Developing a Configuration System for a Simulation Game in the Domain of Urban CO_2 Emissions Reduction

Sarah Zurmühle, João S.V. Gonçalves, Patrick Wäger, Andreas Gerber, and Lorenz M. Hilty

Abstract In order to help decision-makers find ways to reduce CO_2 emissions of Swiss cities, a simulation game is being developed within the "Post-fossil cities" project. During the game, participants take on different roles in which they together explore pathways to a future, post-fossil city. An important requirement to the software system of the game was to be easily configurable in order to keep the game adaptive to different target groups of players. We describe a User Interface Management System (UIMS) that has been designed and implemented to realise the flexibility demanded from the game designers' side. The system allows game facilitators to configure the game and decide what kinds of visualisations are used during game sessions. The paper describes how the configuration system was conceptualised, implemented and integrated into the overall system architecture of the simulation game.

Keywords Simulation game · User interface management system · Configuration system · Greenhouse-gas emissions · Sustainable development

The original version of this chapter was revised: This chapter has been changed to open access under the terms of the Creative Commons Attribution 4.0 International License. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-61969-5_19

S. Zurmühle (✉) · J. S.V. Gonçalves · L. M. Hilty
University of Zurich, Zurich, Switzerland
e-mail: sarah.zurmuehle2@uzh.ch

J. S.V. Gonçalves
e-mail: goncalves@ifi.uzh.ch

J. S.V. Gonçalves · P. Wäger · A. Gerber · L. M. Hilty
Empa Swiss Federal Laboratories for Materials Science and Technology, Dübendorf, Switzerland

A. Kamilaris et al. (eds.), *Advances and New Trends in Environmental Informatics*, Progress in IS, https://doi.org/10.1007/978-3-030-61969-5_12

1 Introduction

The climate crisis is a major issue of this century. The domain of urban development offers the potential for substantial reductions in CO_2 emissions and can therefore contribute to reducing global warming. To support decision-makers, a simulation game is being developed within the "Post-fossil cities" project,[1] focusing on sustainable urban development in Switzerland. The players are given the opportunity to simulate the development of a fictional Swiss city and thereby explore different pathways to a post-fossil future. The project, which belongs to the Swiss National Research Program "Sustainable Economy", is positioned in the context of the Paris Agreement [16] and the UN Agenda 2030 for Sustainable Development [15]. In order to achieve the goals of reaching a post-fossil future and of complying with the criteria of a good life, the players interact and cooperate in different roles.

Besides the role-play part, the simulation game includes a physical game board equipped with sensors, a gameplay system that contains a set of simulation models, a simulation system controlling them and a Graphical User Interface (GUI) for the gameplay system. During the game, the players use the game board to manifest decisions, which are registered and evaluated by the gameplay system. Relevant information is visualised to the players on the GUI. These visualisations are composed of different visual components. The simulation game presents visual components on a screen to inform the players about the game status. Different players will find different visualisation types more or less effective. Furthermore, different visualisations can get different messages across. Thus, additionally, a second GUI is needed for the people who configure the software system before a game starts, i.e., the game facilitators. Through this GUI, they can adapt the software to the requirements of the actual game setup, which may vary in the number and type of participants, learning goals, etc.

The applied configurations must be communicated to the gameplay system, which then shows the selected visual components during the following game sessions. The simulation game uses simulation models of different types. The models take parameters, external variables and initial values for internal variables as input and produce time series as output, which can be used in the visual components. Thus, it was required to define a dynamic extraction mechanism for the available visual components in order to flexibly use them in the simulation game system. If the design of visual components changes, the simulation game should be able to easily adapt to those new changes.

Therefore, in addition to the mentioned parts of the gameplay system, a configuration system is needed. This system receives the input of the game facilitators, processes it and sends it to the gameplay system. This configuration system should provide a visual interface that enables the game facilitators to decide which visual components and decision cards (playcards used during the game, sensed by the gameboard) can be used during the game session. It should also define dynamic data extraction mechanisms used to communicate with the simulation system in order to

[1] http://www.nfp73.ch/en/projects/cities-mobility/post-fossil-cities.

obtain the needed model output data. Furthermore, a storage for visual components should be defined and used by the configuration system in order to dynamically load the needed visual components into the system. The flexibility and configurability of the gameplay system's GUI should not be unnecessarily restricted by the interface of the configuration system.

To meet these requirements, we decided to design a User Interface Management System (UIMS). UIMS separate the visual GUI components from the application's logic, which not only reduces a systems complexity, but also allows these two components to be modified separately from each other [12]. Therefore, it is easier to design the application's logic without having to think about the GUI design. A system with a clear structure furthermore facilitates the construction of a robust and adaptable system which is flexible to changes. This paper presents a UIMS in the form of a configuration system for the game developed in the Post-fossil cities project. In order to meet the requirements mentioned above, the UIMS must provide flexible and dynamic interfaces to all connected system parts. Those interfaces should allow the GUI of the game players to remain configurable[2] and flexible[3] with regard to future project requirement changes. This leads to the following research questions:

RQ1: What is a possible structure of a configuration system that allows the gameplay GUI to remain configurable and flexible to simulation game requirement changes?
RQ2: What is a feasible approach to link data streams of an exchangeable backend system to interchangeable visual components, without generating code dependencies in the simulation game software system?

The remaining part of this paper is structured as follows: Sect. 2 presents background information about simulation games and the Post-fossil cities project. Section 3 introduces and discusses related work. Section 4 elaborates on the approaches we used to solve the problem. In Sect. 5, the results are discussed. Section 6 draws a conclusion and presents some suggestions for future work.

2 Background

This section provides some background about the simulation game developed in the Post-fossil cities project and the concept of UIMS.

The Post-Fossil Cities Project's Simulation Game The goal of the "Post-fossil cities" simulation game is to allow stakeholders involved in the development of urban systems to explore possible pathways towards the post-fossil Swiss City in a playful-but-serious manner, while trying to stay within the remaining carbon budget

[2] The term "configurable" applies to the gameplay system. It means that the gameplay system should not be restricted in changing its gameplay mechanics in the future.

[3] The term "flexible" addresses requirements to the gameplay system. It means that the gameplay system should not be restricted in adapting to evolving project requirements.

and at the same time complying with criteria of a good life. The target audience of the game are decision-makers who are committed to a climate-neutral future and students as future decision-makers. The game is developed in an interdisciplinary consortium of researchers and designers from the Technology and Society Laboratory of Empa,[4] the Informatics and Sustainability Research Group at University of Zurich,[5] the Department of Energy and Process Engineering of the Norwegian University of Science and Technology,[6] UCS Ulrich Creative Simulations GmbH[7] and the Institute for Building and Environment of the University of Applied Sciences Rapperswil.[8] The simulation game provides the players with the opportunity to simulate the future development of a fictional city that represents Switzerland. Through interaction, the players try to reach goals related to the UN Agenda 2030 for Sustainable Development [15] and the specifications of the Paris Agreement [16]. By playing the game, participants with different backgrounds get the opportunity to take on new perspectives and to learn about sustainable development in the context of Switzerland. The simulation game allows the players to experiment with different strategies in a "safe" environment. During the game, players take decisions that are evaluated with simulation models, such as a dynamic stock-and-flow model of the Swiss societal metabolism that accounts for the most relevant materials and energy forms. The players get immediate feedback on the impact of their decisions, which allows them to assess their decisions and possibly learn from "mistakes" [18].

Major areas where players can take decisions during the simulation game include the building, transport and energy sectors. The building sector, for example, includes technology-related decisions such as "more frequent renovations with higher standards", "replacement of existing buildings with climate-friendly buildings", "adaptation of heating systems" or "construction of buildings with carbon capture and storage", but also includes lifestyle-related decisions such as "reduction of living space". In a similar way, the transport sector includes decisions focusing on the electrification of mobility and the decisions in the energy sector concentrate on different sources of energy. The thereby created decision space allows players to understand the impact of measures on greenhouse gas emissions as well as the relevant delays involved and thus to understand the importance of timing and sequencing of measures.

The simulation game consists—inter alia—of a physical game board and a software system that contains a heterogeneous set of simulation models. The software system is connected with the game board to be able to read the game status provided by sensors in the game board. The simulation models are used to simulate into the future based on the general development and on the decisions taken by the players. In order to display the results and for other game-relevant visualisations, the system

[4] https://www.empa.ch/tsl.

[5] https://www.ifi.uzh.ch/isr.

[6] https://www.ntnu.edu/.

[7] http://www.ucs.ch/.

[8] http://www.ibu.hsr.ch/.

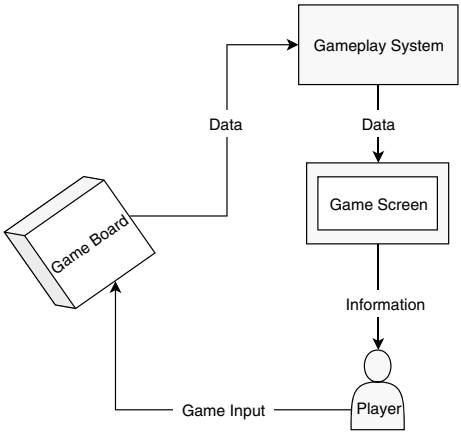


Fig. 1 Interaction between player, game board, gameplay system and game screen GUI

has a GUI. The players interact with the board and the GUI while playing the game. This structure is shown in Fig. 1.

User Interface Management System (UIMS) It is typical for larger software projects that many components have to interact with each other. For instance, if a GUI exists for an application, the input given by the users to the GUI has to be collected, processed and actions in response to the user's input have to be computed. One possible approach to design such a system is to combine the GUI with the mechanism responsible for user input and the application logic methods. However, systems integrating application mechanism and logic with GUI or User Interface (UI) elements have a major drawback: because representation and implementation are tightly coupled, the system is not flexible with regard to later changes. For instance, graphical elements have to be implemented into the application's source-code which increases code complexity—which in turn results in more bugs or more difficult software testing. Thus, having all the different components integrated into the application code makes it hard to maintain the system flexible while in use of GUI applications [12].

To tackle this problem UIMS were developed. The aim of a UIMS is to separate the logical system from the GUI code in a computer program and to maintain communication between them [12]. UIMS allow fast design prototype creation (even before application code is written), enable interfaces to be flexible to changes during development, allow people with different roles in a development team to work together and reduce development time [11]. Hill [5] stated that the usefulness of UIMS is no longer questionable. However, there are needed features that allow UIMS to unleash their full potential. Hill [5] proposed a list of desirable UIMS features that focus on the UI part: Allow the usage of multiple input devices simultaneously, enhance communication between UI modules at run-time and allow the UI to be halted during execution to implement changes. These features may enhance the quality of the UI that the UIMS is managing. While designing a new UIMS, these features can be used as a guideline. Another example of a UIMS guideline can be found elsewhere [10].

3 Related Work

This section presents relevant related work in both UIMS and simulation game research fields.

UIMS There are many implementations of UIMS with several different purposes. Generally, they can be categorised into research-based and commercial products [6]. ProcSee is an example for a commercial product and focuses on the implementation of dynamic GUI [7]. On the other hand, Serpent is one example of a research-based UIMS product. It was developed at the Carnegie Mellon University and enhances the incremental development of UI throughout the whole project phase. It provides a layout editor which can be used for interactive prototyping and a dynamic specification language used for production and also maintenance. Because the architecture is designed in a general way, new interface features could be added even during the development cycle [14].

This paper focuses on the construction of a UIMS in the context of a simulation game and one of the project goals was the construction of a system that lets the GUI of the players be flexible and configurable. Therefore, a general architecture is needed. However, the use-case of this paper is simulation games and not UI development. Therefore, the Serpent's system could not be used for the configuration system UIMS because simulation games depend on more components than the development of UI.

After 1990, more enhanced UIMS were built. One of these UIMS was Alpha [8] which is based on the ideas of the UIMS Serpent. It was designed to make a fast implementation possible which was furthermore clean and powerful. Additionally, Alpha's developers tried to correct the deficiencies of Serpent and its successors [8]. In the research of Shaer [13], the author worked on building dynamically adapted reality-based interfaces by combining a User Interface Description Language (UIDL) and a UIMS to make the system flexible to all kinds of input and output-devices. The UIDL was used to describe and implement the interfaces and keeping the system open to various devices [13]. These kind of researches addressed the issue that modern systems have to be adaptable to all kinds of input and output-devices. In this paper, the configuration system needs to be kept as flexible as possible. Therefore, making it adaptable to multiple different devices is required. However, because all the systems above do not feature simulation games, they could not be used for this paper.

Simulation Games Simulation games are applied in different settings and for different purposes. A number of such games have been applied in the context of sustainable development, amongst others to address environmental issues. For example, Van Pelt et al. [17] investigated the role of simulation games in the context of the communication of climate change uncertainties. They created a simulation game called "SustainableDelta" which was used in a workshop with students and water managers. Like in the "Post-fossil cities" simulation game, the players got the chance to discuss decisions and related consequences [17]. Another example for a simulation game is Septris [4], an online mobile simulation game in the context of health that features the detection and treatment of sepsis.

4 Developing the Configuration System

The infrastructure of the "Post-fossil cities" simulation game, excluding the configuration system developed here, consists of an interactive game board, a gameplay GUI application displaying information about the current game state, models used to run simulation experiments (e.g. a stock-and-flow model to calculate stocks and flows of materials and energy in Switzerland) and an agent-based simulation system which handles all the tasks necessary to orchestrate the simulation.

The game board is a tangible object which lets the players physically interact with it by playing cards. It includes sensors that recognise the players' actions mediated by so called decision cards via sensors and process them directly or send them to the game's backend system for further computation. For example, a decision card could state that the player made an investment in solar energy. Playing this card triggers an update of the current state of the game, to which the players will react again.

The gameplay GUI on a screen is designed to present visualisations in order to inform all players about the game actions and states. Those visualisations are composed of visual components. They can be common charts such as line charts or more creative ones like a filling glass of water that represents a carbon budget. During the game, the visualisations are updated according to the played decision cards of the players. Thus, the players always see the impact of their actions. The gameplay GUI's coupled backend system manages which visual components and information are shown and how they behave on the display during the game. The gameplay backend gets its data from the simulation system, which selects the appropriate models to use. For instance, a model could calculate data about yearly CO_2 emissions. The output of those models are used as data source for the visual components shown on the gameplay GUI and the configuration system's GUI. The UIMS interacts with the gameplay's backend system and the simulation system.

The configuration system handles three interfaces: the configuration data transaction to the gameplay system, the model's data extraction from the simulation system and the definition of the visual component storage. The transaction of the configuration data is done by transferring a JavaScript Object Notation (JSON) file containing the data. The configuration system communicates with the simulation system via Application Programming Interface (API) calls. The visual components designed for the simulation game are stored in a git repository and are dynamically loaded into the configuration system.

The UIMS consists of a frontend GUI, a backend system, a database and an external data storage. Figure 2 shows an overview of the overall simulation game's structure. All components which belong to the configuration system are indicated with the colour orange. A git repository is used as a data storage for visual components. This part is coloured in purple. The blue components are part of the gameplay system and the green box represents the simulation system. The following sections will elaborate on the separate system parts of the configuration system. The source code of the configuration system can be found in a digital repository [19].

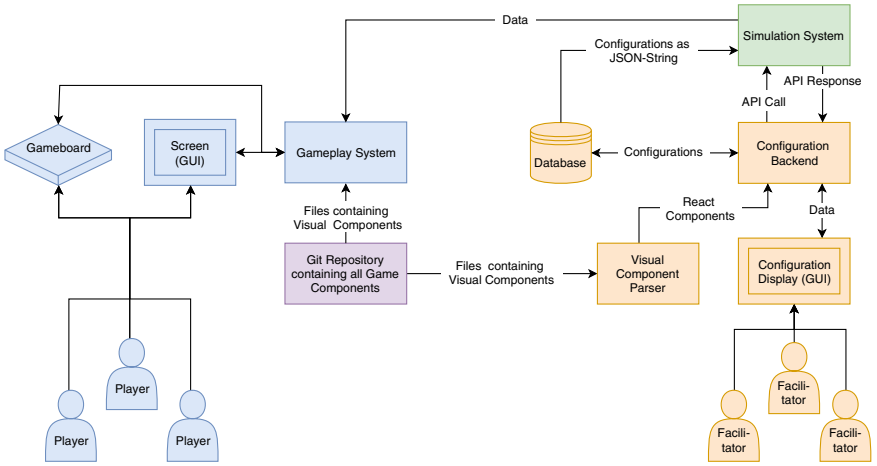


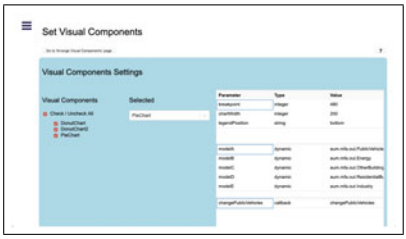
Fig. 2 Simulation game project structure

Frontend The configuration system is a web-application. A web-application is not device bound, which means that every device that has access to the web server of the simulation game has access to the configuration system's application. The frontend part consists of the GUI that the game facilitators interact with to set the configurations. Its most important requirement was to enable the facilitators to make all needed configurations and update the backend system about the made changes. The following were the UIMS frontend's requirements:

1. It must be possible to choose from which git repository to load the visual components from.
2. It should be possible for the facilitators to decide which of those visual components are then shown on the gameplay GUI screen.
3. Facilitators should be able to decide which decision cards can be used during the game session.
4. It should be possible to arrange the chosen visual components on screen.
5. The facilitators should be able to upload their configurations.

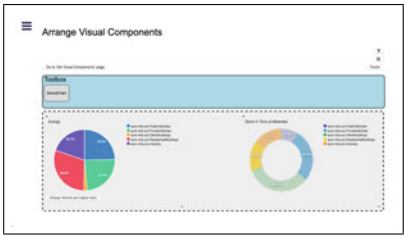
These requirements were collected by conducting interviews with the UIMS's stakeholders, i.e. the game's current facilitators. To keep the system as flexible as possible, facilitators must be able to configure from which git repository the visual components are imported from. The storage of the visual components is therefore not limited to one specific git repository. Thus, to fulfil the first requirement, a separate webpage was created which enables the user to enter a link to the git repository of the visual components. A screenshot of this webpage is shown in Fig. 3a. To fulfil the second and third requirement, a separate webpage was created which is split into two parts: the visual components settings, shown in Fig. 3b, and the decision cards settings, presented in Fig. 3c. All available visual components and decision cards are listed in a separate checkbox list from where they can be enabled or disabled. For each

(a) Settings webpage enabling the game facilitator to enter a git repository link



(b) Webpage enabling the game facilitator to decide which visual components should be used during the game session

(c) Webpage enabling the game facilitator to decide which decision cards should be used during the game session



(d) Webpage enabling the game facilitator to arrange visual components on screen - here one of three components is located in the toolbox

Fig. 3 Screenshots of the web-application of the configuration system

visual component or decision card their corresponding parameters can be modified in a separate box. Parameters can be static, e.g. the colour of the visual component, or dynamically linked, e.g. which model in the simulation system provides the data that the visual component needs.[9] Lastly, in order to fulfil the fourth and fifth requirement, a third webpage was designed which focuses on the arrangement of the chosen visual components. A screenshot of this webpage is shown in Fig. 3d. All previously chosen visual components are shown on the screen. If there are too many components at a time on the screen, individual components can be placed within a toolbox for temporary storage. The size of the components can be adjusted and if the game facilitator is satisfied with their arrangement, they can save the settings by pressing a finish button. By pressing on a preview button, the facilitator can see how the arrangement is going to look like on the gameplay system's interface. By using those webpages, the facilitators can modify all needed simulation game settings before each game session starts.

[9] Because the output of a model itself changes dynamically during a game session, these parameters are called dynamic.

Backend While the frontend of the configuration system is responsible for the GUI display, the backend manages all application logic and interface connections of the system. It is therefore responsible for managing the connection to the database (including inserting or updating and extracting information), receiving from and write data to the simulation system, defining API calls which are used to send to and receive information from the frontend system and loading the visual components from a git repository into the system.

The backend system follows the Model-View-Controller (MVC) design pattern [9]. The application logic, e.g. extracting data from a data storage, is performed in the *Model* part of the design pattern. The *Controller* part is responsible for preprocessing the data and defining API calls between the backend and frontend. The *View* is however implemented in the frontend and defines the GUI display.

Database The configurations made with the UIMS's GUI are saved to a database. A database can store data permanently in a separate place. It is also easy to extract required data if the database is properly managed. The configurations done with the configuration system can therefore be stored for a long period of time and can be easily accessed. A relational database was used, which is based on a relational model [2]. The database is only accessed by the backend system, which extracts and writes data into it.

Interface Between UIMS and Gameplay System To communicate the configurations made in the configuration system's web interface to other system parts in the simulation game, a JSON file is used. JSON files have a tree like structure, which allows the information to be easily extracted. It is also quite simple to add additional information, without changing the existing content [3]. Thus, using JSON format to store information makes it easier for other system parts of the simulation game to process the configuration data. Of course, the data can also be extracted from the database directly. However, to do so multiple SQL queries have to be written as the data is stored in multiple tables across the database. This approach would not be flexible to future changes in the database. Furthermore, information in a JSON file can be accessed with one call. Therefore, additionally storing the information in a JSON file to communicate the configuration data keeps the system simpler and more flexible.

Git Repository Data Storage On the configuration webpage, the game facilitators can view all available visual components and arrange them on screen, as to set how the final gameplay GUI should present them. It makes most sense to directly show the visual components as they are going to look like when they are used during the game session. Thus, the configuration system needs access to all visual components's source code in order to visualise them directly on the configuration GUI. Furthermore, the visual components are bound to change, which means that the configuration system should always have access to their updated version. Storing the visual components in an external repository, such as git, makes for a clean solution in the context of the configuration system. Git is a distributed version control system used to store source code and track its development [1]. The features of git are perfect

for the configuration system's use-case. Storing the visual components elsewhere and loading them on-demand into the system keeps the system flexible to changes. The designers of the visual components can upload them into a git repository. Whenever the visualisations change or are updated, the designers simply need to push their new version onto the repository. The configuration system on the other hand only needs the URL to the git repository that stores the visual components and, by cloning it, gets immediate access to the source code of all visual components. Their code can then furthermore be used in the UIMS's frontend system to visualise the visual components on screen.

Because hardcoding the link to the git repository into the configuration system is inflexible to changes, the facilitators can update the repository URL by using the system's GUI. Thus, the facilitators do not need to know about how the visual components are stored or updated. They only need to know that they need to provide the link to the right repository to the configuration system. After submitting the link, the frontend system transfers it to the backend system by using API calls. The backend then checks the URL, clones the corresponding git repository and updates the database with the new git repository link. Thus, when the git repository link was once submitted through the configuration GUI, it does not need to be re-entered again. The visual components from the entered git repository are then available for usage in the configuration system.

However, there is one major issue when using a git repository as a visual components storage: How can the configuration system detect if a file on a given git repository contains one or multiple visual components? It is not a flexible approach to assume that all files in the repository only contain one visual component. This assumption restricts the designers of the visual components too much because they would lose the possibility to write multiple classes or methods into one file. Therefore, a simple annotation language was created to detect visual components. These annotations are placed within the documentation strings (docstring) of given objects, such as classes, methods or functions. Furthermore, the annotations define which model data the visual components need. In order to define the model output data, static values or dynamic paths to the location of the datastreams can be used. To read the annotations, a parser was implemented which walks through all files in a given folder and checks if a file contains any visual components. It furthermore extracts all needed information stated in the annotations which is needed to include the visual components in the configuration system.

By using these annotations, all visual components can be integrated into the configuration system. Even when the location of the git repository changes, the process of extracting the visual components stays the same which makes the system flexible to newly defined or updated visual components.

UIMS and Game GUI Integration To integrate the configuration system into the simulation game architecture, the connection between the gameplay GUI and UIMS must be established. The gameplay GUI has to be extended in order to successfully combine those two system parts. The following additional components have to be created: a configuration data extractor, API calls between the gameplay system and

the simulation system and a visual components loader as well as parser. In order to get access to the configuration data of the UIMS, the gameplay system has to extract the configuration data from the configuration system's database. Because all configuration data is available as a JSON structure within the database, only this file has to be extracted using a single SQL call. JSON allows easy access to its stored data, therefore extracting the configuration data can be established by using existing methods. The gameplay system then knows which visual components it should visualise, what their size is and where on the screen they are located. The system also knows which data the visual components need in order to be visualised correctly. Static data can directly be used. However, if data paths are given, the system has to extract the data from the given path inside the simulation system. API calls are used to exchange this data. The extracted data has then to be passed to the visual components. In order to extract the source code of the visual components, the gameplay system has to use the same mechanism as in the configuration system to clone the given git repository. The link to the repository is also given in the configuration JSON. Because the visual components contain the annotation language described above, the parser of the configuration system can be used in order to check which files in the repository contain visual components. As a result, the gameplay system knows which classes or methods it has to import in order to visualise the visual components on screen. Because the dynamic data is extracted directly from the simulation system, the visual components data is always up to date. By completing those steps, the configuration system can be integrated with the gameplay GUI of the simulation game. Of course, this procedure is not limited to this specific use-case but can be used for various other simulation games with similar system components.

5 Discussion

Research Question 1 In order to answer the question "*What is a possible structure of a configuration system that allows the gameplay GUI to remain configurable and flexible to simulation game requirement changes?*", the configurations made in the configuration system have to be flexible to changes. The communication channel between the UIMS and the gameplay system consists of a single JSON file. The JSON syntax is in itself flexible to new content because it is structured like a tree. New content can simply be added without changing the whole structure [3]. Therefore, even when the gameplay system changes in structure and requires more information, the JSON output file can still be used because the new content can easily be added. If configuration requirements change in the future, the gameplay system is still able to change its architecture. The JSON output file of the UIMS remains flexible to newly defined requirements as well as to changes in the configuration process. Thus, the gameplay system remains flexible in its definition and structure. Therefore, the answer to RQ1 is to design a UIMS with the properties stated above.

Research Question 2 To answer the question "*What is a feasible approach to link data streams of an exchangeable backend system to interchangeable visual components, without generating code dependencies in the simulation game software system?*", designing a dynamic loading of visual components and their corresponding model data was essential. Given that it is possible that the gameplay system uses other types of visual components than the ones designed, a dynamic visual component extraction mechanism is important for the simulation game. Because the configuration UIMS allows to point to a location where the visual components are defined (in the form of a git repository link), the visual components can simply be swapped out by others. Furthermore, the annotation language enables the system to recognise visual components in a given collection of code files. This annotation language makes it possible to extract all needed information used to load the visual components in the system. Thus, visual components can be dynamically loaded into the system without having to include any additional code in the UIMS's source code. This procedure has the advantage that the gameplay system can use other visual components in the future and the configuration system does not need to be adapted. The game facilitators simply have to input a new link in the settings page of the web-application which loads the new components automatically. Additionally, the defined annotation language and its corresponding parser are not limited to the defined configuration UIMS, but can also be used in other simulation game system parts that use visual components, such as the gameplay system's GUI.

The data used in the visual components are extracted from the simulation system. However, this data processing system might also change. If the gameplay system needs other information for the visual components in the future, the configuration system's structure also does not need to be changed. The only thing that needs to be adapted is the data's location in the visual components annotations and the API between the configuration system and the simulation system. Therefore, the extraction of the model data is not limited to one source which makes the system flexible to changes and provides a dynamic structure. Thus, the answer to RQ2 is to design a UIMS with the interface properties stated above.

6 Conclusion

We showed the design and implementation of a UIMS in the specific form of a configuration system for the simulation game of the Post-fossil cities project. The UIMS enables the gameplay system of the simulation game to remain flexible and configurable even when project requirements evolve in the future. It allows the game facilitators to decide which visual components and decision cards will be used during the game session and to decide which dataset these components will use. The UIMS consists of a web-application with a frontend, backend and database part. To communicate the configurations of the configuration system to the gameplay system, the JSON format is used which has a very flexible structure that is easily extendable. To dynamically load the visual components into the system in order to let the facil-

itators modify them, the UIMS contains a git loader and parser that clones a given git repository, which contains the visual components, and extracts the corresponding files by using an annotation language. This structure allows the visual components to be independently designed as they do not need to be added directly as source-code into the configuration system. The data used for the visual components is stored externally in the simulation system. Overall, the configuration system lets the game facilitators define where the visual components are stored, allows them to define which components are used and let them specify the used model data. The final configuration is then stored into a database, where it can easily be extracted for further usage in the gameplay system. Due to the flexible structure of the UIMS, the gameplay system's flexibility and configurability is still provided when integrating the UIMS into the overall simulation game architecture of the Post-fossil cities project. However, the configuration system is not limited to be integrated with the presented simulation game, but could also be integrated with other kinds of simulation games. The flexible structure of the configuration system allows such a use-case extension.

In a future version it should be possible to store multiple configurations in the UIMS. Only one configuration can be edited and stored at a time so far. Furthermore, the configuration can only be stored in one language. For the future, it would be beneficial to enable the creation of different language versions of the same configuration. This feature would allow to use the settings made by the game facilitators for different groups of users who have similar interests but speak different languages.

To sum up, by using the configuration system in the simulation game, the game can be adapted to various types of target groups, enabling the game to unleash its fullest potential. By adapting how the simulation game represents information, the simulation game can bring across several messages to the players, making the game experience as effective as possible and helping players finding ways to reduce CO_2 emissions in future urban development.

Acknowledgements This work was supported by the Swiss National Science Foundation (SNSF) within the framework of the National Research Program "Sustainable Economy: resource-friendly, future-oriented, innovative" (NRP 73) Grant-N° 407340_172402/1. Aside from the authors, the following team members were involved: Markus Ulrich, Marta Roca Puigròs and Daniel Müller.

References

1. Chacon, S., Straub, B.: Pro Git, 2nd edn. Apress (2014)
2. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**(6), 377–387 (1970). https://doi.org/10.1145/362384.362685
3. Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON). https://tools.ietf.org/html/rfc4627 (2006). Last visited: 13 April 2020
4. Evans, K.H., Daines, W., Tsui, J., Strehlow, M., Maggio, P., Shieh, L.: Septris: a novel, mobile, online, simulation game that improves sepsis recognition and management. Acad. Med. **90**(2), 180 (2015)
5. Hill, R.D.: Some important features and issues in user interface management systems. SIGGRAPH Comput. Graph. **21**(2), 116–120 (1987). https://doi.org/10.1145/24919.24928

6. Iannella, R.: A graphical user interface reference model for messaging systems with directory integration. Ph.D. thesis, Bond University (1994)
7. IFE Institute for Energy Technology: Procsee graphical user management system technical overview. https://ife.no/wp-content/uploads/2018/12/ProcseeTechOverview.pdf (2018). Last visited 13 April 2020
8. Klein, D.: Developing applications with a uims. In: Proceedings of USENIX Applications Development Symposium, pp. 37–56 (1994)
9. Krasner, G.E., Pope, S.T., et al.: A description of the model-view-controller user interface paradigm in the smalltalk-80 system. J. Object Orient. Program. **1**(3), 26–49 (1988)
10. Lane, T.G.: A design space and design rules for user interface software architecture. Technical report , Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA (1990)
11. Myers, B.A.: Creating User Interfaces by Demonstration. Academic Press Professional Inc., San Diego (1988)
12. Olsen, D.: User Interface Management Systems: Models and Algorithms. Morgan Kaufmann Publishers Inc., San Francisco (1992)
13. Shaer, O.: A framework for building reality-based interfaces for wireless-grid applications. In: CHI '05 Extended Abstracts on Human Factors in Computing Systems (CHI EA'05), pp. 1128–1129. ACM, New York, NY, USA (2005). https://doi.org/10.1145/1056808.1056845
14. Software Engineering Institute: Serpent overview. Tech. Rep. CMU/SEI-89-UG-2, Carnegie Mellon University (1989)
15. UN General Assembly: Transforming our world: The 2030 agenda for sustainable development (2015). A/RES/70/1
16. UNFCCC: Adoption of the Paris agreement. In: United Nations Framework Convention on Climate Change. Report No. FCCC/CP/2015/L.9/Rev.1 (2015)
17. Van Pelt, S., Haasnoot, M., Arts, B., Ludwig, F., Swart, R., Biesbroek, R.: Communicating climate (change) uncertainties: simulation games as boundary objects. Environ. Sci. Policy **45**, 41–52 (2015)
18. Wäger, P.: Post-fossil cities. http://www.nfp73.ch/en/projects/cities-mobility/post-fossil-cities (2018). Last visited 13 April 2020
19. Zurmühle, S.: isr-ifi/pfc-uims: Pfc-uims v1.0.0 (2020). https://doi.org/10.5281/zenodo.3690735