IT-Studienprojekt am Institut für Intelligente Informationssysteme (MSc)

Dokumentation VISAB 2.0

Weiterentwicklung eines Tools zur Visualisierung von KI-Agenten in Unity-basierten Spielen mit Fokus auf der Unterstützung von verschiedenen Spielimplementierungen über generalisierte Schnittstellen

Moritz Fröhlich, 353444
Marcel Gaal, 278640
Tim Kunold, 270966
Leon Römer, 263360
Vanessa Schriefer, 267453

Betreuer:

Prof. Dr. Klaus-Dieter Althoff
Dr. Pascal Reuss

Arbeitsgruppe Intelligente Informationssysteme • Institut für Informatik
Universität Hildesheim • Universitätsplatz 1 • D-31141 Hildesheim

Contents

List of Figures

1 Introduction

VISAB stands for**Vis**ualizing**A**gent**B**ehaviour and can best be described as a tool that is capable of communicating with games to extract game information und illustrate it for AI developers to have a deeper insight into the bot implementation's behaviour with perfectly tailored views for each specific game. It was developed with the intent of supporting an AI learning platform at the University of Hildesheim, IIS. The first version already got a publicitation at the ICCBR conference 2020, whereas the second (current) version participated at the ICCBR workshop 2021. A further publicitation at the ICCBR conference will very likely follow in year 2022.

VISAB has grown from a prototypical application to a kind of toolset that is completely maintained on GitHub and even has a hosted online documentation.[1] This document serves as a technical whitepaper which should make further development on the VISAB toolset easier by explaining the underlying architectural structures, how and why new features are implemented, what hurdles we came across, how to use VISAB from user perspective as well as providing an evaluation and conclusion.

Please note, that the cover page and requirements in the evaluation chapter are formulated in german language to have a more direct association to the course assignment, however the rest is described in english to help a wider range of people with this documentation for using and developing VISAB.

[1] GitHub: `https://github.com/VISAB-ORG`, Online Documentation: `https://visab-org.github.io/`.

2 Documentation

As mentioned in the introduction, this documentation merely serves as a grading basis with a more extensive version being hosted online at `https://visab-org.github.io/`. Parts of documentation (e.g. code samples) have been left out, which is why we would generally prefer you to read the online documentation instead.

a) Architecture

(i) Context Overview

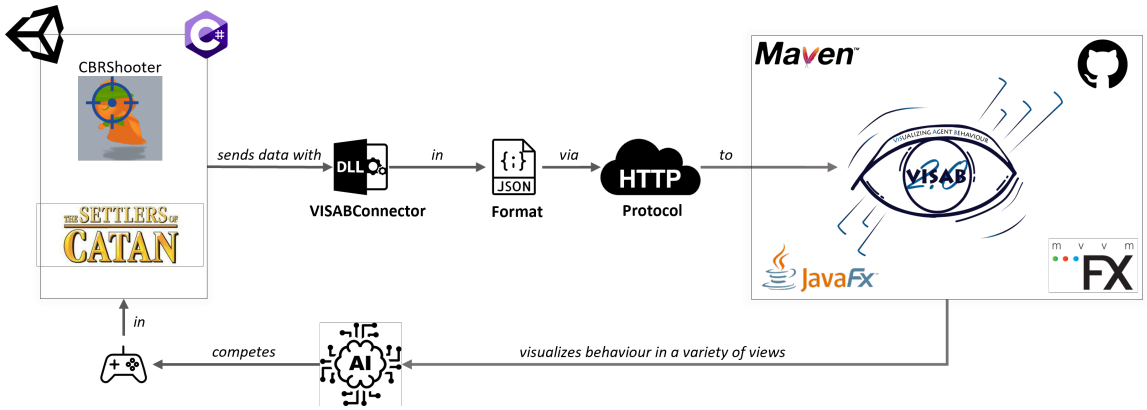

Figure 2.1: VISAB Context Architecture

In order to give a very quick and abstracted overview of how VISAB can be seen in context, the following graph shall illustrate the most relevant anchor points of VISABs current state and components involved in the overall workflow.

(ii) Module Views

Workspace The **workspace** package serves the purpose of persisting and accessing persistant data at runtime. The **Workspace** class functions as an access provider for the **DatabaseManager** and the **ConfigManager**. The managers have functionality for loading, modifying and saving data such as the settings or VISAB files. All IO operators are done using the specific repository classes. Settings, VISAB files, and log files are persisted in a dedicated `workspace/` directory, that can be administrated using the GUI. The workspace is also used to access the game specific class mapping that is set in the ‘classMapping.json‘ file.

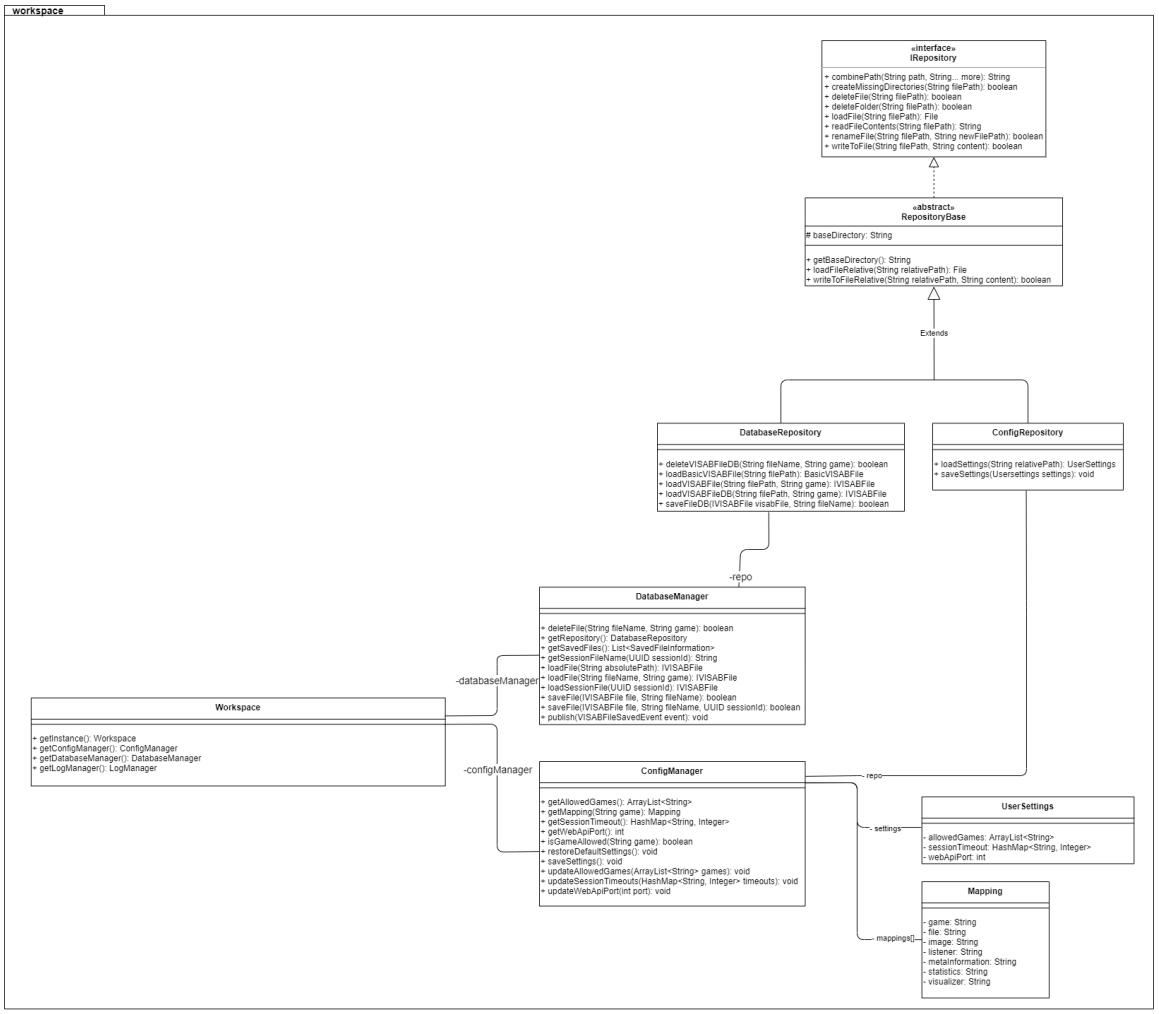

Figure 2.2: Workspace Package View

(iii) Component Connector View

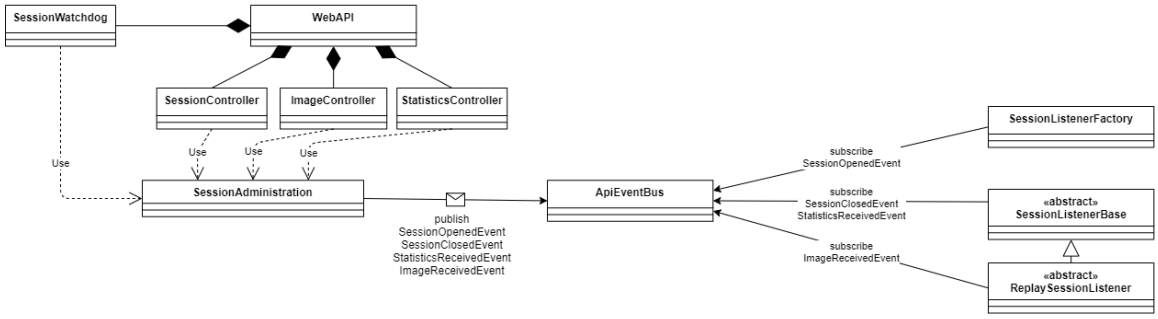

Figure 2.3: WebAPI - Processing Connection View

To achieve seperation of concerns, loose coupeling and enable scalability (through asynchronous event publishing) VISAB simply delegates information received to the processing modules using the Publish-Subscribe design pattern.

The WebApi, or rather the specific controllers, forward information received through HTTP requests to the SessionAdministration. The SessionAdministration then creates a new event by using that informtion and publishes it to the APIEventBus. The APIEventBus then forwards that event to the subscribes of the event type.

There are three classes in VISAB that subscribe to API events.

1. **SessionListenerFactory** Subscribes only the SessionOpenedEvent and creates a new ISessionListener instance on occurance.
2. **SessionListenerBase** Subscribes the SessionClosedEvent and the StatisticsReceivedEvent.
3. **ReplaySessionListenerBase** Additionally subscribes to the ImageReceivedEvent.

For more information regarding the specific types of API events, refer to the the graphic below.

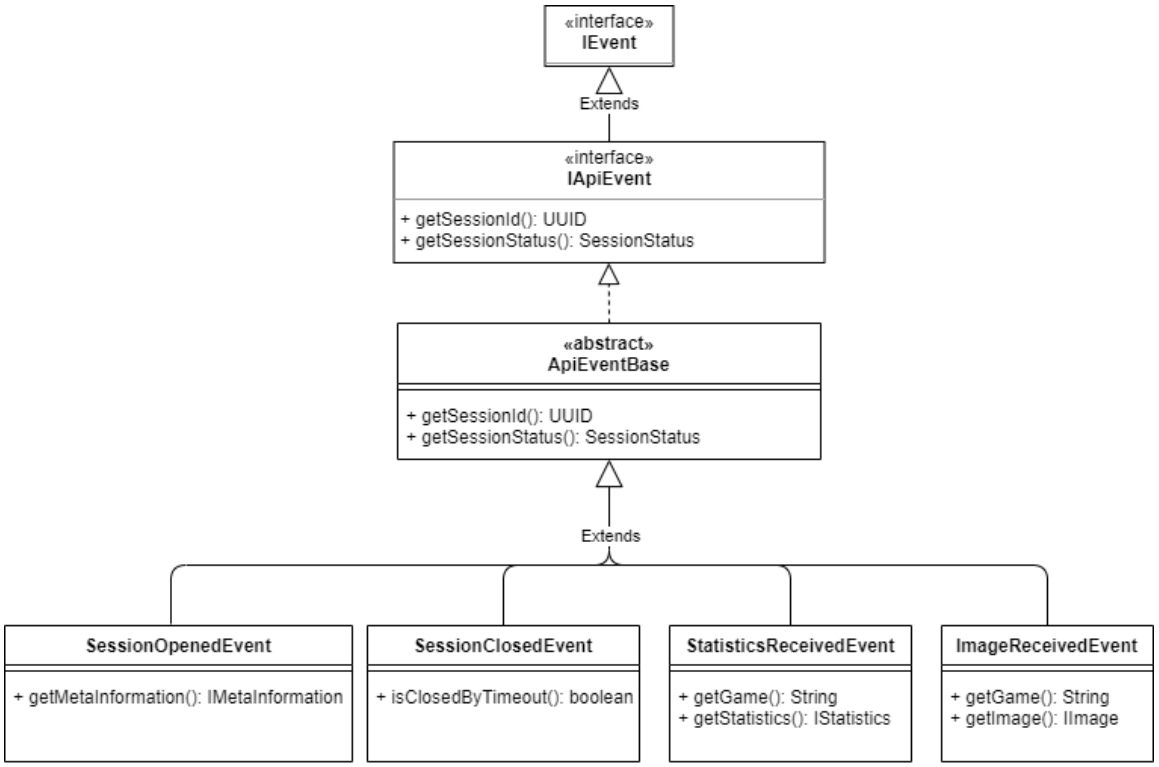

Figure 2.4: API Event Overview

(iv) Allocation View

For communication between game and VISAB, VISAB serves as the server and the games as the clients. Whilst it is possible, to run VISAB and the games on different machines, it is ofcourse not mandatory. VISAB expects the data contained in the HTTP requests body to be serialized as a JSON string. Instead of writing json string literals, you may use one of the many JSON serialization libraries availible for most languages. Games implemented in C# can use the VISABConnector library that provides a C# interface to VISABs WebApi. Additionally, if the game is implemented on top of the UnityEngine, VISABConnector.Unity may be used to perform Unity specific image extraction. Games implemented in Java can use the VISAB-JConnector library, that serves the same purpose as the VISABConnector just for java. This library is only a prototype currenlty, meaning that not all functionality is guaranteed to be working and the code base does not conform to Java standards. Language independant communication can be achieved through creating HTTP requests directly or optionally writing an interface in the language.

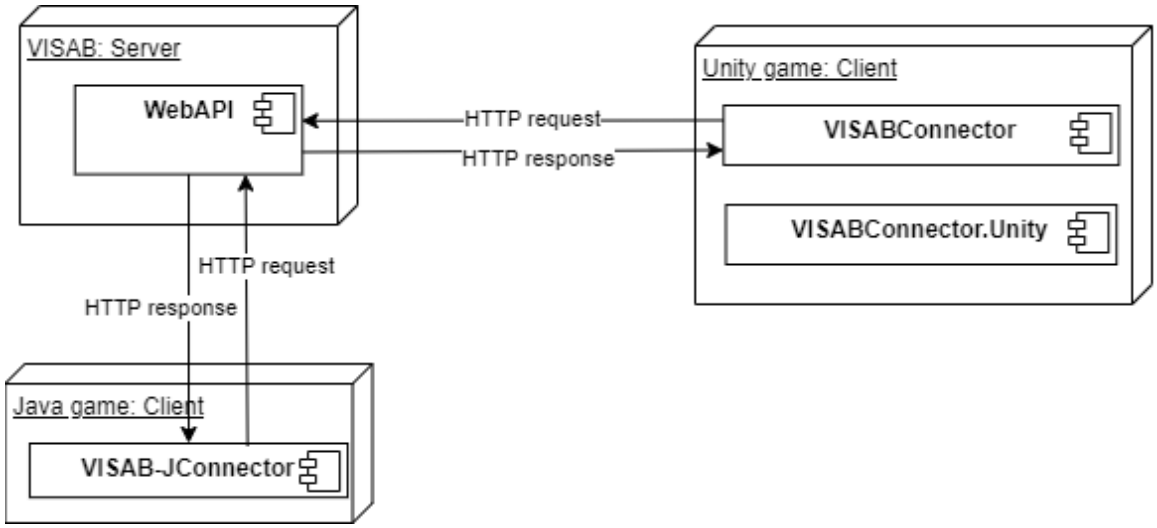

Figure 2.5: Deployment View

b) Development Environment

VISABs source control is done using Git and a remote repository is hosted at `https://github.com/VISAB-ORG/VISAB`. Releases of VISAB include both the jar and the source files. Download the latest version `https://github.com/VISAB-ORG/VISAB/releases`.

Installing VISAB A JRE version equal or greater than Java 11 is required for using VISAB. If that requirement is satisfied you can start VISAB from your shell in two modes.

GUI This mode provides features for visualizing game data files and starts the HTTP API for reciving data.

```
java -jar VISAB.jar -mode gui
```

Headless This mode only starts the HTTP API for reciving data.

```
java -jar VISAB.jar -mode headless
```

Developing VISAB A JDK version equal or greater than Java 11 and Maven (`https://maven.apache.org/`) are prequisites for developing VISAB.

Running the application

1. Clone the repository using HTTPS
   `git clone https://github.com/VISAB-ORG/VISAB.git`.

2. Run `mvn compile`.
3. Configure `org.visab.main.Main` to be the target of your run configuration.

You can now run VISAB from your IDE.

Developing VISABConnector and VISABConnector.Unity .Net Framework 4.8 (`https://dotnet.microsoft.com/download/dotnet-framework/net48`) and a C# IDE (Visual Studio, Visual Studio Code, ...) are prequisites for developing VISABConnector and VISABConnector.Unity.

Building the assemblies

1. Clone the repository using HTTPS
   `git clone https://github.com/VISAB-ORG/VISABConnector.git`.
2. Open the solution file `src/VISABConnector.sln`.
3. Build the solution using your IDE.

The compiled assemblies appear in the `bin/` directory of the project directories.

Developing the Unity games If you want to further develop the Unity games or change what data is transmitted from the games to VISAB you need Unity 2021.1.1 `https://unity3d.com/de/unity/whats-new/2021.1.1`. Afterwards you can simply open the game projects using the Unity Editor.

c) GUI Usage

Excited to use VISAB and get some insights about what it is capable of? This guide will provide a walkthrough to VISABs UI and explain all relevant elements of it.

Home

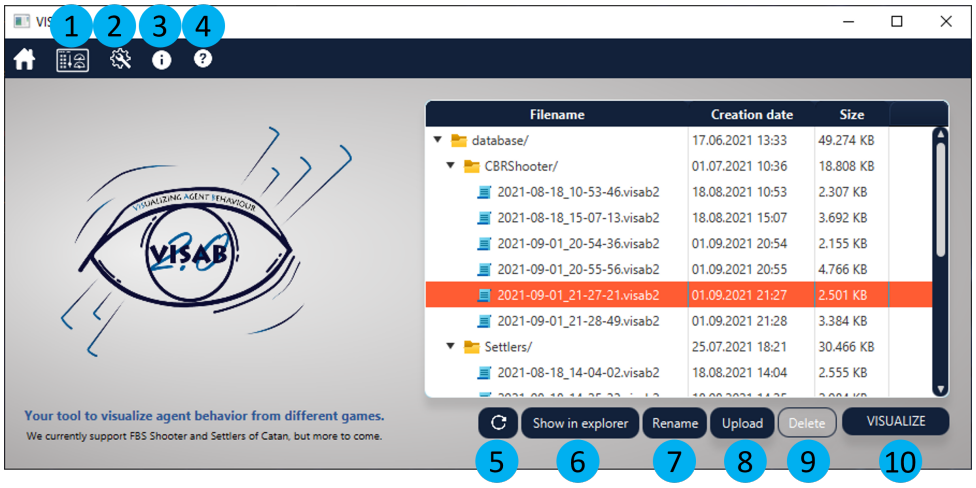

Figure 2.6: Home View - Light mode

The “Home View” has a navigation bar with buttons to open the API-Dashboard **(1)**, the Settings **(2)** with an included option to switch between light and dark mode **(11)**, the About **(3)** and the Help **(4)** window. It also shows a view of the file-system of the local VISAB database. There are all files with filename, creation date and size displayed, that belong to the VISAB supported games. The refresh button **(5)** can be used to reload the database. With the “Show in explorer” button **(6)** the file system will be opened where the selected file or folder is stored on the system. With the “Rename” button **(7)** a file or folder can be renamed. The “Upload” button **(8)** opens the systems file explorer, where the user can upload an existing VISAB file to the workspace. This is also possible via drag and drop into the HomeView. With the “Delete” button **(9)** the selected file or folder can be deleted. The “VISUALIZE” button **(10)** will open the Visualizer view if a VISAB file is selected.

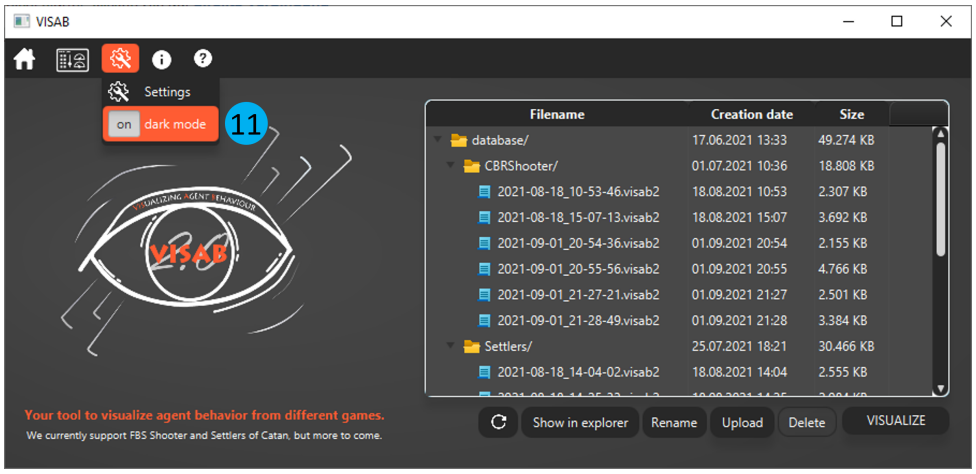

Figure 2.7: Home View - Dark mode

Visualizer

When a user selects a file from a game in VISAB and presses the "VISUALIZE" button **(10)**, the Visualizer view is loaded. The view contains three tabs **(1)**: File Overview, Statistics and Replay.

- The **meta view** is used to provide static information of the VISAB file which are valid by the end of it.
- The **statistics view** aims at providing cumulated information for each player that is well comparable.
- The **replay view** as a counterpart to the statistics view serves the purpose of re-experiencing specific moments of the game over and over whilst highlighting information according to the users needs.

Because the visualizers are designed similarly for both games, they will be described once with highlighted differences.

Meta View

The "Meta View" provides information about the respective file itself, the underlying session information as well as static game information such as the players and how / by whom they are controlled. The "CBR Shooter Meta View" further contains weapon information.

Any other static data could also be displayed here.

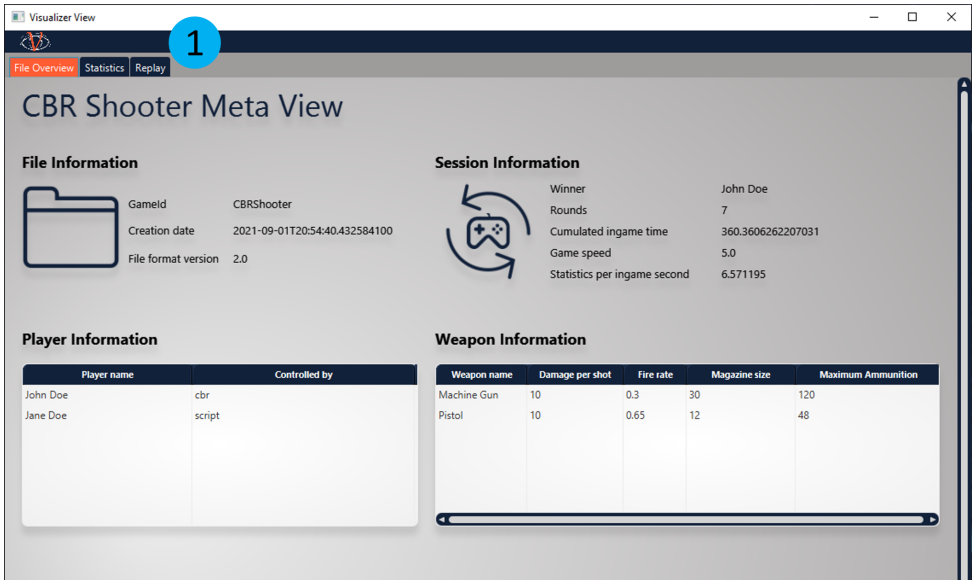

Figure 2.8: CBRShooter Meta View

Statistics View

The "Statistics view" consists of a comparison statistics table with a "Show in chart" button **(1)**, a comparison graph **(2)** and a section with pie charts **(3)** indicating the plan usage distribution for each participating player / bot. For Settlers of Catan, there is also a "Show Resource Details" button **(4)** available, that is capable of giving a more complex insight about only the resource-related rows of the comparison statistics table in a Detail View.

In the comparison statistics table, the statistics for each measure are compared between players and can be viewed in a line graph when a line is selected and the "Show in Chart" button **(1)** is pressed.

Figure 2.9: Settlers Statistics View

The “Detail View” illustrates the resources for each player in each round are displayed in a stacked bar chart with different colors for every resource. The displayed rounds can be changed (*in intervals of 10 rounds*) by moving the slider **(1)** back and forth.

Please note that JavaFX reassigns the colors for each resource when the slider is moved to another interval.

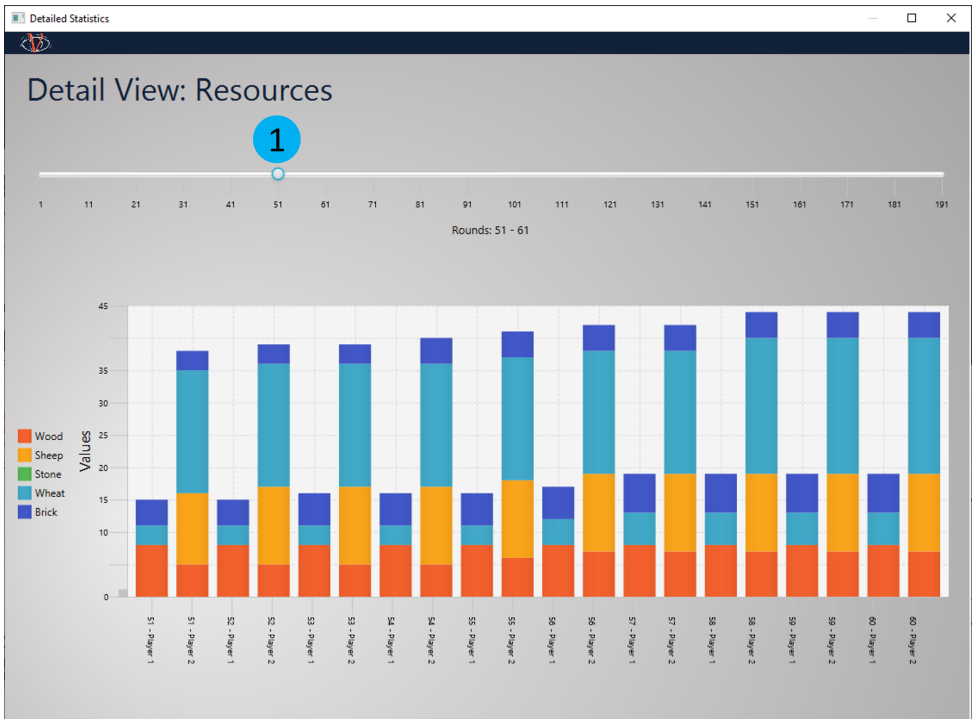

Figure 2.10: Settlers Statistics Detail View

Replay View

The "Replay View" displays the map of the underlying game data. Furthermore it has several capabilities of modifying the visible content to enhance the users focus on specific aspects. As the name already indicates, the view can be played and paused by pressing the designated button **(1)**. Additionally the velocity (speed in which the map data is updated) can be adjusted with a slider **(2)**. If you want to jump to a specific moment of the game data, you can also directly use the turn slider **(3)**. The visuals table **(4)** gives information about player-specific color-coded visuals and offers the possibility to hide certain objects or even the whole player by simply un-checking the respective check boxes. The color-coding is implemented to optimize the visual distinction between the different players and relies on colors provided by the game itself. If the color-coded items are hardly separatable from the underlying map, there is also an option to check "Show in black and white" **(5)** to grey-scale the map image. The CBRShooter also contains items that are not player-specific, which can also be hidden by un-checking their respective check boxes **(6)**. The "light-blue bubbles" indicate the changes applied to the view by utilizing the previously mentioned options.

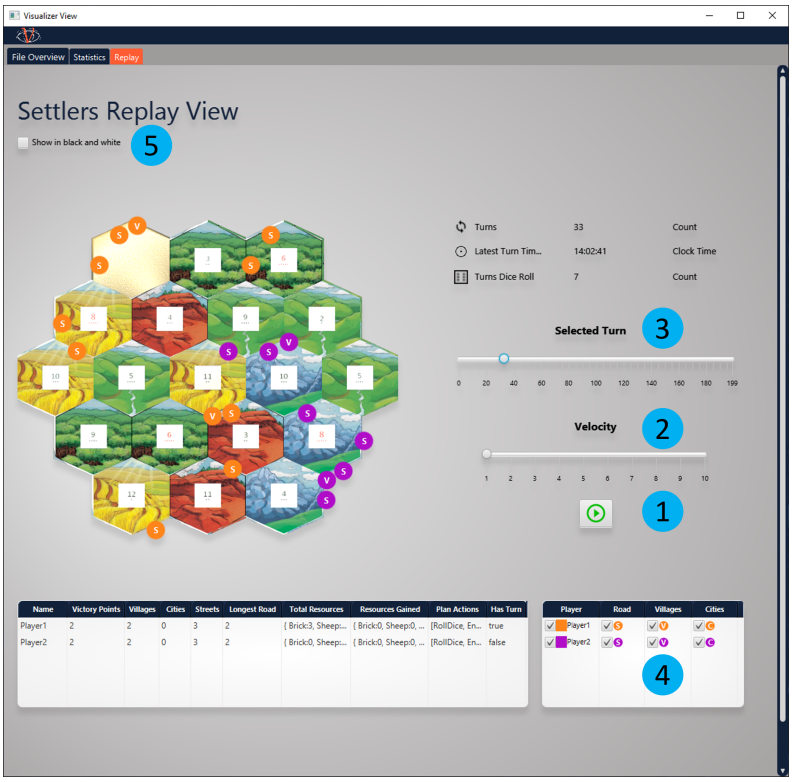

Figure 2.11: Settlers Replay View - All enabled

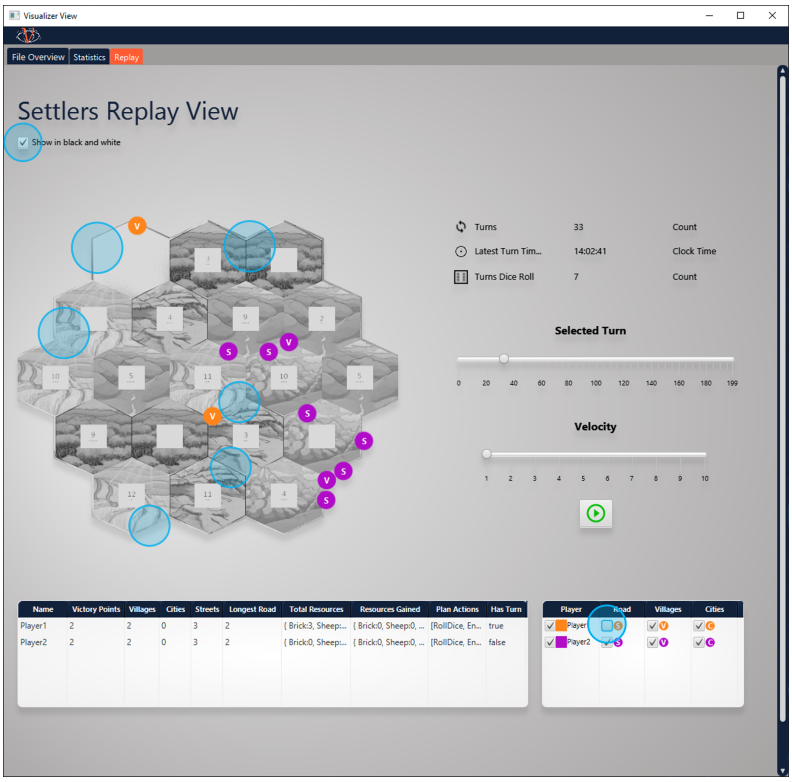

Figure 2.12: Settlers Replay View - Some hidden

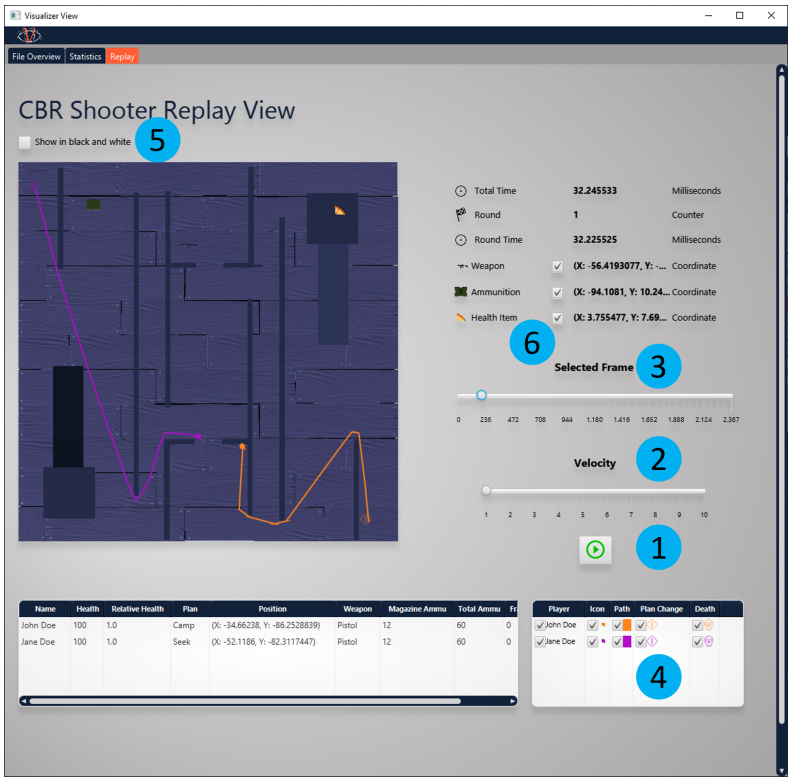

Figure 2.13: CBRShooter Replay View - All enabled

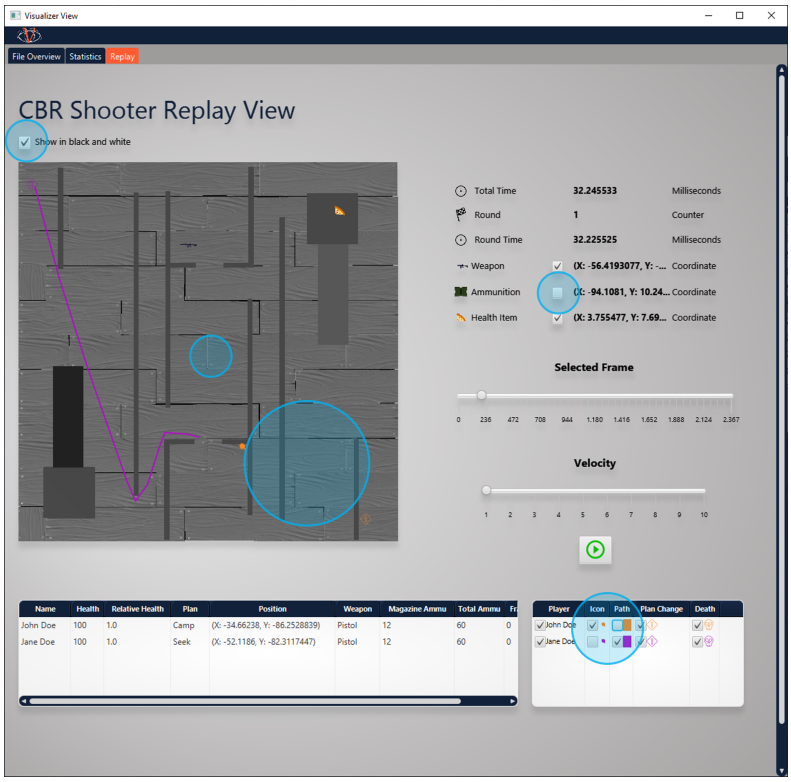

Figure 2.14: CBRShooter Replay View - Some hidden

API Dashboard

The “API Dashboard” exposes the most important informations of VISABs underlying HTTP API which is used as the communication interface with the respective games. To configure the connection in the game correctly, VISABs adress can be directly copied from a text field **(1)**. Below you will find an overview of all transmission sessions and their respective status and details. By clicking the “Clear Inactive Sessions!” button **(2)** all sessions that are either timeouted (grey) or canceled (red) will be removed from the overview. Any active (green) session can be canceled from VISAB side by clicking the “X” button **(3)**. VISAB also offers the option to show the Visualizer view for a session in a live mode [2] **(4)** while it is still running. After a session has ended, the “Open Live View” button **(4)** changes into a “Visualize” button **(5)** which also opens the Visualizer view just as you do from the Home view file explorer. For each session several informations are shown like the name of the game, the ip-address, the opening time of the session, the time of the last request, the time when the session was closed and the total amount of request that were sent in the during the transimission session.

[2] The live mode also uses the same visualizers as the regular mode, but the data updates are performed incrementally on each request done by the game, which gives a nice look and feel for the charts being dynamically updated and the replay view more or less being on the same status of the real running game.

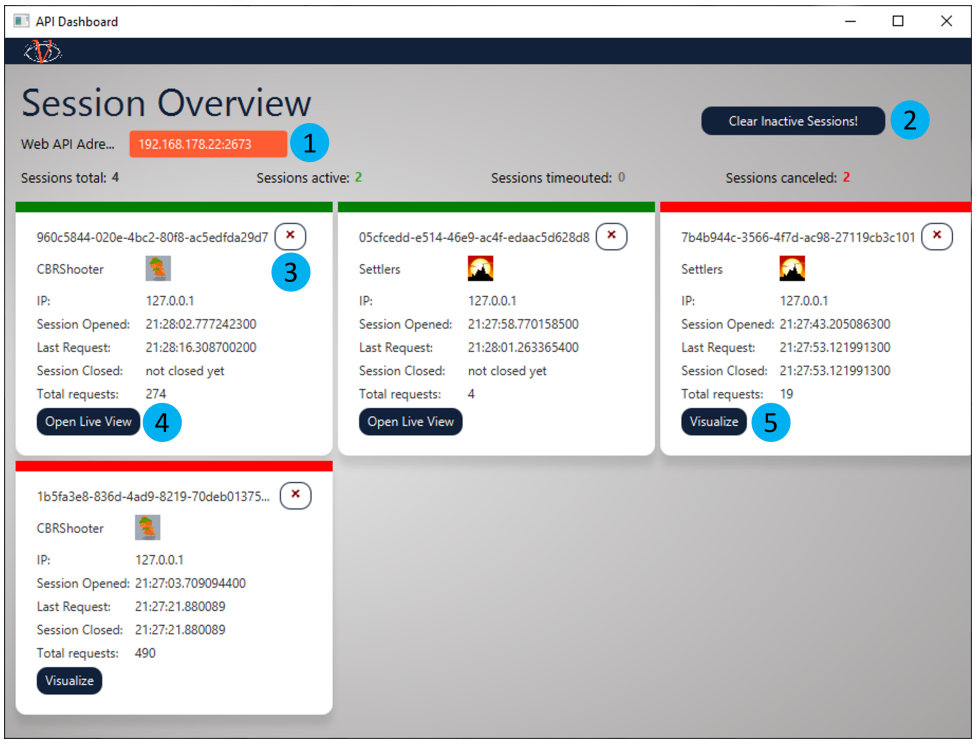

Figure 2.15: API Dashboard View

Settings

The “Settings View” offsers several options to configure VISAB. The Web Api Port can be directly changed in the text field**(1)**. The “Session Timout” **(2)** and the “Allowed Games” **(3)** offer Edit buttons to open the respective “Edit View”. Furthermore there is an option to restore the default settings on button press **(4)**. Lastly, for any change to be made persistent a “Save” button **(5)** needs to be pressed. Editing the session timeout can be done by choosing the game **(6)** from the select box and adjust the session timeout in the text field **(7)**. The edit view for The Allowed Games offers the possibility to add a new game **(8)** to VISAB or to remove a selected game **(9)** from VISAB. With the “Return” button the “Edit View” will be closed and the made changes need to be saved in the “Settings View”.

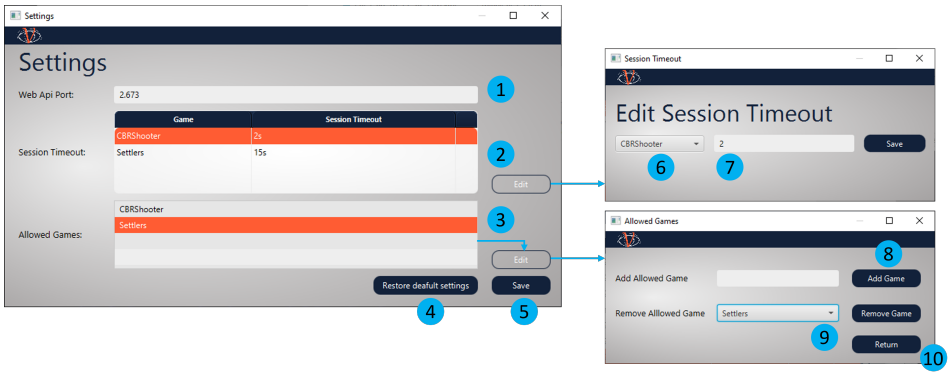

Figure 2.16: Settings View

Help

The "Help View" shall help anyone using VISAB by providing important information in utilizable formats. By clicking on the specific tab **(1)** one can either have a look into the Video tutorial or into the PDF documentation

Video Tutorial The "Video Tutorial" tab contains a video (`https://www.youtube.com/watch?v=znG2ZtcAfqA`) that is directly embedded from YouTube and can be either viewed in the application itself or in the web browser if you click on "Ansehen auf YouTube / View on YouTube" **(2)**. The video explains how VISAB works and what it is used for.

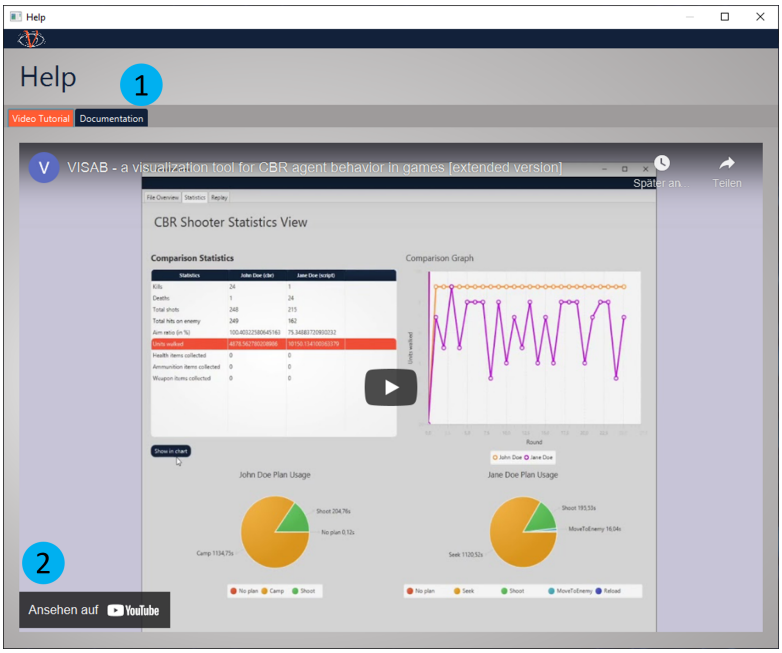

Figure 2.17: Help View - Video

PDF Documentation The "Documentation" tab contains a PDF viewer **(1)** with all basic features that you would expect. A user can simply scroll through the document and even search for specific key words with ctrl+f.

Figure 2.18: Help View - Documentation

About

The "About View" contains a tab **(1)** for every VISAB version. In the VISAB 1.0 tab all developers with names and mail addresses **(2)** are mentioned. In the VISAB 2.0 tab the developers are also mentioned with their names and mail addresses **(2)**. Furthermore a rough overview of all new features, compared to VISAB 1.0, are displayed in a list **(3)** in the VISAB 2.0 tab.

The tab structure offers a comprehensive way of illustrating VISABs current and also future progress throughout the different versions.

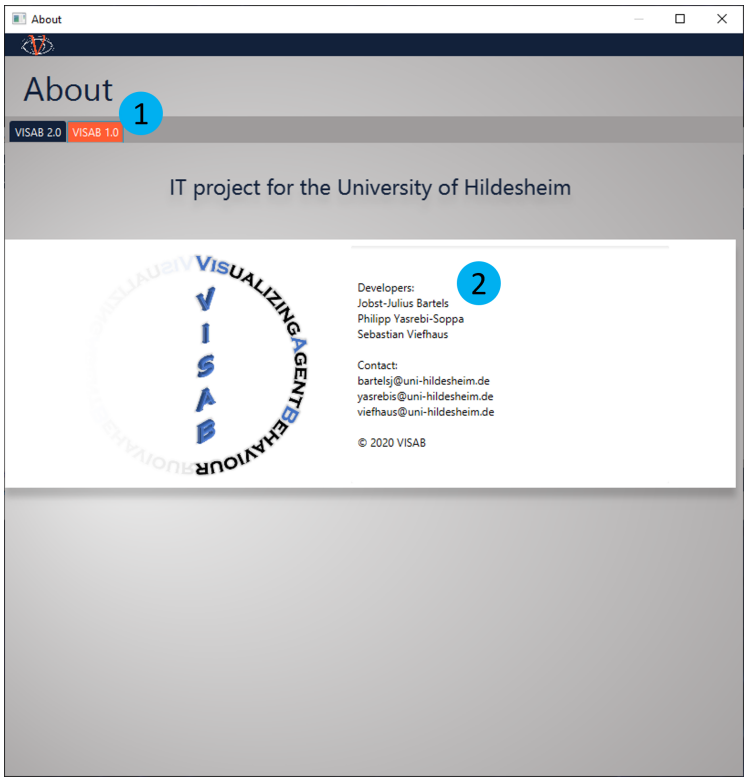

Figure 2.19: About View - VISAB 1.0

Figure 2.20: About View - VISAB 2.0

d) Features

(i) Visualization of Agent Behavior

The core competence of VISAB is to illustrate agent behaviour in games. Consequently VISAB needs to have fitting views implemented that provide useful information extracted from the game data and show them in a clear by allowing multiple different view points.

This page describes the underlying visualizer concepts and gives hints about the current set of implemented visualizers in VISAB. If you want to have a deeper look on how to control the UI from user perspective, please have a look at the GUI Usage Article at `https://visab-org.github.io/getting_started/usage.html`.

VISAB is designed in a fashion, where any type of game may have their own set of visualizers (each visualizer adresses a different view point) that may be useful to illustrate certain aspects of the game. However, VISAB in its current state, implements three different types of visualizers for both **Settlers of Catan** and the **CBRShooter** which can be seen as a recommended set that should fit for any game you might want to integrate.

Due to the fact that each visualizing view is very custom regarding the respective game, there are no code samples provided. If you want to learn about the practical implementations, please have a direct look into the GitHub Repository `https://github.com/VISAB-ORG/VISAB`.

Please note that when implementing Views, the MVVM pattern has to be followed, so that any game will have at least two classes that realize the respective view, e.g.:

- AnyGameReplay**View**: Is only responsible for how the UI elements are displayed in the UI
- AnyGameReplay**ViewModel**: Is responsible for data access across the underlying model and provides it for the view class

Furthermore, the related .fxml has to have the exact same name as the as the view, e.g.: AnyGameReplayView.fxml and also must be located under the exact same package pathing within src/main/resources to be automatically locatable by the FluentViewLoader of JavaFX/mvvmfx.

Meta (File) View The meta view is used to provide static information of the VISAB file which are valid by the end of it. For example it may contain general information about items in the game, the winner of the whole session, etc.

This view is very basic and features no interactiveness and should be rather simple to implement.

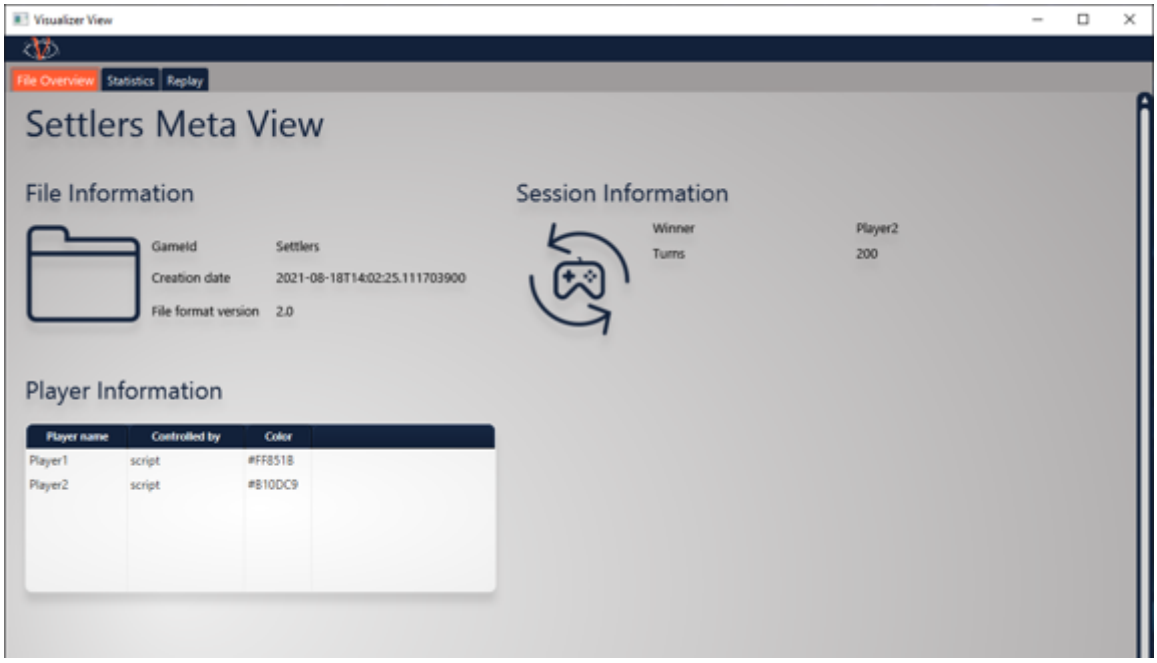

Figure 2.21: Meta View Example

Statistics View The statistics view aims at providing cumulated information for each player that is well comparable. For example, it is highlighting the plan usage for each player very in separate charts.

This view has some interactive options such as showing the comparison between some players for a specific measure. Here you need to be careful about how to fill the respective graphs with data, because JavaFX sometimes shows strange behaviour if the refilling of graph data is not done smoothly.

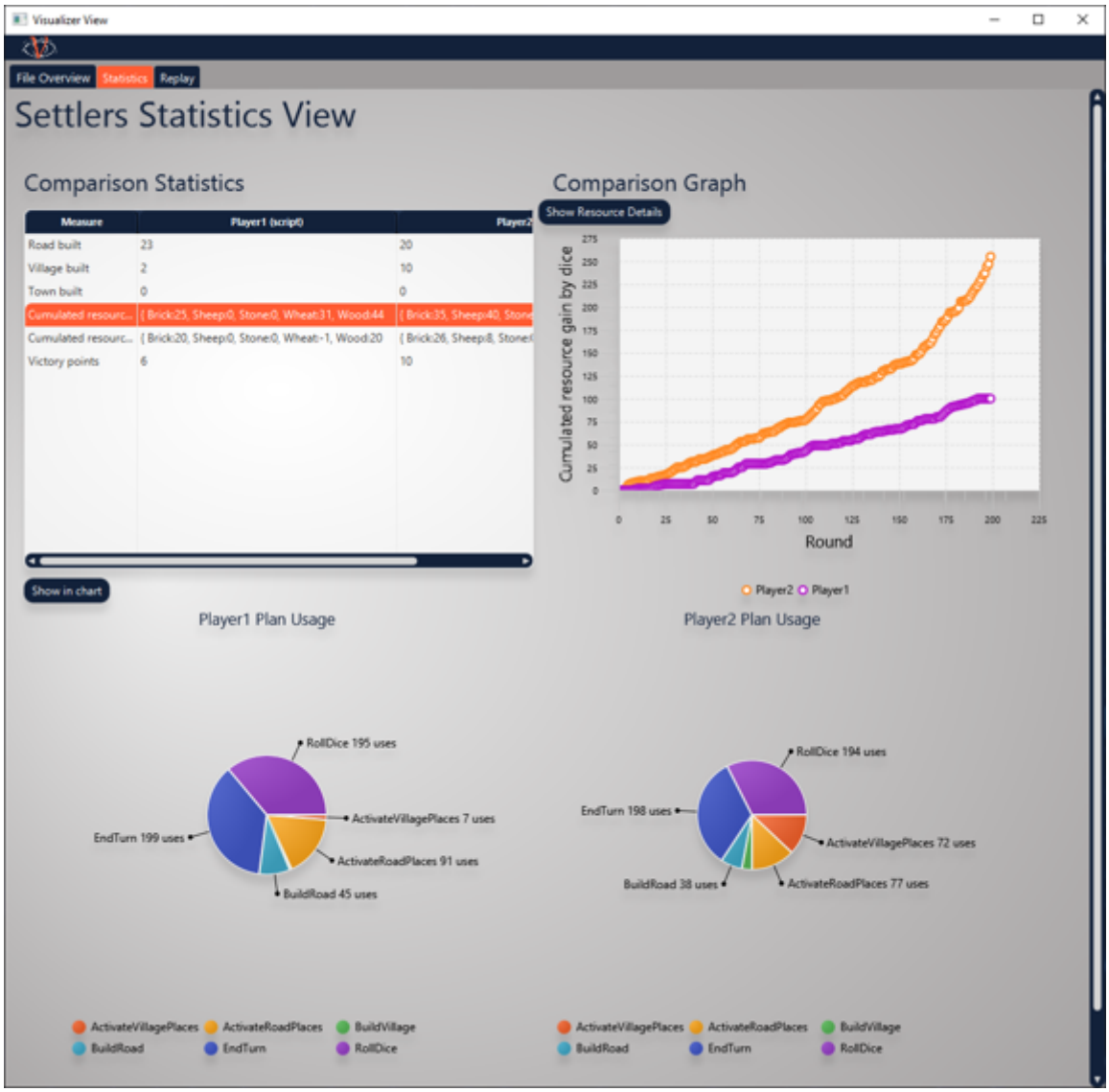

Figure 2.22: Statistics View Example

Replay View The replay view fills another gap, the statistics view still leaves open. It may be interesting to replay a match or any specific moment of it as often as you like to get a better insight of what circumstances in the game may have lead to specific decisions of the AI bots competing against each other.

The replay view is the most complex view at the current state of VISAB. Due to this fact, there are some things to watch out for while implementing:

- The underlying map image is correctly scaled in terms of width and height.
- The game objects need to be positioned correctly by translating their game coordinates to the JavaFX GUI boundaries of the map image.

- Be careful on how and when to set the game objects to the scene graph in JavaFX due to performance issues.

  For example completely replacing all of them at each turn might take too long and it will cause the slider to behave strange.

- Due to its capability of going **forward** and **backward** in the game data, visualization of all map objects has to be handled in a reasonable way.

  For example moving the turn slider back from 60 to 40 should correctly hide objects that were not there in turn 40 of course.

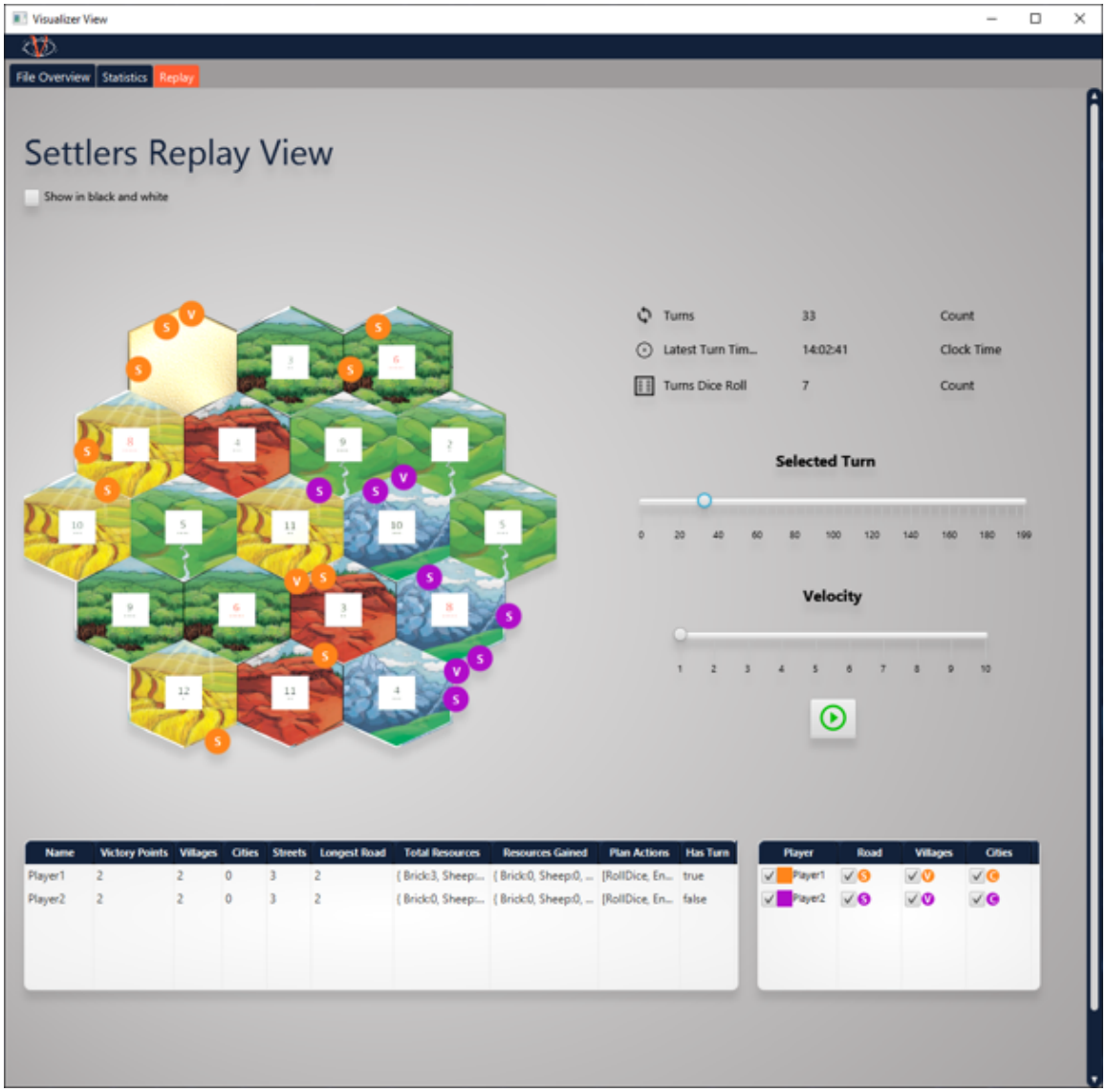

Figure 2.23: Replay View Example

(ii) HTTP API as Communication Interface

The new version of VISAB provides a generalized communication interface in form of a HTTP based Web API. This API acts as a server, to which games communicate their data. To achieve scalability, incoming data is delegated in an event-based manner to the processing modules. This HTTP API is theoretically capable of receiving requests from K different machines each running m_{kn} instances of one of the N games supported by VISAB. For each of these

$$\sum_{k=1}^{K}\sum_{n=1}^{N} m_{kn}$$

many visualizer views may be opened concurrently.

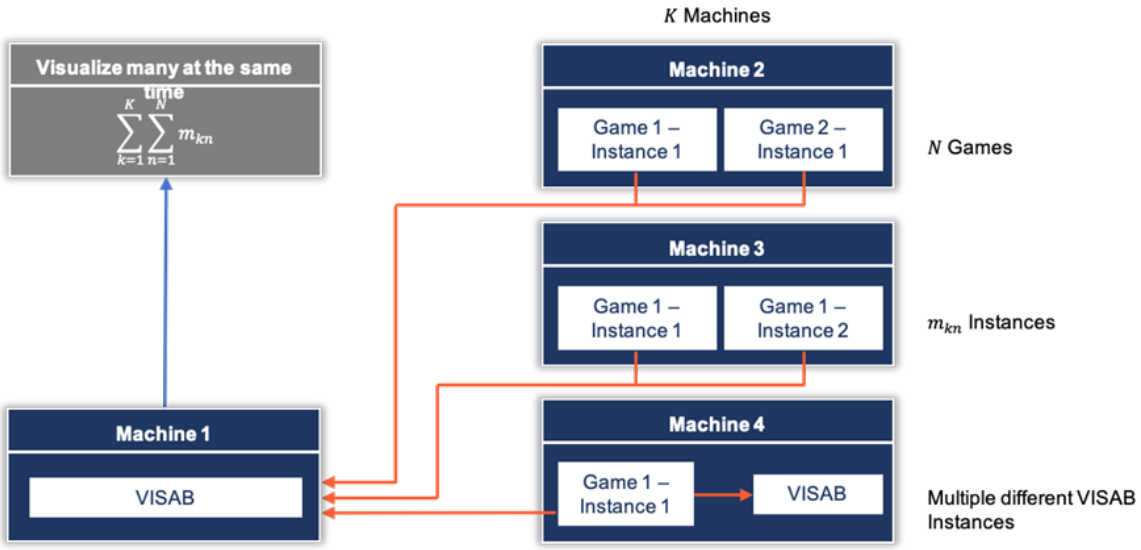

Figure 2.24: Scalability HTTP API

(iii) Extendability

The underlying code base of VISAB is designed to be easily extensible. This article will guide you through all relevant changes you need to perform, if you desire to make VISAB capable of supporting a new game. As the below order already indicates, we advise you to start with the modifications in your game, because then you will have a fixed basis, of what shall be processed within VISAB.

Step by Step Guidance

1. Setup your environment

2. Deploy the VISABConnector in your (Unity-) game
3. Create necessary POCOs in the game
4. Implement a VISABHelper in your (Unity-) game to extract data and send it with the VISABConnector
5. [**OPTIONAL**] Use the VISABConnector.Unity to perform snapshots
6. Create a new SessionListener in VISAB
7. Create necessary POJOs in VISAB
8. Create a VISABFile in VISAB
9. Implement desired visualizers (at least one) for your (Unity-) game
10. Add respective classes to the classMapping.json

(iv) Other Features

- **Headless vs. GUI Mode** VISAB can be used in two modes.
  - **GUI Mode** This mode starts VISABs HTTP-API such that data can be received and opens a window using which game data files can be visualized.
  - **Headless Mode** This mode starts only VISABs HTTP-API.
- **Dark Mode** VISABs GUI can be displayed in both a light mode and a dark mode. This setting is also persistant over different runtimes.
- **File Explorer** VISAB Home View contains a file explorer control that supports all basic file operations. Additionally drag and drop and shortcuts such as **Commandkey + C**, **Commandkey + V**, and **Delete** are supported.
- **Logging** VISAB has a logging system using which important events are shown at runtime and stored in log files. The files additionally contain debug output.

3 Evaluation

a) Requirement Evaluation

VISABs features were tested across the whole development team and their respective machines on all three operating systems (Windows, Mac, Linux) that we officially declared as supported. Any requirement and feature that was formulated in the first place was either successfully implemented as intended or even surpassed its expectations by enlarge in VISABs current state. Furthermore we were able to improve the overall structure of VISAB to be more flexible when it comes to extending it - which can be considered as one of the most valuable capabilities of it. Throughout the project, we often even took care of improving already implemented features, such as for example the scalability for the CBRShooter, which priorly supported only two players but now is also flexible for even more ones.
In the following, we evaluate the requirements set for VISAB 2.0 in the "Konzeptpapier".

(i) Map auslesen

Requirement **A1** "Map auslesen" was: *The system must include reading and displaying of arbitrary maps from Unity based games.*
In order to ensure high flexibility and adaptability for VISAB, one of the main requirements was to enable a dynamic extraction of the map directly out of the game. Rather than the original static approach where a placeholder image was used for the CBR-Shooter game, this approach should support the relatively quick and easy extension of any arbitrary game. Two ideas emerged as viable: The first one was to use Unity's transform attributes of the game scene, which would be used to draw the map at VISAB's runtime fully dynamically. The second idea required the insertion of an additional camera within the unity game itself. This camera would make a snapshot of the actual scene/map at the game's runtime and send it to VISAB at the start of the according game. For reasons of practicality we decided to implement the latter approach by creating a camera at the start of the game that would be moved to any game object in order to be able to snapshot it. The object's bounds (size of game objects in unity) were taking into account while snapshotting them. Moreover, with the possibility of instantiating any game object by

passing their prefab path to the camera, we encountered the problem that some objects that also needed to be extracted at the beginning of the session would be spawned at a random time.

(ii) Generalisierung Schnittstelle

Requirement **A2** "Generalisierung Schnittstelle" was: *The system must include a generalized interface to the Unity Engine. The interface must only provide communication to VISAB and not from VISAB.*
Our initial idea was to realize this either through "Sniffing" the TCP/IP communication of the game with the CBR-system or to create a HTTP API inside VISAB. We discarded "Sniffing" early on, as can be read in **TCP/IP Sniffing** and went with the HTTP API. Our scetch of the communication included the Broker design pattern, which in the end we didn't have to implement ourselves, as the library we used already provided a full (tiny) HTTP Server implementation. The idea of informing the data processing modules of new incoming data via the Publish-Subscribe pattern was implemented as expected. In hindsight, the overhead HTTP has over more lightweight protocols can be problematic with a high request frequency ($>$ 100 per second). The main advantage we saw in using HTTP, was that we would have no trouble finding an existing framework for it. The features HTTP provides such as authentication are not necesarry for our use case at all. With regards to performance, we have not encountered issues yet, but it is not entirely unreasonable to assume that it may become an issue with games wanting to send data at a very high frequency.

(iii) Integration Siedler & GUI Siedler

Requirements **A3** "Integration Siedler" and **A5** "GUI Siedler" were: *The system must include extracting game data from the Unity game Settlers of Catan* and *The system must provide the user with a visualization of the game data from Settlers of Catan. The data must be visualized as a recurrence view and as a generic statistical view.*
The integration of Settlers of Catan as a new supported game in VISAB directly ties with the generalization of communication interface as well as our overall GUI redesign. Settlers of Catan and the CBRShooter both use the VisabConnector to expose their game information to VISAB. The tact rate of communication is different due to the games natures.

While the CBRShooter communicates loop-based, Settlers of Catan only sends the data turn-wise. Settlers of Catan has the same view set as the CBRShooter (MetaView, StatisticsView, ReplayView) whereas the last two were already contained in the first Version of VISAB, but received massive improvements for both games during this project. For Settlers, we implemented an additional view for resource details, which enables the user to dive even deeper gathered and spent resources for each player in specific intervals of the game session.
During development we encountered some difficulties with JavaFX that were not completely eradicatable. Firstly, we observed that the comparison graph in the statistics view has a chance of being broken if new data is selected too quickly, even though everything is cleaned up appropiately. The second issue was, that the color assignment for the StackedBarChart in the resource detail view is picked randomly again, if the graph gets filled with new data (on slider move), which is why there is no static color mapping for the resources displayed in the graph.

(iv) Konvertierung Spieldaten Siedler

Requirement **A4** "Konvertierung Spieldaten Siedler" was: *The system must include the conversion of game data to the VISAB or an alternatively chosen generic format.*
We chose JSON to be the designated data format for both the communication between VISAB and the games as well as the stored visab files as we decided to avoid an unnecessary custom format as it was used in the first version of VISAB. JSON is very well human readable format and perfectly fits into the underlying toolset and games. This way, any (de-)serialization process depends on the same data format across the whole VISAB toolset.

(v) TCP/IP Sniffing

Requirement **A6** "TCP/IP Sniffing" was: *The system is to include "sniffing" of the data exchange between the CBR system and the game.*
Due to the fact, that the games communicate over TCP / IP with their agents to send data back and forth in JSON format, we thought about an approach that would enable VISAB to directly sniff that communication and being completely independant from the game itself. Unfortunately this was not really possible, because VISAB requires more

detailed information to properly display game data as there is sent between the AI agent and the Unity games. Additionally, use cases like extracting map images and other visuals from the game finally invalidated this approach for us, which is why we decided to go for the highly scalable HTTP API as it serves VISABs needs perfectly well and even provides more flexibility when used in tutorials at the university.

4 Conclusion and Outlook

AI is a striving expertise field, that steadily gains relevance. In order to help young developers to get their hands on this topic, an AI learning platform as it is intended to be built up by IIS, University of Hildesheim is highly beneficial. However, understanding, developing and debugging agents can be a very hard task and that is where VISAB comes into play. By enabling any game (currently Unity games are supported) to send as much data as necessary to VISAB via the HTTP interface, a huge potential of visualizing all sorts of things is given. Even the educational aspect is considered within VISAB. Due to the deployment architecture, VISAB can be accessed from different machines or even from other networks with tools like hamachi. For example, it would be possible for the lecturer to have VISAB running on this or her machine and actually receive game data from different students and maybe different kinds of agent implementations.
That being said, VISAB has a high potential of being a fundamental pillar to make the AI learning platform as successful and convenient as it can be.
We, as the development group, made use of collaboration tools like Microsoft OneNote and maintained everything of our code within a centralized GitHub organization, which we would recommend anytime. Especially in times, where meeting face-to-face becomes rarer, an efficient project management with regular meetings and QA sessions with the lecturer, supported by a variety of tools for work package organization and code maintenance appeared to be indispensable.

In the future, a helpful feature would be automatic code generation for initializing the integration of new games. Since the scheme for the game implementation is always similar, this should be feasible. Furthermore, a visualization of the casebase which is accessed by the cbr agents would be useful. Here one could go so far as to allow queries on the side of visab, so that for a given game situation the most similar case can be obtained. This could further simplify the debugging of the agent behavior. Another, not so exciting, point would be the implementation of an interface to VISAB's HTTP API in other languages and the development of libraries for extracting images and other desirable game data. The current more or less directly translated VISAB-JConnector is such an implementation besides the functional VISABConnector.