



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

*Distributed
Computing*



Improving the Interface of the Jass AI

Distributed Systems Laboratory

Nicolas Kupper

`nkupper@student.ethz.ch`

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:

Benjamin Estermann

Prof. Dr. Roger Wattenhofer

June 12, 2023

Acknowledgements

First of all I want to thank my supervisor Benjamin Estermann for all the great discussions, inputs on how to improve things, ideas for new features, testing feedback and guidance during this semester thesis. You truly created a very pleasant working environment and made this thesis a joyful experience. Furthermore, I want to thank my friend Nicola Bögli for his expertise and discussions about state of the art technologies and practices in the context of frontend development. I also would like to thank Roger Wattenhofer for testing new features and providing feedback.

Abstract

The Jass AI project is a platform where you can play Jass against a Jass AI. It is developed by the distributed computing group of ETHZ and is the result of several student projects. It has grown rapidly over many student projects. Leaving its frontend in a technological outdated state. A Npm install yields 151 security vulnerabilities. Furthermore, the project lacked essential frontend features and a modern development workflow. In this project the frontend part of the Jass AI was updated to a state of the art stack. As part of this update the outdated state management tool Flux was completely replaced with Redux. With the performed update all 151 security vulnerabilities were closed. As a next step a test environment as well as automated deployment pipeline for the test and the production environment were set in place. Furthermore, new frontend features were introduced, including an analyzer which lets users analyze their games and provides feedback on how accurate they played. To make sure all features are found on the page, onboarding tours guiding users through the page, were implemented. The Jass AI frontend is now on a modern software stack leaving it ready for development in further student projects.

Contents

Acknowledgements	i
Abstract	ii
1 Introduction	1
2 Package Upgrade	2
2.1 Node upgrade	2
2.2 React upgrade	3
2.3 Bootstrap upgrade	3
2.4 Replacing Flux with Redux	3
2.5 Upgraded Packages Summary	3
3 Development Workflow	5
3.1 Test environment	5
3.2 Automatic Deployment	6
4 New features	7
4.1 Analyzer	7
4.2 Onboarding Tours	13
5 Conclusion	19
Bibliography	20

Introduction

Over the years and many student projects the Jass AI project has grown rapidly. In these projects most of the focus was on the AI aspect of the project and not on the frontend part of it. This has left the frontend part of the project in an outdated technological state. This is a major security concern as those outdated packages contain many security vulnerabilities. It also makes developing harder as there are missing features and not state of the art practices involved with those old packages. Furthermore, the current development workflow is very manual which is really prone to errors and not very maintainable. The goal of this semester project is therefore to in a first step bring the technological stack of the frontend side to a modern stack. Then in a second step introduce a proper development workflow including a deployment pipeline. And finally, in a third step develop new frontend features.

Package Upgrade

A simple npm [1] install on the current package.json file yielded 151 security vulnerabilities of which 9 were categorized low, 37 moderate, 82 high and 23 critical. In this section the steps taken to upgrade to a software stack with 0 security vulnerabilities and newest versions of the core components of this project is described. The steps to achieve this goal were:

- Upgrade Node.js [2] to current LTS version
- Upgrade React [3] to v17 including all side packages
- Upgrade Bootstrap [4] to v5
- Replace Flux [5] with Redux [6]
- Upgrade React to current v18 version

2.1 Node upgrade

The project was running in a docker [7] container with Node.js (shortly referred to as Node) version 12.22.2 which comes with Npm (node package manager) version 6.14.13. Node 12 LTS was released in 2019 and the LTS support for it ended in 2022. When trying to run the project with the current Node LTS version 18.14 which comes with Npm 9, Npm raised a broken dependency graph error. This means that Npm was unable to resolve the listed dependencies from all declared packages as they contradicted each other. Concretely some packages demanded react v15 but React v16 was installed in the project. The reason this did not show up in the old node version is that Npm did not have transitive dependency checking prior to v7. In order to solve this broken dependency graph the packages demanding React v15 were upgraded to the version that used React v16. If the packages were discontinued they were either removed or replaced with similar packages achieving the same functionalities.

2.2 React upgrade

The reason the upgrade of React and all side packages had to be made in two steps is that one of the most essential core components of the project, the state management tool Flux is in maintenance only mode and only supports React up to v17. The state management tool is responsible for all data flow within the frontend such as keeping track of the state of the game and navigating the client through the page. It was therefore clear, that in order to get to a modern technology stack Flux will have to be replaced.

2.3 Bootstrap upgrade

Bootstrap is an open source css framework that is used for styling in the Jass frontend. The version in the project was v4 and the current version was v5. As this is a major upgrade, to no surprise it came with multiple breaking changes such as changes in the way components are spaced and renaming or removing of components which were used in the Jass project. As there is no "make bootstrap 5 look like bootstrap 4" solution, the affected components were replaced in a fashion where the looks of the page is as close as possible to what it looked like before.

2.4 Replacing Flux with Redux

As briefly mentioned Flux is in maintenance only mode and states on their Npm page *The Flux project has been archived and no further changes will be made. We recommend using more sophisticated alternatives like Redux, MobX, Recoil, Zustand, or Jotai.* Based on the popularity and the widespread usage it was decided to migrate to Redux. However, there is no easy migration guide as Flux and Redux have different architectures. One core difference is that Flux uses a global dispatcher which dispatches actions when called from the frontend to its different stores (multiple stores) while Redux only has one global store and uses so called reducers to change the state of the store which are called when actions are dispatched directly from the frontend. Therefore, one has to include Redux in the project and then port each action manually from Flux to Redux. In the Jass project this were 61 actions with in total around 1200 lines of code.

2.5 Upgraded Packages Summary

A minor version upgrade is an increment of the second number of a package version so for example from 12.2 to 12.3. This usually means no manual intervention

has to be made while a major version upgrade, which is an increment of the first number, for example from 12.2 to 13.0 means that there are breaking changes which needs manual intervention. In the following table all packages that have been upgraded are listed. If a package was not upgraded, was removed from the project or only a minor upgrade was done it is not listed. If a package was introduced newly to replace a discontinued package it will be written rdp (replaced discontinued package).

Package name	Old version	New version	Major upgrades
@babel/core	6.24.1	7.21.0	1
babelify	7.3.0	10.0.0	3
bcrypt	-	5.1.0	rdp
browserify	14.3.0	15.2.0	1
bootstrap	4.5.0	5.2.3	1
core-js	-	3.29.0	rdp
express-handlebars	5.3.4	6.0.7	1
glicho2	0.8.4	1.1.0	1
less	3.5.1	4.1.3	1
mysql2	2.3.2	3.2.0	1
react	16.13.1	18.2.0	2
react-bootstrap	1.3.0	2.7.2	1
react-router-dom	5.2.0	6.9.0	1
react-transition-group	1.2.0	4.4.5	3
react-redux	-	8.0.5	rdp
redux	-	4.2.1	rdp
ws	7.3.1	8.13.0	1

Development Workflow

There was neither test environment nor a deployment pipeline in place in the Jass project. This means that developers wrote code locally. When they wanted to deploy it to the server they had to manually connect to the server, checkout the git repository, set the environment variables such as the MySQL [8] password and google OAuth keys and then build the docker containers and run it. Not only is this a very time consuming workflow but also it requires the developer to have access to the server, have the knowledge on how to build and deploy the docker container and forced them to access essential passwords such as the MySQL root password and the OAuth secret keys. Furthermore, if errors occurred which were not present in the local environment this was only found out after deploying onto the live web page.

3.1 Test environment

To avoid finding errors that only occur on the server side just after deploying to the production, a test environment was needed. For this a new Google OAuth client appropriate was setup for the test environment. This is needed to make logins with the google account possible. Furthermore, the IT team from ETHZ TIK made an ETH internal dns entry for <http://test.jass.ethz.ch> which points to the server where the Jass application is running. After this a new site was configured in the webserver (NGINX) [9] which listens on test.jass.ethz.ch and forwards its traffic to port 3002. Furthermore, the docker configuration was extended with two additional containers for the test environment, the application container on port 3002 and an additional mysql-test container for a separate database from the production database. Docker compose profiles were used to decide which environment to start or stop. For example

```
docker-compose --profile test up
```

will only start the two docker containers for the test environment (application and database).

3.2 Automatic Deployment

To address the issue of developers needing to have the knowledge and access to passwords an automatic deployment was needed. For this the server where the Jass application is running was configured as a gitlab [10] runner. Then in gitlab following variables were configured:

Variable	Environments	Description
MYSQL_PASSWORD	prd, test	Jass database mysql password
MYSQL_ROOT_PASSWORD	prd, test	Jass database mysql root password
MYSQL_URL	prd, test	mysql (prd) or mysql-test (test)
GOOGLE_CLIENT_ID	prd, test	Credentials for google OAuth
GOOGLE_CLIENT_SECRET	prd, test	Credentials for google OAuth
PORT	test	Custom port for test
SSL	prd	SSL for websocket communication
CERT	prd	Certificate for SSL
PRIVKEY	prd	Private key for certificate

Afterwards a .gitlab-ci.yml was created with two environments: test and prd for the corresponding docker containers. The deployment script gets the variables from gitlab and then builds and starts the docker containers. Following rules are enforced for the separate stages:

- build-deploy-test - Trigger: Commit branch \neq master
- build-deploy-prd - Trigger: Commit branch = master
- cleanup - Trigger: Always

This means that whenever a developer pushes a commit to a branch which is not the master branch, this state will be automatically checked out and deployed on the test environment <http://test.jass.ethz.ch>. Furthermore, as the master branch is a protected branch and one can not directly push to it, this also means that after an approved merge request to the master branch the changes are directly deployed to the production page <https://jass.ethz.ch>. The cleanup job is used to remove unused docker containers in order to free up disk space. A permission issue occurred during the build where the docker clean up left files with root privileges in the build directory of the gitlab-runner user. This caused the next build to fail with permission denied errors. To fix this, a Linux access control list (ACL) was used to make sure that no matter who creates a file in the build directory of the gitlab-runner, it always belongs to the group gitlab-runner.

New features

4.1 Analyzer

Losing in a game of Jass is frustrating but what is even more frustrating is losing over and over again without knowing what you could have done better. For this reason an analyzing feature for past games was urgently needed. Following things had to be decided:

- When and how to analyze the game?
- What analyzing information should be displayed to the user?
- Where and how should this information be displayed?

Analyzing the game

There were two particular options when to analyze games. Either the games are analyzed upon saving them to the database and the analyzing information is saved to the database or the games are analyzed in real time during the replay. Since the first option would mean that games that were played in the past could not be analyzed and also that whenever a better engine comes out to analyze the games, the old games could not benefit from the new engine, it was decided to analyze the game during the replay.

The best AI agent trained in previous theses and used to currently play Jass will also be used as feedback for the performance of the players. This agent is written in Python [11]. However, the frontend application is written in node and react. This means that for analyzing the game a Python agent has to be launched and communicated with upon playing moves as each application (the Python agent and the frontend) keeps track of the game state on its own. Meaning whenever a move is played it is broadcasted on the websocket where the frontend application as well as the Python agent are listening on. Upon receiving this message the Python agent or the frontend application updates its game state. As one can see these two states have no knowledge of each other and are only synchronous if they

updated their states in the same matter. This caused major problems because if a message is for some reason lost in the communication or is sent in a different order the state of the two applications can differ which breaks the communication flow and with that stops providing analyzing information because the frontend application might for example request an evaluation for a state which is not even reached yet in the Python agent. To fix those issues following measures were implemented:

- Evaluations are only requested on replaying the next card but not on replaying the previous card. This is because when going back in a replay, all turns prior to the current turn have already been analyzed.
- The full game round is analyzed upon joining the replay. For this the game is quickly played through and then returned to the first move. This is done behind a loading animation [4.1](#) and not visible to the user.
- A queue has been implemented on the frontend side that checks if the evaluation information received is successive to the previous one. If not the evaluation is added to the queue. Upon receiving an evaluation the queue is checked. This fixes the issue when an evaluation for a turn is returned before another one but the turn is later in the game. For example the evaluation information for turn 0 and 1 are both requested at the same time but the information for turn 1 is returned before turn 0. This issue can arise due to asynchronous messaging.
- Sometimes when loading a new round the state of the Python agent was not quick enough to update before the request for the evaluation of the first few turns of the new round came through. This resulted in losing the evaluation information for the first or/and second turn and hence in being stuck when displaying the evaluation information as they are calculated based on previous turns. To fix this issue the round is restarted and the analyzing information is freshly requested whenever this happens. This again happens during the loading animation and is not visible to the user.

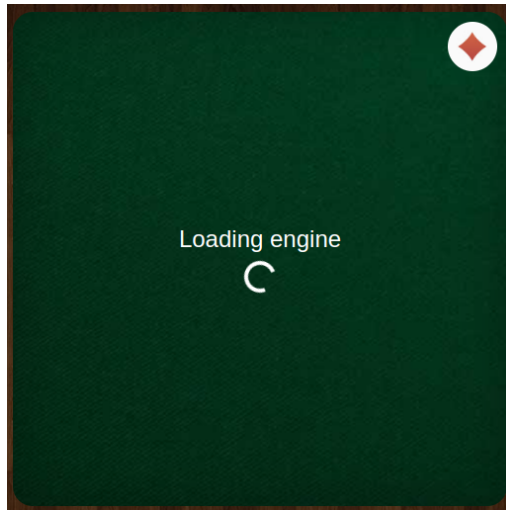


Figure 4.1: Loading animation

Analyzing information displayed

When displaying information to users there is always a small degree of displaying too much information in a sense that it overwhelms users and not displaying enough information in the sense that it has not a lot of use to them. It was decided to display the following information to the user:

- A table that shows the accuracy of each player.
- A table that shows analyzing information for the current turn.
- A history for each player which makes it possible to see how he performed in which turn.

The accuracy table shows for each player the weighted sum average of the accuracy up to this turn. To calculate this the information provided by the AI agent was used. The AI agent provides for a given turn the probabilities of playing each possible card.

Accuracy Overview		Current Turn
Player	Accuracy ⓘ	Details
Nicolas Kupper	34.20%	↗
Karoline	100.00%	↗
Andreas	100.00%	↗
Oskar	100.00%	↗

Figure 4.2: Accuracy table

This accuracy is calculated in the following way:

$$\overline{X} = \frac{1}{k} \sum_{i=1}^k \frac{p_c}{p_b}$$

Where k is the current turn, starting at 1, p_b is the probability of the best card and p_c is the probability of the card that was played.

The current turn table shows the probability for each possible card for the current turn. The cards are sorted from best to worst card. The card that will be played next is highlighted. This allows the user to already compare the card he played to the other cards he could have played. Hovering over a card will magnify it.




Accuracy Overview		Current Turn
Card ⓘ	Probability ⓘ	
	50.88%	
	25.98%	
	16.44%	

Figure 4.3: Current turn table

In the history panel all previous turns can be seen. For each turn the turn accuracy as well as the sum accuracy up to that turn is shown. Furthermore, the card played and the best card for this given turn is shown. There is also a jumpto icon that will resume the replay to that turn.










Nicolas Kupper				
Turn	Played Card	Best Card	Turn Accuracy	Sum Accuracy
1 			2.59%	2.59%
2 			0.00%	1.30%
3 			100.00%	34.20%

Figure 4.4: Player cards history

Placement and space concerns

As one can see the replay section is already pretty overloaded and there is not a lot of space left for analyzing information.



Figure 4.5: Replay overview

To make the best out of the available space it was necessary to make use of space saving features. The first of those is using tabs to switch between the accuracy table and then current turn view. The active tab is always highlighted.

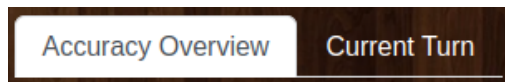


Figure 4.6: Tabs

The next feature that was used is in the current turn view. In the worst case, from a space concerning point of view, there are nine cards that can be played. Displaying nine cards would totally overfill the user interface. To solve this problem a pagination was introduced which displays up to three cards per page.

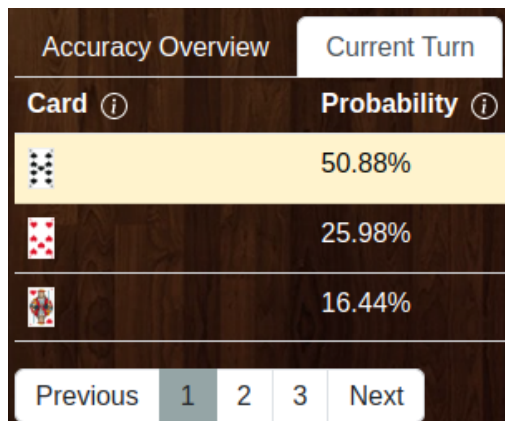


Figure 4.7: Pagination

To display the player card history a modal was used. The modal can be reached clicking on the goto icon in the accuracy overview tab [4.2](#).

4.2 Onboarding Tours

The best feature has no use if nobody knows about it. For this particular reason a new feature was needed that guides an user through the page and the features that are available in the page. A lot of things on the page like "Quick Game" or "Create a Table" are pretty self-explanatory. However, other things like finding your past games, replaying your past games and analyzing a game are a bit more complex. It was decided that two onboarding tours will be created for the page:

- Main page onboarding tour that guides logged in users to their past games
- Replay onboarding tour that guides the user through the analyzing feature

The onboarding tours were implemented using the package react-joyride [\[12\]](#).

Main page onboarding

After logging in the user is created with the onboarding message.

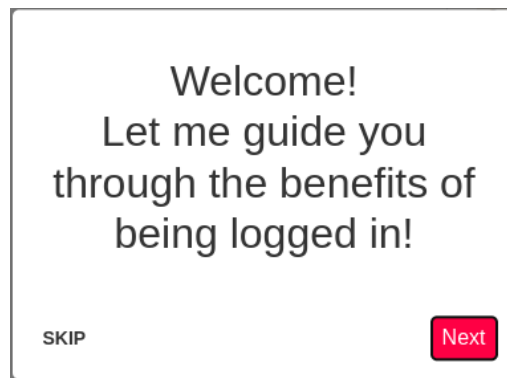


Figure 4.8: Step 1

Clicking the SKIP button will immediately exit the onboarding tour. Clicking the Next button will lead to step 2. In step 2 the username of the logged in user is highlighted with the following message:

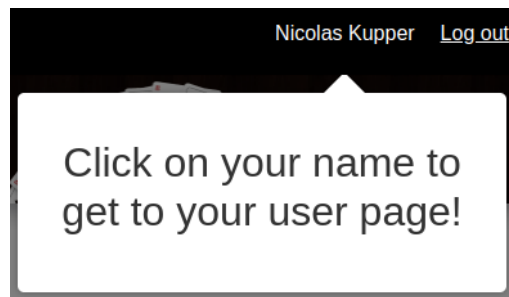


Figure 4.9: Step 2

After clicking on the username, the user is taken to his user page and is informed about his personal stats.

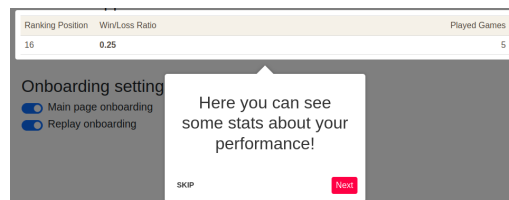


Figure 4.10: Step 3

In the next step the user is informed on how to disable or enable the onboarding tours. The onboarding tour settings are saved as cookies in the browser.

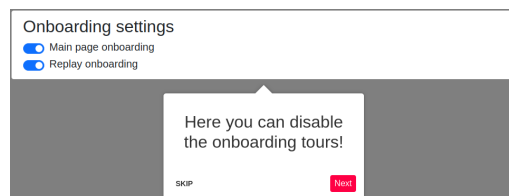


Figure 4.11: Step 4

In the final step the user is informed on where to find his past games and the possibility of analyzing them.

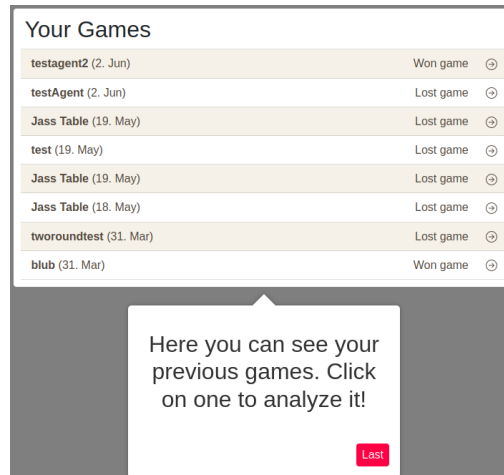


Figure 4.12: Step 5

Replay onboarding

Upon joining the replay the user is welcomed to the replay onboarding tour.

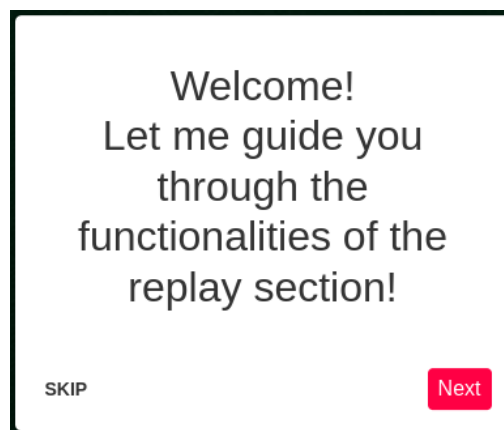


Figure 4.13: Step 1

Again pressing the SKIP button will immediately exit the tour.

In the next step the user is informed about the possibility of enabling the analyzing info by pressing the switch.

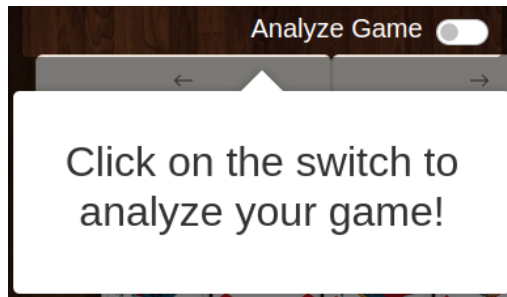


Figure 4.14: Step 2

Upon pressing the switch the user is taken to the slider which can be used to jump to a specific turn.

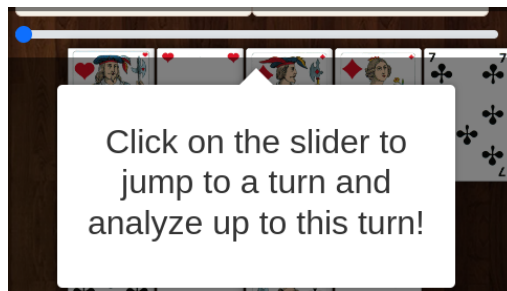


Figure 4.15: Step 3

Once a position has been clicked on the slider the replay jumps to this turn and the onboarding tour highlights the accuracy table.

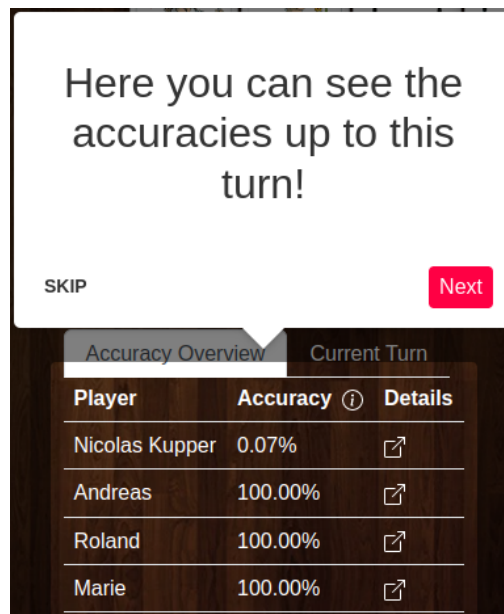


Figure 4.16: Step 4

As a next step the user is informed that he can click on the detail icon.

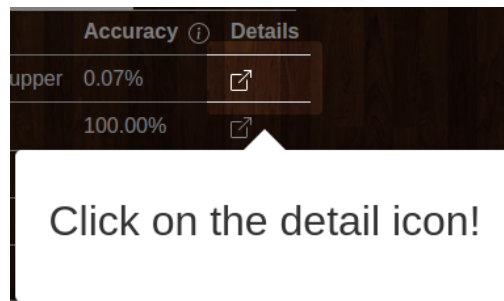


Figure 4.17: Step 5

Once clicking the detail icon, the card history modal is opened and highlights the first turn.

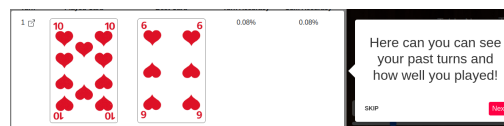


Figure 4.18: Step 6

As a next step the goto icon of the first step is highlighted.

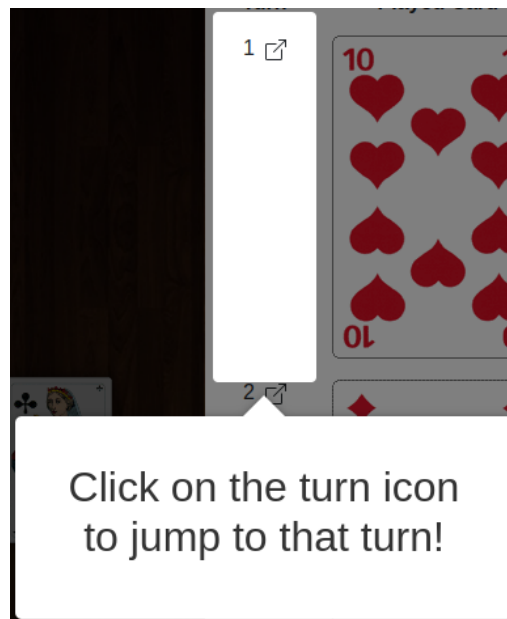


Figure 4.19: Step 7

On click the goto icon, the user is taken to the first turn and the "Current Turn" tab is highlighted.

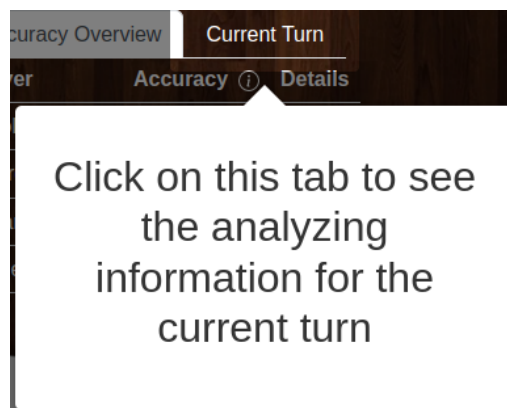


Figure 4.20: Step 8

A click on the current turn tab will end the replay onboarding tour and show the current turn information as described in chapter 4.1.

Conclusion

With the package upgrade the frontend of the Jass AI project is now on a modern software stack without any known security vulnerabilities. The newly introduced test environment together with the automatic deployment pipelines makes developing and testing much easier. The new onboarding tour feature ensure that users make full use of all features available. The analyzer provides helpful feedback and lets users improve on their Jass skills. In conclusion we severely improved the software maintainability of the Jass AI frontend and it is ready for further student projects.

Bibliography

- [1] “Npm,” <https://www.npmjs.com/>, accessed: 30.05.2023.
- [2] “Node.js,” <https://nodejs.org/>, accessed: 30.05.2023.
- [3] “React,” <https://react.dev/>, accessed: 30.05.2023.
- [4] “Bootstrap,” <https://getbootstrap.com/>, accessed: 30.05.2023.
- [5] “Flux,” <https://facebookarchive.github.io/flux/>, accessed: 30.05.2023.
- [6] “Redux,” <https://redux.js.org/>, accessed: 30.05.2023.
- [7] “Docker,” <https://www.docker.com/>, accessed: 30.05.2023.
- [8] “Mysql,” <https://www.mysql.com/>, accessed: 30.05.2023.
- [9] “Nginx,” <https://www.nginx.com/>, accessed: 30.05.2023.
- [10] “Gitlab,” <https://about.gitlab.com/>, accessed: 30.05.2023.
- [11] “Python,” <https://www.python.org/>, accessed: 30.05.2023.
- [12] “React joyride,” <https://react-joyride.com/>, accessed: 30.05.2023.