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

Figure 2.23: Statistics View Example

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.

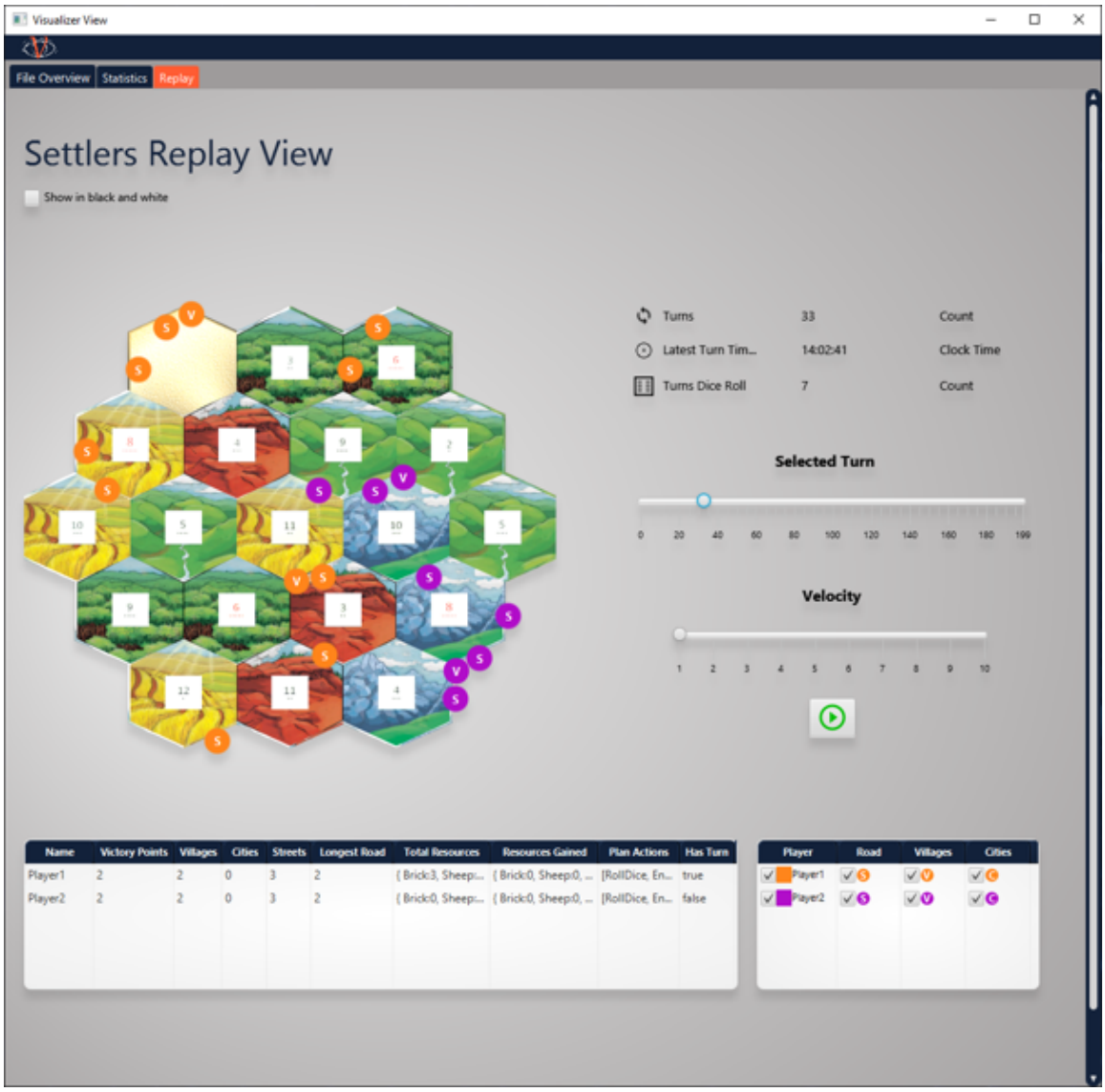

Figure 2.24: Replay View Example

(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.

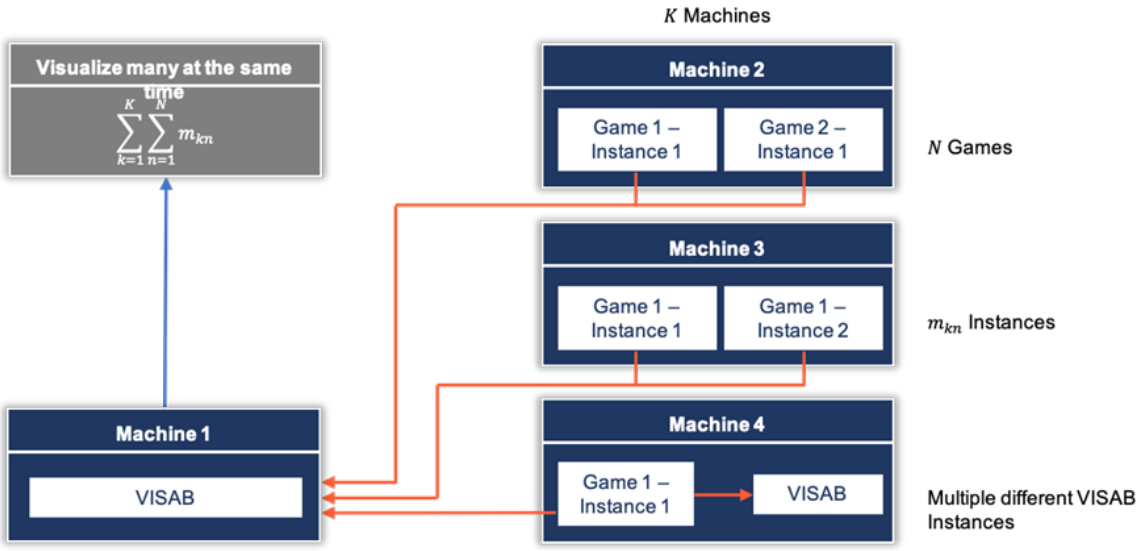

Figure 2.25: Scalability HTTP API

(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.