, its goal is not to teach the basics (which many other courses/tutorials do), but to enable everyone, without the need of any tools or registration, to play around by writing their own AIs.

The platform itself will be publicly and freely available for all by the end of the year, to get early access simply provide an email-address at the end. This address will only be used to send one email with basic instructions on how to use the framework.

Which web-browsers are available in your school?

- ☐ Chrome / Chromium
- ☐ Firefox
- ☐ Edge
- ☐ Opera
- ☐ Safari
- ☐ Other:

How often are your programs (especially web-browsers) updated in school?

- ○ at least once per month
- ○ at least once per semester
- ○ at least once per year
- ○ at least once every other year
- ○ it can take more than two years
- ○ i don't know

If any, which online programming tools do you use in class?

Your answer

If you want to gain early access to the platform, insert your email here:

Your answer

Back   Submit

Appendix B.

Feedback

AI Playground - Feedback

Thanks for providing your feedback which helps a lot to improve this project.

How many hours have you been using the framework for?

- ◯ < 1 hour
- ◯ 1-5 hours
- ◯ 5-10 hours
- ◯ 11+ hours

What have you been using the website for?

- ☐ Teaching AI to others
- ☐ Learning AI for yourself
- ☐ As a simple JavaScript editor
- ☐ Other:

How much do you agree with the following statements?

	fully agree	partly agree	partly disagree	fully disagree
I was able to navigate the website without problems	○	○	○	○
The scenario description files were very helpful	○	○	○	○
I experienced faulty behavior (bugs) while programming (like disappearing code, wrong error messages, ...)	○	○	○	○
The examples were very helpful	○	○	○	○
I was not able to figure out how to use the website, the documentation in general was insufficient	○	○	○	○
I like the agent based approach, it helps to focus on the relevant parts	○	○	○	○
I would like to see more scenarios of the same complexity	○	○	○	○
I would like to see more scenarios of higher complexity	○	○	○	○
I would like to see more scenarios	○	○	○	○

of lower complexity				
I think this tool is useful for teaching AI	○	○	○	○
I will be using this tool in class, if possible	○	○	○	○

What general features are you missing?

Your answer

What new scenarios would you like to see?

Your answer

Final thoughts and additional remarks:

Your answer

Submit