Bachelor end project: Supporting large-scale escape rooms with a modular system

Bachelor thesis

Matthias Bakker
Ayrton Braam
Wouter Morssink
Tim Nederveen
Alexander Sterk

TU Delft

Bachelor end project: Supporting large-scale escape rooms with a modular system

Bachelor thesis

by

Matthias Bakker
Ayrton Braam
Wouter Morssink
Tim Nederveen
Alexander Sterk

Project duration:	April 23, 2019 – June 28, 2019	
Thesis committee:	Dr. H. Wang,	TU Delft, coordinator
	Ir. O.W. Visser,	TU Delft, coordinator
	Prof. dr. M.M. Specht,	TU Delft, coach
	J.W. Manenschijn BSc,	Raccoon Serious Games, client

June 28, 2019

Preface

This is the report for the Bachelor End Project (shortened to BEP) created by Matthias Bakker, Ayrton Braam, Wouter Morssink, Tim Nederveen en Alexander Sterk as a part of the Bachelor Computer Science program at Delft University of Technology. Over the course of ten weeks we have built a new system for creating and managing escape events from the ground up using TypeScript. This project was commissioned by Raccoon Serious Games.

Our deepest thanks to everyone at Raccoon Serious Games for their support during the project, and a special thank you to Jan-Willem Manenschijn, founder and CTO for taking the time to coordinate us. We deeply appreciate the trust they have given us when our system was used to run one of their contracted events.

Finally, we would like to thank our coach and coordinator Prof. Dr. Marcus Specht, professor for Digital Education at the Technical University of Delft and Director of the Leiden-Delft-Erasmus Center for Education and Learning for always finding time in his busy schedule to guide us and provide criticism where it was needed.

Matthias Bakker
Ayrton Braam
Wouter Morssink
Tim Nederveen
Alexander Sterk
Delft, June 2019

Executive Summary

Raccoon Serious Games hosts so called escape events. These events are similar to an escape room in which teams work together to solve puzzles, only on larger scale. Different kinds of challenges, arise when hosting these events, such as monitoring the progress of the teams and managing collection of physical puzzles. For an escape event, different kinds of staff help the teams to have the best experience possible. The teams need to be able to submit their answers to puzzles, while the staff should provide them with the right materials and monitor their progress. To facilitate these needs, Raccoon Serious Games created the Massive Online Reactive Serious Escape (M.O.R.S.E) system. However, the old system lacked in the parts of modularity and flexibility, while having the event hardcoded.

To solve this problem, we created a new system to improve the lesser points of the old system while also improving the usability and maintainability. Over the course of ten weeks, we researched and created the Massive Online Reactive Serious Escape 2.0 (M.O.R.S.E 2.0) system. The first part of the project consisted of researching the previous system to find the requirements for a new system. After that, a new system was designed and implemented from the ground up.

M.O.R.S.E 2.0 contains all the functionality of the old system, while also improving the structure of the administrator panel. Furthermore, the final product comes with several editors, allowing for designing the puzzles, schedule and logic in the form of rulesets for an escape event and allowing multiple events to be created managed. Furthermore the configuration editor allows the employees of Raccoon Serious Games to adjust different types of settings and cosmetics. The product was successfully tested during an escape event and will be used by Raccoon Serious Games for future events.

Contents

1 Introduction

In recent years, there has been a major growth in so-called “Escape Rooms” [1]. In these rooms, a group of people must work together to solve puzzles within a time limit. Companies have shown great interest in their own personalised “Escape Events”. The reason for this is that these events promote problem-solving, encourage teamwork and convey important messages to the employees. These events are similar to a regular escape room, but on a larger scale. So an event can span an entire building in which multiple teams, instead of one, roam around solving puzzles.

As companies usually have hundreds of employees, the fact that these events are larger scaled makes them very enticing for that very reason. Another incentive is that employees will participate enthusiastically [2], as it is more enjoyable for them to take part in an activity and learn through those experiences rather than through more traditional methods, such as a seminar. This not only boosts morale, but it also engages the employees more which causes them to extract greater value from the experience [3, 4].

For this reason, our client, Raccoon Serious Games, provides companies with their own Escape Events. They do so using web application, that allows teams to solve puzzles and their staff members to monitor the progress of the event. Unfortunately, this system is not modular and unmaintainable.

To this end, we have created a new system from scratch, which allows someone to create and host these “Escape Events” as our Bachelor End Project (BEP). The system has a great deal of features. It allows for the creation and managing of teams, it has tools for creating and customising different types of puzzles, and there are features for the rest of the crew, for keeping track of the game and statistics, and the ability to send out hints.

All features of their old system are also present in our new system, and our new system has even been used to host an actual Escape Event for a client of Raccoon Serious Games.

In this report, we will document our experience of the BEP. In chapter 2, we have defined and analysed the problem this BEP should address. The research of the original system, along with the requirements for this BEP’s challenge, follows in chapter 3, followed by the documentation of our initial design phase in chapter 4. Chapter 5 deals with the implementation of the new system, and the problems we faced during development. In chapter 6, an extensive explanation of how we tested our implementation and improved our code through static analysis is given. An overview of the final product is given in chapter 7, and our process throughout our BEP is documented in chapter 8. Finally, we conclude this report in chapters 9 and 10, which are our Conclusion and Recommendations.

2 Problem definition and analysis

This chapter goes into what the challenges are that the solution needs to address.

2.1 Challenges

During the execution of large scale escape-events, different challenges arise which would be a hassle to deal with without the use of technology. In this section, these challenges will be discussed.

2.1.1 Monitoring game progress

Companies want to host events for hundreds of people. Manually checking answers for each team would take a lot of time, reducing the pleasure of playing the game. Also, keeping track of which team is where in the game and which team solved which puzzle can be done efficiently with a system to facilitate this.

2.1.2 Logistics

To solve puzzles, participants will need physical attributes and puzzles to advance in the game. Since these attributes should only be available to teams who have access to these puzzles, a mechanism is needed to keep track of which team has or should get certain physical attributes.

2.1.3 Rapid change

During the development of an escape event, puzzle questions, texts and answers are subject to change. To accomplish a successful event, these changes should be reflected in the system as soon as possible.

2.2 Features

This section will go over what features are needed to address some of the challenges found above. These features are what drive the design of the solution.

2.2.1 The need for online events

The solution needs to be online or on a local network. An online system implies that players and crew members can connect to it remotely, to solve puzzles and monitor progress respectively. Logistically it is easier for tracking how players are progressing with the puzzles. It also eases assisting players, as they can fill in their answers online and receive immediate feedback, which means very little time is wasted, as compared to a more analogue solution.

2.2.2 The need for modularity

The solution needs to be modular. A higher degree of modularity makes it easier to create new types of events and allows the solution to grow and evolve. This makes the solution more flexible, making it easier to create complex events for clients that have very specific needs. It also makes for a better solution in general, as a high modularity allows for a relatively quick creation of new assets. It is possible to create an immersive escape experience in a relatively short amount of time.

2.2.3 The ease of managing

Another challenge is managing the hundreds of people taking part in the event. Their progress needs to be tracked and, if required, support must be given in a timely fashion. To this end, the solution needs to be able to organise people into groups, keep track of these groups and allow for a quick access to the event handlers. Another part of this is the ease of usability on the players part, if the solution is easy to use and has a clear and easy to understand interface, then the ease of managing is increased.

2.3 Conclusion

In conclusion, when creating large scale “Escape Events” for companies, several challenges need to be faced. These challenges are the monitoring of game progress in large scale games, the logistics of physical attributes during the game and the rapid change in the contents of events. Features that address these challenges are the need for online events, the need for modularity and the ease of managing.

3 Research

A big portion of the research for this BEP was reviewing the original M.O.R.S.E system. The first step of the research process consisted of investigating whether the previous system could be upgraded or that creating a new system from the ground up would be a better choice. The next step was evaluating the previous system to learn its strengths and weaknesses. The functionality of the original M.O.R.S.E system was foremost in creating a new system, since all the old functionality must be present in the new system.

3.1 Old system vs. starting from scratch

There are two options for solving the main problem of the client, which is the lack of modularity in the original system. This can be done by improving the old system with bug fixes, tests, documentation, re-factoring and functional improvements, or by rebuilding the system from scratch.

Improving the previous system implies having to learn the framework it used (i.e. Meteor), writing tests for existing methods, investigating the function of every method for in the documentation. Furthermore, the architecture needs to be altered in such a way that the system is modular enough to not be duplicated and deployed separately for every new event. A lot of time will be needed to understand the existing code, before changes or additions can be made.

On the other hand, rebuilding the entire system from scratch means investigating and choosing frameworks ourselves, re-implementing all of the existing functionality and subsequently adding additional functionality. Furthermore, the client would have to learn how to work with the new framework, if adjustments to the code are necessary after the project. However, since most requirements for a new system are already known before developing the system, it can be built with these requirements in mind. This was not the case for the old system, in which features were added on-the-fly. This means most of the system can be designed ahead of time, with especially modularity and testing in mind. With proper testing and documentation right from the start, it will also be easier for the client to maintain the system in the future.

Due to the extensive focus on maintainability and modularity for the client, which was lacking in the original system, it was decided to rebuild the system from scratch. It was not possible to add these requirements to the previous system, without an extensive overhaul of the code, at which point there might not even be that big of a difference between reusing or rebuilding for the amount of work required. Starting from scratch also makes properly testing the system feasible and possible.

3.2 Previous system

In this section, the old system will be analysed with regards to the front- and back-end.

3.2.1 Overview of the system

The original system consists of three main components, namely the client, the projector and the administrator panel. In the following paragraphs the relation between those three parts of the system will be explained. The game consists of a group of players and a staff team. The staff team currently consists of three tasks: admin, foreman and project desk.

The client provides functionality for the players. In the client, they can log in as their team, answer questions and track high scores. Players can open the client on their own device or on a device provided by the staff. Apart from the screen of their device, players could also get information by looking at the projector screen. On this screen several things can be shown, such as the remaining time or a video.

The administrator panel is used by staff members with three different roles: the administrator, project desk and foreman. The administrator controls the game flow, manages the teams and controls the content of the projector in the admin panel. The foreman manages his/her cluster, by tracking their progress and, if necessary, sending a hint. The project desk gives out puzzles and monitors this in the system, such that players only can solve puzzles they have collected.

3.2.2 Front-end

In this paragraph the front-end for the administrator panel and client will be described and analysed.

Admin

In the original system, the administrator panel is divided into six different parts. The administrator is concerned with two of them. One related to controlling the game and another to monitor the progress of the game. The foreman and project desk each have their part. The last two parts are a leaderboard and a client built into the admin environment. A navigation bar at the top of the screen allows the admin to change between different parts and different pages in these parts.

Control In the control part, several screens only consist of buttons, which only indicate whether they have been pressed, but most do not show the effect they have on the game or if they have any effect at all. Furthermore, after changing the game state or projector display and revisiting that page there is no indication of which option was selected.

Other screens allow an administrator to manage teams, clusters and events. These screens are all similar to each other. In these screens the currently created items are displayed in a table as shown in B.3. The input boxes next to or below the table allow the administrator to add items. In the teams page, there is no indication and limitation on the input of these fields, but only certain inputs can be used or have an effect. The input fields on the cluster and event pages do have a description in most places, but field to add a team to a cluster does not have a description. The event page does have a button to fire an event, but there is again no indication that it is triggered. The same holds for the finish and reset buttons of the teams page.

The configuration page initially consists only of a reset button and two input boxes which ask for a key and value, but does not describe what options can be added. After pressing the reset button a table is created with predefined options and default values.

The puzzles page consists of view of the properties of a created puzzle and a table with all the puzzles and some properties, as shown in B.4. An index above the puzzle allows for the navigation through the puzzles with their properties. For each puzzle, the properties are divided over several groups and each can be edited by replacing the input field. The overview table contains several empty columns and two of the three buttons, the refresh and delete button, do not work.

Progress The progress part consists of three parts. All three of these screens show statistics about certain aspects of the game.

The team window shows a table with a row for each team indicating which puzzles are answered and whether these are correct by marking them green. By hovering over puzzle in the row of a team, all attempts are shown in a hover box next to the mouse. The box expands as more answers are given and the answer length increases. This can lead to unexpected and undesired size.

The statistics view consists of three parts: a field which shows how many players of the total are online, a table with all the puzzles and the amount of team that have tried and solved them and evaluations, which isn't currently part of the functionality.

The cluster part consists of one big table to show for each cluster its name, colour and the amount of teams that have solved a puzzle. It does not indicate how many teams are in this cluster and the cluster colour field remains empty.

All tables in the progress section span the entire screen and all columns are the same size.

Project desk The project desk part consists of only one screen as shown in B.1. On the left-side, all the teams have button in the colour of their cluster, which can be pressed. Above the buttons is a search box in which team codes can be entered. On the right-side is the team view, a table with the information about which puzzles have been handed out to the team yet. In this table, every row contains which priority the puzzle has, the status of the puzzle and name of the puzzle. In the name part is a button if the puzzle status is available. By pressing this button, the colour and size of the entry changes and a undo button appears next to the puzzle status.

Foreman The foreman screen consists of three screens, which can be navigated by the four buttons on the left-side just below the navigation bar as seen in B.2. The first button shows all the clusters to the foreman, which can be selected. After a selecting a cluster, an overview is shown of each team in the cluster, which can also be done by pressing the second button. This overview is presented in the form of a table. The first columns consists of the team code and name, the next few columns monitor the progress of a team. The last column consists of a dropdown and a form. The dropdown's default option shows the name of the puzzle a team is currently working on as well as the amount of attempts and their last attempt. The remainder of the dropdown is filled with possible hints for the puzzle. Selecting a hint, also puts the hint in the form. This form can be submitted to the player by pressing the button. The third button leads to an instruction screen on which instructions are shown by means of bullet points. The fourth button, the automatic button, selects either the second or third screen depending on the state of the game.

Leaderboard The leaderboard view consists of one table. In the left column are the positions and in the right column the team names. Currently the left-side is equal to the amount of teams, but the right-side does not show their positions in the ranking.

Analysis of admin front-end In the previous system, there are several problems with the front-end of the admin environment. Many buttons do not show their effect, which makes it hard for the users to keep track of what they selected. Also with the input forms for adding new teams, it is unclear what is required from the user. While in other input forms a description is provided, this does not always make it clear for the user. For adjusting certain configuration options, the option needs to be re-added to update the value.

The editing of puzzle is not straightforward. When looking at a puzzle, all properties and variables are shown. Each variable field spans nearly the entire screen, which makes the overview of the puzzle large and obscure. To add a property to a puzzle, it needs to be selected in the properties menu, but it is not clear what the effect of these additional properties is. The edited and added properties need to be confirmed with a confirm button at the top-left of the page, which is not at an intuitive place.

A lot of pages include a table. Their styles are consistent, but they all have the same issues. Most tables span the whole page and each column is of equal size compared to the other in the table. This makes the tables not efficient, since the lots of empty space between the information. Also buttons are sometimes part of a cell in a table, when they should have a column for themselves. In the project desk part, the size and style of a row changes depending on the puzzle status. At first, the rows have a big button in the same cell as text. This button disappears when pressed and the text becomes smaller. This change of style and the use of the use of a button in the text cell feels unnatural. Also the undo textfield functions a clickable field, but this is hard to make up from its layout.

Throughout the administrator panel, there are several buttons which give a browser alert. Although the effect of an confirmation could be desired, a browser alert disables the user to switch tabs and the style of the alert cannot be controlled. Most elements on the different pages span the whole width of the screen. This results in elements looking out of proportion and as a user it can be harder to extract information from these elements.

Client

In the old system, the client consists of multiple screens: a game screen, a pre- and post-game screen and a suspension screen. Below each client screen there is a small black bar. In this bar is a clickable link to site of Corporate-escape and the privacy statement of Popup-escape.

Game screen The left-side of this screen consists for a two-thirds of the question screen as shown in B.7. A title indicates which question or part needs to be answered. Just beneath the title is a picture, which could be related to the question, a box with the question itself and an answer field. On the right-side is a scoreboard where the scores of the top six teams are shown. For each team the name and their score is shown. The playing team's score is shows just above the scoreboard. The question section of the game screen can turn into a question menu. In that case B.6, multiple possible questions are shown. The topics that the team can answer have a yellow accent, topics for which materials need to be picked up are blue and topics that are currently not available are grey.

Pre- and post-game The client side contains two pre-game parts and one post-game screen. The pre-game screen consists of a coloured background with a title in the middle and a form to fill in the team code to log-in, see B.5. Logging-in leads to the second pre-game screen. There is a text box with some general info and a second form to chose a team name. After submitting the team name, the team moves to the game screen. The post-screen part is similar to the pre-game views. It has the same coloured background and just a post-game message.

Suspension screen The suspension screen is nothing more than an empty page displaying a title in the top-left and a message just below the title, but in the middle of the screen.

Analysis of the client front-end The previous client UI does not have major flaws and serves its purpose. An improved suspension screen would be desired, but since it is not intended to be used often this is not a necessity. The pre- and post-game screens are simple, yet effective. The question screen are decent, but can be improved. The question itself is relatively small compared to the title and image on the page. Also the scoreboard does not line up with the rest of the page.

3.2.3 Functionality

This part will review the functionalities available in the original M.O.R.S.E system. It is divided into the most vital parts of the system.

Teams

Description In the system the players are represented as teams. Each team has at least one device with which they can interact with the game.

Each team has an unique team code, which has been determined by the administrator before the escape-event. The team enters the game by submitting their team code into the client part. After this the team can choose their own team name, which will be shown on the leaderboard. After this initial phase there are several possibilities, depending on the type of game.

One of the functionalities for the team is the ability to answer questions or fill in answers to puzzles, with confirmation if the answer is right. However, there are puzzles which require physical attributes, which need to be picked up at the project desk. Once the team retrieved the attributes for a certain question, the project desk will unlock the question for the team.

Apart from a queue of questions, there are also other options for admins to add to the game, for instance a prioritisation phase. In the prioritisation phase the teams can choose a certain topic in which they want to prioritise in. This will reflect in the questions which are available at the start.

The teams are also able to see the leaderboard on which they can see their own progress as well as the progress of the top five teams, in terms of amount of puzzles solved.

Apart from the device used to interact with the game, each team also has a phone linked to their team on which foremen can send hints to using the M.O.R.S.E system.

Analysis The functionality for the teams is quite straightforward. They need to be able to interact with the game to identify their team, solve their puzzles and see their progress.

These functionalities are fundamental and should be in the M.O.R.S.E system, however there is some functionality that could be added to increase the experience for the teams.

One of those functionalities is the possibility to receive hints as a pop-up on the device, while also having the possibility to read hints that already have been read. This functionality would make it easier to organise an event, because it would require less devices to be provided by the staff.

Clusters and Foreman

Description For the purpose of controlling the game, groups of teams are combined into clusters. Each cluster has a foreman assigned to it, who will monitor the process and send hints if necessary.

In the admin part of the M.O.R.S.E system, clusters can be created and teams can be added and removed. Also a colour can be assigned to a cluster, this can be used to track the clusters.

in the admin part, there is a dedicated page for the foreman. On this page the foreman can see an image of the placement of the clusters, this image is hard-coded. Using this the foreman can pick his position in the room. The foreman has also the ability to select certain clusters. From the selected cluster he can see general info, for instance the progress of the teams, but also send hints to certain teams inside the cluster.

As mentioned, the foreman can send hints using the option in the dedicated screen. If this option is used and the team has a phone number set, then the message of the foreman will be sent a message by SMS.

Analysis The part which is dedicated for the foreman has got enough functionality for a foreman to fulfil its role. However, the system is especially designed for one role, namely the foreman. There is no option for the foreman to only see their own screen and not other admin control functionality. Even though the M.O.R.S.E system is currently only controlled by staff members, it is beneficial to have this distinction in roles. A foreman can accidentally click on buttons which influence the game in a unwanted way. This would be prevented by having roles.

Project desk

Description On certain games there is also a section with a project desk. Some puzzles require physical attributes to be solved, which are at the project desk. Once a team unlocked a puzzle, they can go to the project desk and give their team number and retrieve the puzzle.

The project desk has its own screen on the admin part. On this screen the project desk can select a team and for that team it displays a list. On this list the different kinds of status can be seen. For the project desk these status are solved, delivered and available. At the project desk one can see whether a puzzle is solved, unlocked, locked or delivered.

Apart from typing the team code, the project desk has also the ability to pick a team from a list, in which both the team name and team code are mentioned. Also the colour of the cluster can be seen in this overview.

Analysis The project desk screen does have the functionality it requires to have for the game and there is not much additional functionality required if the concept of the game stays the same. The only part of functionality that could be added is the same as at the foreman, namely the role of project desk member, which ensures that you can not see what features that are not necessary for a project desk member.

Events

Description One of the main ways to control the projector screen is events. An administrator can create, delete and fire events. Events allow for dynamic behaviour during the game on the projector. An event has a name, type, trigger and optionally a next event. Each event has a type which can be video, timer or other. Timer is used for sounds and other is just a blank screen. For the triggers there are four simple options, namely manual, after previous event, timer-based and at the start.

Analysis Events are needed to create more complex structures for games. In the previous system events are used to display videos, show the timer and play sound. A better way to prevent having to use events for this, is the possibility to add an action (e.g. playing a video) to a timeline.

Events can be useful to automate parts of the game, especially if there are enough triggers available. However, at this moment the M.O.R.S.E application is lacking triggers which results in a limitation of problems which can be solved with the event system. Some suggestions for triggers to create more opportunities is for example to add score reached or all teams answered question. Using this, there are more possibilities. This could also be incorporated with having standard triggers on objects in the game, for instance for every question a trigger on unlocked, retrieved and answered, for a specific team or for all teams.

Puzzles

Description An important functionality of the M.O.R.S.E system is the ability to ask questions to the user. Most puzzles have physical attributes which must be used to find the answer to the puzzle. This answer could then be entered in the M.O.R.S.E system.

There is also a puzzle editor in which puzzles can be defined with a number of different parameters. Editing those parameters updates the questions in real time.

Each puzzle has an identifying number, an input type, a game part, a question and an answer. Other optional parameters are an image and a description.

Currently the possible input types of questions are text input and multiple (checkbox) inputs. However, if a multiple choice question has multiple correct answers, the answer is stored as a concatenated string of all the correct options, which makes it difficult to read.

Each question can also have one or more ‘wrong answers’. For these a special response can be set, instead of simply that the question is wrong. This can be used if someone is close to the answer, but missed one step, to help him/her automatically.

Furthermore, there are several states a puzzle can be in for a team. The puzzle can be locked until other puzzles are finished, the puzzle can be unlocked but not picked up at the project desk, the puzzle can be picked up but not solved yet and the puzzle can be solved. Those different states are available for the teams and the staff to see.

Analysis The attributes of the puzzles are well thought out and allow for additional functionality to help the teams.

One of the possible helpful functionalities is a question editor that does the work more efficiently, with an option to export one or multiple questions to JSON format and an option to import those again. Which allows saving the questions outside of the system.

The original version of the M.O.R.S.E lacks a proper way of implementing different kinds of questions. For now the editor only supports simple text questions and the previous games (without use of the editor) have simple text questions as well as checkboxes/multiple choices questions. The checkbox question’s answers are simply a string of all correct options combined together, which works but is not easy to read. A question with an array of correct options would allow for more dynamic UI options as well as the possibility to shuffle the possible answers.

Apart from that, some questions require a custom UI (e.g. a ‘crack the password’ puzzle). At this moment in time those puzzles are hard coded in the system. Adding flexible templates or layouts for questions should solve this problem, but this requires a clearer and more advanced way to add questions. It would be useful to be able to define a layout, and in that case a preview would be useful as well.

Administrator

Description The administrator has the ability to see every screen and has access to every functionality. The administrator has the task to control the game and manage all the processes. There are several distinct screens which the administrator can use to control the game, some of which are already mentioned, such as the puzzle editor and the event screen.

The administrator has access to the team edit part, in this part they are able to create new teams by giving team codes and have the ability to add a phone number to an existing team. The admin can also delete teams or reset the teams. Resetting a team results in resetting the puzzle state for the teams and their progress, the only thing they keep is their code, phone number and their name.

The administrator also has a screen in which he/she can control the flow of the game. The administrator can start the game, switch between phases, suspend the game, reset the puzzle status, and add or remove time to the game. The admin can also change the timer type (date or time) in the configuration, as well as start and end time.

Analysis The administrator has an important role in the system, namely controlling the game. One disadvantage of the old M.O.R.S.E system is that for a new escape event an administrator can not create a new event without changing the code to make a new version. For this some hard coded things in the back end should be changed, so there is a need for a programmer or someone with knowledge about the back end if a new event has to be setup.

Once a game has started the administrator has enough functionality to control the game flow as well as monitor the progress of the different teams.But for preparation and event creation there is definitely some lacking functionality, but once a game starts the administrator has enough functionality to do what he/she needs to do.

3.2.4 Back-end

Technology

Meteor The original system makes use of the JavaScript platform Meteor. Meteor is used for developing web and mobile applications. The developer can only use JavaScript to code, both client- and serverside [5]. Furthermore, Meteor has great integration with MongoDB, NPM and Blaze, which the old system makes use of.

Blaze The previous system makes use of the HTML template library Blaze. Blaze has two major parts: a template compiler and a reactive DOM engine. The template compiler converts the created template into JavaScript code. The DOM is build by the reactive DOM engine. The engine also manages and reactively updates the DOM at runtime [6].

Mongo The old system makes use of MongoDB [7] (server) and Minimongo [8] (client) as databases. Minimongo, a type of Mongo database, stores data in JSON-like documents. MongoDB / Minimongo is the only integrated database in Meteor. The high built-in reactivity is also an advantage for the original system. The system uses this reactivity for updating the availability of the puzzles and the scoreboard. While the main database is kept on the server in MongoDB, Meteor is able to publish and store parts of it in Minimongo on the client's device.

Data structures

The data is stored in collections, these collections can be published such that the entire application could retrieve the data. There are several different collections in this app, some of which are not used in the previous version of the M.O.R.S.E system, in this paragraph the original structure of the most important data will be explained and reviewed.

Teams All the teams are stored in a team collection. When created, each team only exists of a team code. When a team logs in, the team name, online status and puzzle states are also stored. Some of the puzzle state attributes are initialised as `null`, which then can also be left out until they would be used. In the old structure, the puzzle states are not linked to the puzzle table, which can cause problems when adding or removing puzzles. Also, the online status is stored twice, which is unnecessary.

Clusters Each cluster has a list of team ids and, to keep track of the scores, an array with an entry for every puzzle, the integer indicates how many teams in the cluster solved that particular puzzle. Here the same holds for the scores system: once something happens that makes the score of a team invalid it needs to be reset, which does not happen in the original version. The score per puzzle can also be retrieved by simply counting the amount of solved entries from the teams belonging to that cluster.

Events Each event that is created by the admin forms an entry in the collection of events. Each event has the attributes defined by the admin consisting of name, type, trigger and possible next event as well as an attribute called 'active'. This attribute determines whether an event is active and happening at that moment in time. There could be at most one event happening at the time, which might be enough if the events are only to control the projector screen, but if events were to be used for more than only the projector it might be limiting that no events can happen in parallel.

Scores To have the functionality of a scoreboard a table for scores is added, it contains a team name and the score. Every time a team gains points their entry in the scores gets updated. Once updated all other clients will also receive that update and therefore see the live version of the scoreboard. Unfortunately this requires deletion of the entry in the scores if the team gets reset or the team gets deleted, this is not implemented in the old version which creates unwanted behaviour. A feasible way of solving this problem would be to retrieve the amount of solved puzzles directly for the team and determining a score based on that.

Game data The game data collection consists of all important configuration options and game states which need to be available for both the admin and the client part. To track the phase there is a field this creates an unnecessary complication.

Puzzles All the puzzles are saved in the puzzles collection. This allows the admin to edit puzzles and update them, such that they are updated live for all clients. The puzzles have a lot of fields, such as question text, title, answer, options, image, game phase and a template.

Maintainability
The original system has been used for different events and each time it has been adapted to fit the right questions and information. Since it changes frequently, it needs to be maintainable. This section will look at two factors, testing and code quality, to asses the maintainability of the old system.

Testing The previous system has no test cases to test its code. This makes the entire system less maintainable, because once a bug occurs, it might be difficult to locate the bug. Also a system without tests is more bug-prone, since during development some mistakes might go unnoticed while testing the functionality manually.

Code quality The code quality of the old system could be improved. The previous system contains some comments explaining what a method or part of a method does, but this is not present in most cases. There is also a lot of code, which is currently unused or commented out.

The variable names make sense, but they are not written consistently. Some are in caps, others camel cased and others have no capital letters at all.

The cyclomatic complexity in the M.O.R.S.E system is not good, since there are methods with a large amount of paths. Some files do have one or multiple unused imports. In some cases the code contains very long lines which makes it more difficult to understand what is happening in that line. Furthermore, the indentation is not always correct making it harder to read.

3.3 Requirements analysis

In this section, we will investigate what the requirements for the new system will be. For this, we define who the stakeholders for this system are, i.e. who will be using it. Next, we determine what functionality is required by each of these stakeholders. Finally, we establish non-functional requirements which are needed for a system which can confidently be used in production.

3.3.1 Stakeholders

There are a few stakeholders that will use the created solution. Each group of stakeholders has their own objectives and expectations from the system.

Player
The first group is the players that will use the system to play the game. They only use the system during the escape event. They have no prior knowledge about the system or the way it works. Players will use the system on portable devices like laptops, tablets or mobile phones.

Game-designer
The game-designer is tasked with setting up the system before an event. The game-designer uses the system multiple times for multiple events and can receive training or documentation to handle the system. The game-designer will use the system on a computer.

Game-host
A game-host is responsible for managing and controlling the game during an event. Although the game-designer and game-host are often the same person, some events are hosted by someone else without prior experience. The game-host will use the system on a computer or tablet.

Foreman
During an event, the game-host is supported by a group of foremen. A foreman is assigned to a cluster of teams and is tasked with monitoring their progress and helping them through the event. The foreman is not necessarily a regular user of the system, but will receive training for using the system. The foreman will use a portable device like a laptop or tablet during the event.

Project-desk staff member
The final stakeholder is a staff member at a project-desk. The project-desk is tasked with handing out physical puzzles to teams who have access to a puzzle. The project-desk staff member can, like the foreman, be a one time user of the system but can be briefed in advance. The project-desk will use a computer or tablet to run the system. Multiple staff members could use one or multiple devices.

3.3.2 Functional requirements
To describe the functional requirements of the system, we will use the *MoSCoW* method [9]. For each of the stakeholders, we will describe must haves, should haves, could haves and would / won't haves.

Team
For a player, the system enables communication in two directions: teams receive information about puzzles, the progress of the game and extra hints. Teams themselves can make choices and validate answers.

Must haves
- The team must be able to sign-in using a team code.
- The team must be able to set a name for their team.
- The team must be able to only see puzzles which are available for their team.
- The team must be able to solve puzzles and see if their answers are correct.
- The team must be able to see the remaining time of the game.
- The team must be able to answer puzzles only if the physical objects for these puzzles are collected.

Should haves
- The team should be able to see the score of their team, based on the amount of solved puzzles.
- The team should be able to see the leaderboard with scores of other teams.
- The team should be able to see a text once initiated by the game-host, e.g. when the game is paused or after the finish of the game.
- The team should be able to view hints sent by a foreman.
- The team should be able to log out.
- The team should be able to see an overview of all solved and available puzzles.

Could haves
- The team could be able to view an overview of all provided hints sent by a foreman.
- The team could be able to give feedback on their experience (e.g. with smileys and a short feedback question).

Would / won't haves
- The team would be able to select a preferred language and use the system in this language.
- The team would be able to request help from a foreman.
- The team would be able to to chat with a foreman.
- The team would be able to play visual escape-games.

During the event, players will also make use of the system as displayed on the projector. These projectors provide additional information during the event. The players will, depending on the phase of the game, see remaining time, videos, the leaderboard or the winner. This is a one-way communication, as players cannot interact with the projectors.

Must haves

- On the projector, the player must be able to see the remaining time of the game.
- On the projector, the player must be able to view a video.
- From the projector, the player must be able to hear a sound.
- On the projector, the player must be able to see a pause screen with text or an image.

Should haves

- On the projector, the player should be able to see the winner of the game.
- On the projector, the player should be able to see the leaderboard.

Could haves

- On the projector, the player could be able to see a presentation of statistics after the game has finished.

Game-designer
A game designer will use the system to prepare the system for events. The game designer will create events, set up puzzles, configure timers, schedule videos and configure triggers and actions. This stakeholder does not have any must haves, as an event could be executed with a hardcoded configuration.

Should haves

- The game-designer should be able to create a new event.
- The game-designer should be able to change predefined configuration options for the game.
- The game-designer should be able to create a puzzle and specify the question, answer type and correct answer.
- The game-designer should be able to indicate that collection of physical objects is required to open a puzzle.
- The game-designer should be able to predefine hints a foreman can send to a team for each puzzle.
- The game-designer should be able to add stages to a schedule, change the order of stages in the schedule and remove stages from the schedule.
- The game-designer should be able to add multiple puzzles to a stage.
- The game-designer should be able to specify video, image and text screens to be used in stages.
- The game-designer should be able to setup actions to alter the game state based on triggers in the game.
  Example triggers: game-host presses button, remaining time, team / cluster / everyone solves a puzzle.
  Example actions: play a sound, play a video, change projector screen, move to item in schedule, change visibility of a puzzle, alter the timer.
- The game-designer should be able to change the styling for the client side.

Could haves

- The game-designer could be able to specify the amount of points a team receives for answering a puzzle correctly.
- The game-designer could be able to duplicate an event.

- The game-designer could be able to prepare automatic hints if a team gives specified answers.
- The game-designer could be able to create input templates in the admin interface with code.
- The game-designer could be able to add evaluation questions to the schedule.
- The game-designer could be able to export answers after a game has finished.
- The game-designer could be able to track the time it took for teams to solve each puzzle.
- The game-designer could be able to create statistics screens for after the game has finished.
- The game-designer could be able to let the system call a webhook on certain triggers.
- The game-designer could be able to let the system perform actions when a webhook is called.

Would / won't haves

- The game-designer would be able toimport and export the entire configuration of the game.
- The game-designer would be able tocreate dynamically generated puzzles.
- The game-designer would be able tovisually build in-browser puzzles, e.g. escape puzzles.
- The game-designer would be able tobuild a dynamic map of the locations of the groups in the room.

Game-host
A game-host controls the system during an event. The game-host will setup teams and clusters, start the game, control the stages and setup the timer. During the game, the game-host will have an overview of the progress of the game.

Must haves

- The game-host must be able to create teams.
- The game-host must be able to start the game.
- The game-host must be able to create clusters.
- The game-host must be able to add teams to clusters.
- The game-host must be able to alter the remaining time.
- The game-host must be able to view the progress of all teams.
- The game-host must be able to control what is shown on the projector (e.g. remaining time, leaderboard, winner).
- The game-host must be able to reset the game.
- The game-host must be able to trigger (a group of) actions with a button.

Should haves

- The game-host should be able to move a team, cluster or everyone to a selected stage.
- The game-host should be able to pause the game.

Could haves

- The game-host could be able to alter or reset the status of puzzles for a team, cluster or everyone.
- The game-host could be able to set a phone number for a team.
- The game-host could be able to hide teams which are not participating.

Would / won't haves

- The game-host would be able to roll back to a point in history of the game.

Foreman
A foreman has responsibility over a cluster of teams. During the game, the foreman monitors the progress, answers questions and gives hints.

Must haves
- The foreman must be able to select and set the cluster (s)he is responsible for.
- The foreman must be able to list the teams in their cluster.
- The foreman must be able to have a visual overview of the progress of each team.
- The foreman must be able to send hints to one or all teams in their cluster.
- The foreman must be able to see given and correct answers for each puzzle.

Should haves
- The foreman should be able to view instructions for each stage.
- The foreman should be able to have an adaptive overview based on the stage of the game.
- The foreman should be able to send predefined hints for puzzles.

Could haves
- The foreman could be able to simulate the screens a team can see.
- The foreman could be able to send hints as text messages to teams' phone numbers.
- The foreman could be able to update the puzzle status for a team in their cluster (i.e. mark as solved).

Would / won't haves
- The foreman would be able to have an overview of teams that have requested help.
- The foreman would be able to communicate with the team via chat.
- The foreman would be able to select groups based on location in the room.

Project-desk staff member
A project-desk staff member will use the system to see which team can pickup which puzzle. In the system, the project-desk-staff member will have an overview of teams and puzzles which can be picked up.

Must haves
- The project-desk staff member must be able to select a team from a list, or by team code or name.
- The project-desk staff member must be able to see which puzzles are available for pickup for a specified team.
- The project-desk staff member must be able to mark a puzzle as picked up.

Should haves
- The project-desk staff member should be able to undo a pickup in case of an error.

Could haves
- The project-desk staff member could be able to sort the list of teams based on the likelihood of them coming to pickup a puzzle.

Would / won't haves
- The project-desk staff member would be able to receive instructions to deliver puzzles at teams' tables.

3.3.3 Non-functional requirements
This section goes through the the requirements which are important to the development of the system.

Maintainability
Maintainability is the probability that a failed system can be restored to its normal operable state within a given timeframe. In other words, how well can defects or their causes be corrected, how well can unexpected working conditions be prevented and by how much can the maximum useful life of the product be extended for? To this end, code coverage and the code evaluation from the Software Improvement Group (SIG) are used as metrics.

Modularity
Modularity is about separating the functionality of the system into as many independent interchangeable modules as possible, such that each contains everything necessary to execute only one aspect of the desired functionality.

Usability
Usability is the ease of use of the system for both teams and admins. If implemented properly it ensures a smooth experience for both types of user.

Deployment
Software deployment is all of the activities that make the system available for use. This is an important metric when you, for example, make an adjustment and want to quickly redeploy the system.

Scalability
Scalability is the property of the system to handle a growing amount of work by adding resources to the system. Scalability is important to assist and track the hundreds of participants.

Performance
Performance is how well the system runs, its importance cannot be overstated. As a smooth experience must be provided in every phase of the game.

Robustness
The robustness of the system is its ability to deal with errors while it is running and how well it can cope with erroneous input. This is important as, no matter how much you test, if 400 people are all at once trying out different things with your system, you are bound to get some erroneous input.

Security
Security is the state of being protected against the criminal or unauthorised use of electronic data, or the measures taken to achieve this. It is of importance to ensure that the security of the system is up-to-date, so that no participant hacks the system and cheats to win.

4 Design

This chapter will document the initial ideas for a solution to the proposed problem of Raccoon Serious Games.

Section 4.1 gives an overview of the different technologies that were considered and researched for a potential solution, as well as the choice of improving the old system, or starting over from scratch. Section 4.2 discusses the initial design ideas we had for the new system.

4.1 Design choices

This section will give an overview of the possible design choices that were considered for this project. Several different form factors and technologies will be compared, in order to select those which are right for creating a new system.

4.1.1 Technology

Form-factor

When it comes to deciding the type of system (e.g. Server Software, Desktop Program, Mobile Application or Web Application) that will be produced, the choice is rather straightforward.

First of all, because the system will be used for events with large groups of (new) people, the system needs to be easily accessible. Currently the participating players will receive a tablet or laptop from the client company, which is already set up with the system. However, there are plans for future events where players should be able to use their own phones. As such, installing the system should be easily understandable and take minimal effort. For this reason, a web application seems to be the most appropriate option. A mobile application is also a good option, but the biggest drawback would be that all players/teams would have to explicitly install it to their phones. There is a multitude of reasons why this is not always possible: lack of storage space, unsupported OS, no permission on the device, etc. However, a web application has the potential to be supported by phones, tablets and laptops, as they can be accessed with any modern web browser.

Second of all, because the original system is a web application, the client company is accustomed to this form-factor. This poses a strong argument for developing the new system as a web application as well. Furthermore, there is no current or future functionality that requires more features than a web browser is able to provide.

Language & framework

This section will describe some of the desired features, when looking for a framework, and subsequently compare some frameworks that meet these demands.

Requirements

Firstly, the system needs to support real-time messaging between the clients and the server. For example, if a team solves a puzzle, their score should update on all devices. Furthermore, the game host should be able to update the timer and send this change to all teams.

Preferably the selected framework therefore supports methods such as WebSockets [10], allowing the server to push changes to the clients. While it is technically possible for clients to update through the use of polling or AJAX, it is not desired, because then the server would have to deal with continuous requests from around 100 clients at the same time.

Secondly, the framework should easily support a database connection. The overhead for accessing, storing or changing data in a database should be minimal.

Thirdly, the framework should be adequately testable. Since testing the developed software should be a big part of any software project, selecting a framework with proper support for things such as unit, integration and system tests is preferred over choosing one without it.

Lastly, for ease of use, it would be beneficial if the framework uses a language the team is already familiar with. Practically this limits coding to the most common languages such as Java, Python, PHP and JavaScript / Node.js.

This is not a 'hard' requirement, but it will help narrow down the available options for frameworks, because there are so many. Furthermore, choosing a commonly used language also makes the code more maintainable for the client.

Frameworks

No Framework The first option is to use no framework, and pick a language and some libraries that meet the demands given above. With this option there is a lot of variability, i.e. it is necessary to mix and match libraries, or to implement existing functionality ourselves. However, choosing libraries also comes with many drawbacks: Every library would have to be researched individually, their documentations and implementations can vary greatly, and it would therefore take a lot of time to set up functionality, which is already present in a framework right from the start. Overall this option would probably cost the most time. Furthermore, it would require testing the chosen libraries as well.

Some possible implementations include: A Node.js server combined using Express[1], Socket.io[2] and some form of database driver as Middleware, or a standard web server[3] using a PHP codebase with Ratchet[4] and a database library.

Feathers Feathers.js is a Node.js API that acts as a wrapper for Express, Socket.io, and many other Node packages [11]. It has APIs for many different databases and is compatible with different rendering engines such as React and VueJS. It serves more as glue code than a full-fledged framework, and as such is very modular, but also requires more set-up. For example, it is still necessary to establish a WebSocket connection yourself, instead of letting the framework take care of it.

Since Feathers.js is a Node.js framework, it is possible to use any package from NPM. This includes test scripts with using `npm test`, and usage of different testing frameworks. Furthermore, Feathers.js has an extensive guide for testing applications[5].

Firebase Firebase is a platform by Google that supports Realtime databases, synchronisation and hosting [12]. It is not a framework, but a hosting service capable of running "Cloud Functions" instead of a proper back-end code base, to manipulate the database.

An advantage of FireBase for this project is its scalability, as Google will automatically assign more resources when more Cloud Functions will be called through FireBase's client API [12]. A drawback is the fact that FireBase is not a framework, but a hosting platform. As such, there is no back-end code besides the Cloud Functions, which are limited in what they can do. However, due to the nature of the platform, deploying the application would be relatively easy.

Another advantage is FireBase's extensive testing platform. Not only can Cloud Functions be tested with both off- and online unit tests, FireBase applications can also be ran on real devices in Google data centres, simulating real usage [12].

[1]`https://expressjs.com/`
[2]`https://socket.io/`
[3]Apache HTTP, nginx, etc.
[4]`http://socketo.me/`
[5]`https://docs.feathersjs.com/guides/chat/testing.html`

Meteor Meteor is a full JavaScript framework built on Node.js for realtime applications. Meteor comes with an extensive MongoDB integration and its own templating language called ‘Blaze’, but other templating languages are also supported. Furthermore, while Meteor can use any NPM package, it also comes with its own Marketplace ‘Atmosphere’, with packages designed for Meteor [5].

A big advantage of Meteor is that the original M.O.R.S.E system is also built in Meteor. Building the system with Meteor would therefore help with maintainability, since the client also has knowledge of Meteor. Furthermore, the previous system is also deployed to a special hosting platform designed for Meteor applications. Sticking with Meteor for the application would require the least amount of changes to this deployment process. Also, Meteor has extensive features available for testing, for instance, a special “test mode” that automatically loads test files, which support database operations, generating fake data, mocking, etc. [5].

However, a potential disadvantage of Meteor is its lack of support for databases other than MongoDB. This is because Meteor achieves synchronisation between the client and the server by storing part of the server’s MongoDB database in the client’s local storage using Minimongo [5]. This is actually desired for the project, since it ensures that all connected clients will have access to the same data. Additionally, according to the client, Meteor does not always deliver the promised performance, but nevertheless it is suitable for the client.

For the reasons mentioned above, it was decided to use Meteor. There was not a specific preference for any of these frameworks, as each of them comes with its own positives and negatives, but it was believed that the client’s knowledge of Meteor will be beneficial during and after the project.

Front-end framework

There are various technologies which support the rendering of client and administration screens. This section compares these technologies and discusses which option is most suitable for the desired system.

In Meteor, the used framework, the use of a front-end framework is required. Meteor integrates with a couple of front-end frameworks: Blaze, React, Angular and Vue. Each of these frameworks uses templates with reactively updating variables to update the user interface based on data from the server.

Blaze Blaze [6] is the original front-end framework for Meteor. It is a simple framework built around the way Meteor handles the client side database. Compared to other options, Blaze has less functionality than other frameworks, which makes it easier to learn but means more manual work to get advanced functionality working. Blaze allows access to variables in a raw way, while other frameworks enforce stricter data control. This is an advantage in terms of freedom but can be a disadvantage in code quality and maintainability.

Blaze is developed as part of Meteor to at first be the only front-end framework. Later, other front-end frameworks were integrated into Meteor, decreasing the need for a Meteor-specific framework [13]. The Blaze framework has not been updated since September last year.

React React [14] is a front-end framework developed by Facebook. Of the possible options, React is the most used one. React uses JSX as language to write HTML in JavaScript. JSX has a slightly different syntax from normal HTML. Also, to implement for example conditional rendering, normal JavaScript is used. The same holds for the handling of forms; React does not offer out of the box methods to bind form inputs to data fields and validate user input. Instead, this is done with JavaScript events.

Angular Angular [15] is developed by Google and splits templates in HTML, CSS and JavaScript files. Angular uses components to define views which can be rendered, and makes it possible to inject services to for example reuse code in multiple pages. This makes it easy to build a modular front-end by splitting screens into smaller components. Angular and Meteor work together through an open source library which makes the interaction between server and client easy.

Vue Vue [16] is the most recent addition to the possible front-end frameworks for Meteor. Although it is not officially supported, Meteor provides documentation on how to use Vue with Meteor. Like Angular, it uses components to divide an application into smaller parts. These components can be defined in one file containing the HTML, CSS and JavaScript in separate parts. Through an external package [17], data from Meteor can be inserted into Vue templates reactively. Because the integration between Meteor and Vue is relatively new, compared to React and Angular, there are fewer resources on how to use the combination. Vue however seems to have a better learning curve as it is designed to be incrementally adoptable, i.e. one does not have to use and understand every part to be able to get started.

Picking Blaze, React, Angular or Vue seems like an arbitrary choice. Each framework is suitable to build the system and has different benefits and drawbacks. In terms of performance, each framework is built to be high-performing. Various comparisons recommend not looking at the performance when choosing a framework [18, 19], since the small difference in performance does not weigh up to other factors. Because Blaze is not actively under development and has less functionality than the other frameworks, this is not the best option. React seems to have a steep learning curve for a project of 10 weeks, making it inefficient to choose this framework. Since Angular is used more with Meteor than Vue is, there is more documentation and support for Meteor-Angular. That is why it was decided to use Angular for the project.

Code quality

For code quality, Meteor has a detailed guide in their documentation on proper code style [5]. They recommend the use of ESLint as a static analysis tool.

Not only is ESLint used by many big companies, such as AirBnB[6] and Google[7], which already have good configuration profiles available, ESLint itself supports Meteor through a plugin and a recommended configuration[8].

Another advantage of ESLint is the fact that the IDE, PHPStorm[9], has built-in support for it. Furthermore, ESLint configuration files can be imported directly into the IDE. This means that instead of having to run the static analysis manually, it is running continuously as you type.

While there are some alternatives to ESLint, namely JSCS, JSLint and JSHint, they are not suited for Meteor, since Meteor actually uses ECMAScript, and not pure JavaScript [5].

Testing technology

The testing environment is another important feature. Luckily, Meteor has their own guide[10] on the subject. It contains useful information such as the challenges of testing in Meteor. The first challenge is the client/server data, as Meteor closes the client-server gap, it becomes critical to test the code that actually works across that gap. Meteor's full app test mode allows for integration tests that cover both client and server side interfaces in the program [5]. Another challenge is the reactivity, as Meteor's reactivity system is eventually consistent, this means that when you change a reactive input to the system, the user interface will take some time to reflect to this change. This can be quite a nuisance during testing.

Meteor also has integrated support for the Mocha[11] test framework, which is one of the most popular frameworks for Node.js testing.

JavaScript has many other great tools at its disposal. One of those is jsdom[12], a simulation of a browser's environment without running anything but JavaScript.

4.2 Design ideas

To create a system which is able to both make managing a specific event as easy as possible as well as be flexible enough to be used for different types of events, the system is built using design ideas which are also reflected in the requirements. This section discusses these design ideas and provides reasons for why these idea's help to achieve the goals of our system.

[6] `https://github.com/airbnb/javascript`
[7] `https://github.com/google/eslint-config-google`
[8] `https://www.npmjs.com/package/eslint-plugin-meteor`
[9] `https://www.jetbrains.com/phpstorm/`
[10] `https://guide.meteor.com/testing.html`
[11] `https://mochajs.org`
[12] `https://github.com/jsdom/jsdom`

4.2.1 Events

The current system can only be used for one event at a time and has to be adapted before each event. To prevent this kind of overhead, the new system should incorporate a way in which multiple events can exist in the system at the same time.

These events are accessible using a specified URL. In this way, the game-designer can prepare different events without having to set-up or redeploy the system.

4.2.2 Schedule

To give game-designers the capability to build an event with every possible structure, the way views for the teams are defined has to be robust. An event will have a schedule with all the possible views a team can see. The items in this schedule can have different types, like a view with a puzzle, a group of puzzles, an image or a view with text.

Since the flow through the stages is not necessarily in order and is dependent on the state of the game, the ordering in the schedule cannot be used to determine this flow. Instead, the game-designer should be able to specify the flow through the stages based on rules i.e. with rulesets.

4.2.3 Puzzle statuses

A puzzle can have multiple statuses. These statuses should be reflected in the state in the game. A puzzle can be open or locked, determining if a team has access to the puzzle. A puzzle can require a collection of physical attributes. If a puzzle requires a physical attribute, whether the team collected the attributes determines if the team can answer the puzzle in the system. Also, a team can have solved a puzzle. The combinations of properties can be reduced to four states: locked (L), available (A), open (O) and solved (S). The truth table below describes the logic involved in determining the state of a puzzle.

Open	0	0	0	0	0	0	1	1	1	1	1	1
Needs pickup	0	0	1	1	1	1	0	0	1	1	1	1
Has picked up			0	0	1	1			0	0	1	1
Solved	0	1	0	1	0	1	0	1	0	1	0	1
Status	**L**	**S**	**L**	**S**	**L**	**S**	**O**	**S**	**A**	**S**	**O**	**S**

Table 4.1: Logic to determine puzzle statuses

These four states will be computed when displaying the state. Whether a puzzle is open, solved and/or picked up should be stored per puzzle per team, and whether a puzzle needs to be picked up at all needs to stored per puzzle, ultimately making all state transitions possible.

4.2.4 Projector

Since the views on the projector do not coincide with the views on the teams devices, these views can not be a part of the schedule. Instead, these views should be controllable during the game, either manually or automatically. To enable this, the projector can be linked to rules. Then a game-developer can, for instance, specify that the projector should open the leaderboard when the game starts.

4.2.5 Rulesets

For each event, a form of logic is involved. Firstly to define a flow through the game, e.g. the user should go to another puzzle when it solves one. Secondly to control the projector. Thirdly to enable features to enhance the game experience, like awarding extra points for the one who solves a puzzle first or opening puzzles based on the answer of a team. Fourthly to make it easier for the game-host to prepare sequences of actions which can be used during the event.

A lot of this logic could be incorporated in the existing components. For instance, a schedule item could define where the team should go when solving the puzzle. This does however not provide enough flexibility to customise events. Instead, this can be done using a set of definable rules which determine what should happen. Like in the tool If This Then That (IFTTT)[13], one could link triggers from different parts of the game to actions to alter the games state. This can be extended with conditions to make more complex rulesets possible.

[13]https://ifttt.com

5 Implementation

This chapter will look at the how the new system was implemented. First the overall structure of the application will be explained, which will be followed by the structure of the database. Then a more in-depth explanation is given about the main components of the application, namely schedules, puzzles, users, event configuration and rulesets.

5.1 Overall structure

M.O.R.S.E 2.0 is built with Node.js using the Meteor platform for real-time connections between the clients and the server. The code itself is written in TypeScript and divided over three separate modules: Server, Client, and shared code. This section will give an overview of these three modules, and their purpose within the application. A visual overview is given by figure 5.1

5.1.1 Server

The server module contains the code that is executed by the Node.js server. This module is responsible for the main logic of an event, and is the only module that has a (direct) connection to the database. The server module is divided into two submodules: publications and methods. These modules will now be described in further detail.

Publications

In order for data such as puzzles to be available to a team in a browser, the data needs to be sent from the server to the client. By default, Meteor does not send any data to a client, besides its userprofile, if the client is logged in. All the other data has to be 'published' to the client.

In the Publications module, parts of the database are published to clients using Meteor's publication API. Publications are defined with a name and a function that should be called when a client subscribes. This function contains logic determining what data should be available to the client. Here it is possible to check for user's authentication. For example, a regular player should not be able to see the answers to a puzzle, or the answers of other players, while this data is crucial to the foreman.

Using Meteor's "livequeries", the data that is sent from the server can be constantly changing and "reactive" on the client. This means that if data that was initially sent is changed or removed, or if new data is added, the data will on the client will reflect these changes as a result of the publications.

Methods

When a client needs to enact a change on the server, it needs to call a remote function. In the Methods module, the server's Remote Procedure Calls (RPCs) are defined using Meteor's Method API. These methods allow players to submit answers to puzzles, and project desk members to mark a puzzle as delivered. In these functions, there are also checks to verify whether a client has the permission to execute them.

For a client, calling a method is the same as calling a regular JavaScript function, only it happens on the server and returns its response by means of a callback. An overview of the connection between the client and the server is given in figure 5.2

5.1.2 Client

Clients of the M.O.R.S.E 2.0 system are internet browsers. In order to use the application, users connect to its URL, and are subsequently presented with a website that connects to the server using Meteor client library, which connects to the Meteor Server using its own DDP protocol.

This website is dynamically generated using Angular, which consists of larger “modules”, and “components” within these modules that built the individual pages. Furthermore, to provide a layer of security, Angular “Router guards” are implemented in the client module as well. These three parts will be outlined in the following sections.

Modules

Angular modules are groups of Angular components and other Angular modules, that together form a part of the application. To illustrate, M.O.R.S.E 2.0 has modules for the player screen, the administrator screen and the projector screen. Each of these screens can be accessed using a specific URL, resolved by the Angular Router, which maps a route to a component. In a module directory, there are two directories: “parts” and “pages”, which contain the components of the module.

Components

An Angular Component is a combination of an HTML template, an SCSS stylesheet and a TypeScript class, which together make up the structure, look and functionality of a page or part of a page. Components represent single objects in the Document Object Model (DOM), and can be included in other components, in order to build a full page.

In a component, data is retrieved from the database by subscribing to the server’s publications, and subsequently inserted into the template. Because of Meteor’s real-time functionality, the data in the client is updated, whenever the data it is subscribed to is changed. Thanks to Angular, these changes are also reflected in the template and therefore the screen of the user.

Router Guards

Router guards are small functions that are imported into Modules and called whenever the Angular Router attempts to load the component they are assigned to. They can allow or block a routing attempt, or enact a redirect. Router guards act as a layer of security, redirecting clients to and from login pages if they are (not) logged in, and prevent players from accessing the administrator screen.

5.1.3 Shared

Since Meteor is a full-stack framework, it makes use of “Universal JavaScript”. This means that, besides having separate code for clients (in this project’s case, a browser) and servers, there is also code that is shared between the two. This code is stored in this module, stored in the “imports” folder, and this directory is split up into two subdirectories, “collections” and “helpers”, which will now be described in further detail.

Collections

In collections, all the project’s entity classes are defined. These classes represent the building blocks of the application, such as a `Puzzle` class or a `User` class. This code is shared between the clients and the server, because both the client and the server use the fields in these classes.

Furthermore, in these classes database collections are set up for the top-level entities. Through these collections, the database can be accessed on both the client and the server. The big difference, however, is that the server uses the real database, and the clients use a local copy of the database, containing only the data they are subscribed to (see 5.1.1).

Helpers

The helpers directory contains miscellaneous functionality that both the client and the server make use of.

5.2 Database Structure

This section will describe the structure of the database of the new M.O.R.S.E 2.0 system. It will go into detail on the type of the database, the model layer between the database and the application, the use of the database in the application, and the different types of objects stored in the database.

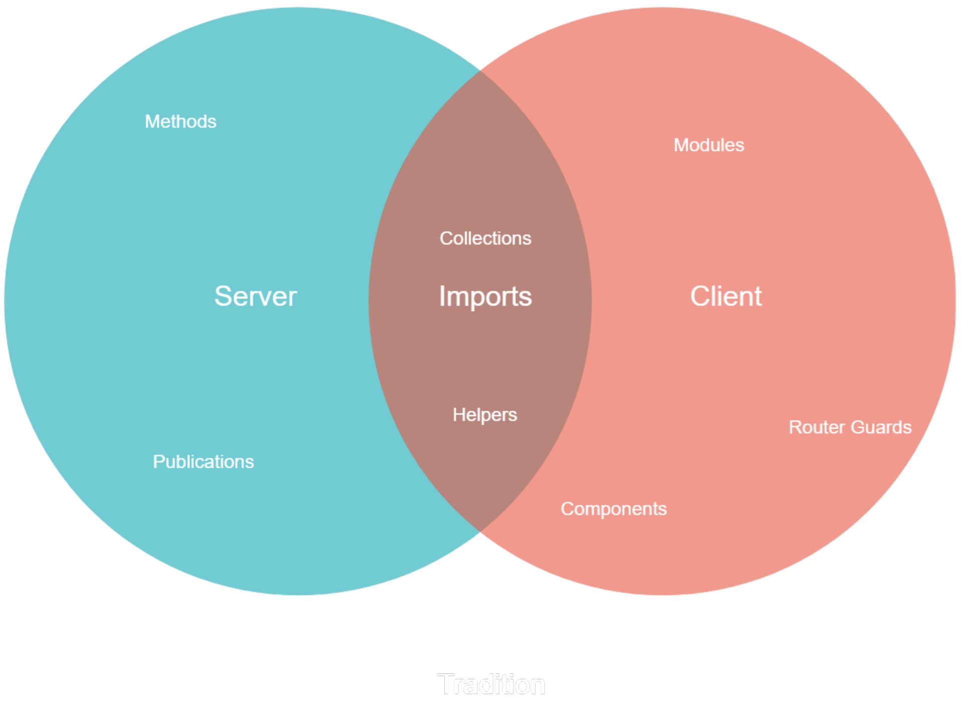


Figure 5.1: An overview of the different modules of the system

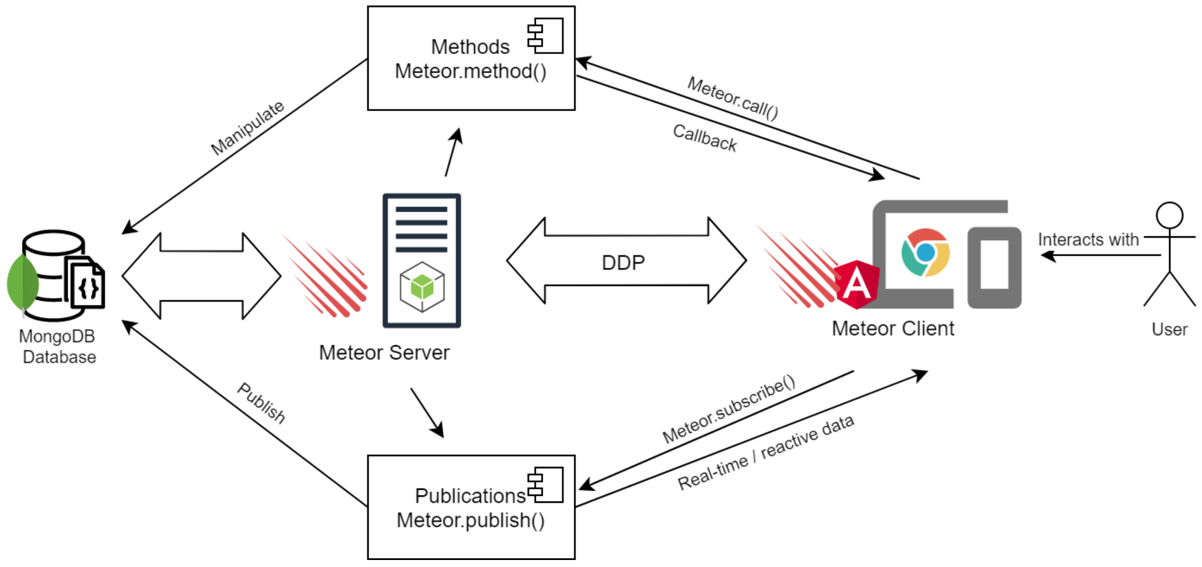


Figure 5.2: The connection between the client and the server, handled by Meteor

5.2.1 MongoDB

The application uses a MongoDB database. MongoDB is a NoSQL database, which stores its data in JSON-like documents, which are stored in Collections. MongoDB is the default database of Meteor, which is why it is used in both the old and the new system. While other databases can also be used with Meteor, the support for them is limited, leaving MongoDB as the best choice overall. Furthermore, clients use a JavaScript implementation of MongoDB, called minimongo. The minimongo database is automatically updated when there is a change in the server's database (More about this in 5.2.3).

MongoDB is not a relational database, and there is no built-in support for relations using keys such as a primary or foreign key. However, in order to avoid storing all event details in a single event document, which would need to be updated and sent to the clients whenever a team attempts a puzzle. For example, there are separate collections, which represent different classes from the object model. Since MongoDB provides each document with an '_id' field, this field is used to set relations between objects and classes.

5.2.2 Astronomy

Astronomy[1] is a Meteor package that provides Meteor with a Model Layer for the MongoDB collections. It serves as the Object Document Mapping (ODM) system.

Astronomy provides the possibility to define classes, which contain fields with predefined types, functions and even Meteor methods, which can be called on the client to enact functionality on the server. It also includes OOP features such as inheritance, and nested classes which are stored as subdocuments in a document and not in a separate collection.

Furthermore, the way the MongoDB database is accessed with Astronomy, is exactly the same as the way to access it without Astronomy, except with Astronomy, the queried documents are already mapped to objects.

Another benefit of Astronomy is validation. Astronomy objects automatically validate their fields when they are saved. Examples of validation include: the type of the field, which values the field can be set to, etc.

5.2.3 Reactivity

One of the key features of Meteor is the real-time functionality it provides. Querying a database inside a Meteor 'Tracker' causes the computations using the queried objects to be re-run whenever the queried data changes. The Astronomy query functions return a MongoDB Cursor object, which is 'reactive' by default.

This functionality is used in nearly every Angular component of the new system, in order to provide the players, project desk members and foremen with instant updates about the status of a puzzle, for example.

A drawback of MongoDB not being a relational database, is that the relations used in the system are not tracked in reactive database queries, since most of the database is normalised, as opposed to denormalised. For example, puzzles are not stored within an event, but in a separate collection. Thus, in order to find puzzles belonging to a certain event, this information has to be stored in either the puzzle or the event. The choice was made to store this One-To-Many relation on the side of the child (puzzle), since that would only require a single database query: finding all puzzles with a certain event ID, as opposed to two queries: getting the list of all puzzle IDs in an event, and then querying puzzles whose ID exists in this list.

5.2.4 Collections

This section describes the different collections in the MongoDB database. A visual overview is given in figure 5.3.

Events

The Events collection stores `EscapeEvent` objects, and is the cornerstone of the M.O.R.S.E 2.0 system. Nearly every document in the other collections have a field 'eventId', referring to a document in this collection.

[1] `http://jagi.github.io/meteor-astronomy/`

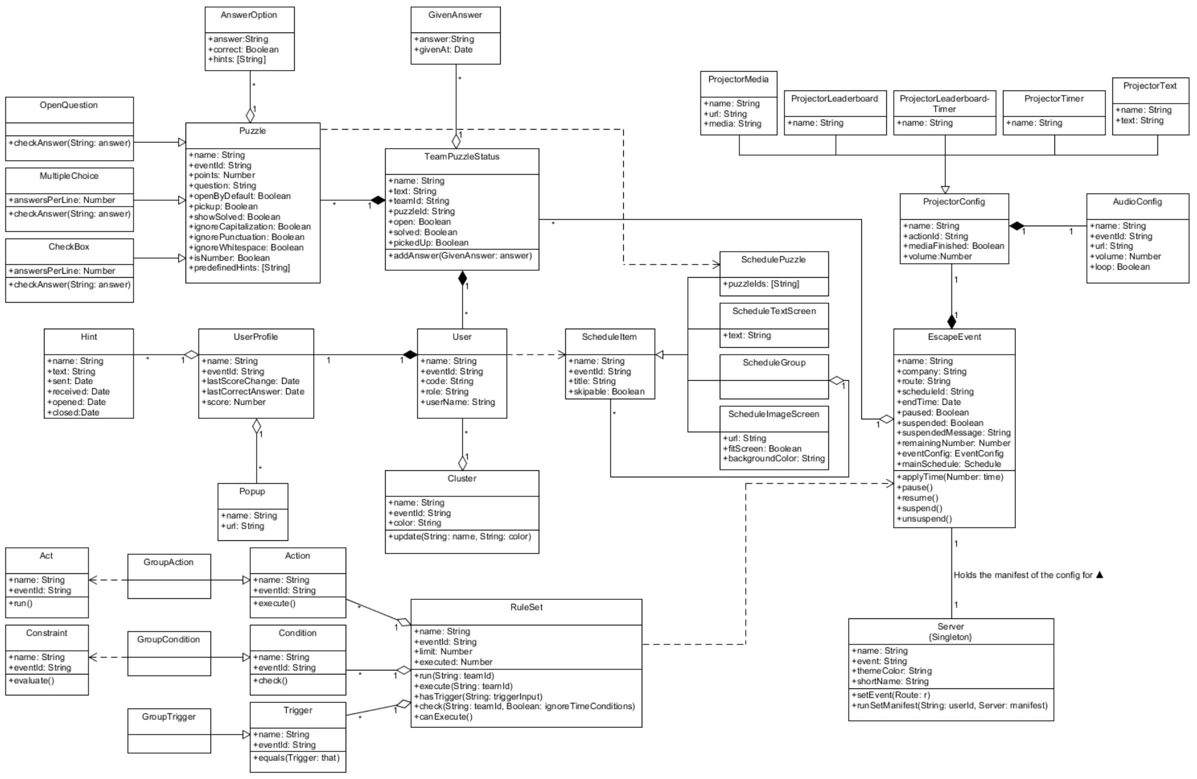


Figure 5.3: UML diagram of the entity classes of the application

An event document contains a lot of information about an event, such as a reference to the main schedule object and the configuration (see 5.6). Furthermore, it holds the current state of the projector, the timer and through which URL the players and staff members can connect to this event.

Users

The Users collection contains `User` objects and is a default collection of Meteor. It provides the application with log in and sign in functionality, as well as remembering users across browser sessions. In the application, a user can be a team or a staff member, depending on their ‘role’ field. This role is checked whenever a user attempts to call a function meant for (specific) staff members only.

A user object stores a team’s username, teamcode, teamname and score. Furthermore, it contains references to the ID of the cluster, the ID of the event it belongs to and an array of Popup and Hint objects.

Clusters

Clusters are used to group teams, such that a foreman oversees a single cluster. In a usual event, a cluster is a group of physically neighbouring teams, so the foreman is responsible for a specific section of the room where the event takes place.

A `Cluster` object has a name, a reference to the event and a colour. This colour helps distinguish the teams of the different clusters for the foremen and the project desk.

Schedules

The Schedules collection contains `ScheduleItem` objects. A `ScheduleItem` represents a stage in an event. An event has multiple stages, which are different screens the players see during the event. A `ScheduleItem` object has a title, and a boolean ‘skippable’, which implies users can skip the stage through rulesets. For an in-depth explanation of Schedules, see 5.3.

Furthermore, the `ScheduleItem` class has a number of subclasses:

- `SchedulePuzzle` - Represents a stage with puzzles. Has a field of puzzle IDs, referring to puzzle documents in the Puzzles collection.

- `ScheduleTextScreen` - A stage with text. Has a field for the text it shows to the players.
- `ScheduleImageScreen` - A stage with an image. Has a field for the URL of the image, whether to scale the image to the screen of the player, and a background colour, to fill the remaining space.
- `ScheduleGroup` - Has a list of ScheduleItem objects. To the players this has no function, but in the schedule editor or progress screen for the staff this is used to group `ScheduleItem`'s belonging to the same phase of the game (for example signing in, the first set of puzzles, the endgame).

Puzzles

The Puzzles collection stores objects of the class `Puzzle`. Puzzles have a title and subtitle, a question, a description, how many points they are worth and references to the `ScheduleItem` and an `EscapeEvent` they belong to. Furthermore, they come with some styling options, such as a background, a logo and colours for the title and subtitle.

Furthermore, a `Puzzle` object also stores some information for the foremen, such as predefined hints they can send and notes giving them extra information. They also store information about the puzzle within the events. This includes whether the puzzle requires extra information, available at the project desk, if the puzzle is considered locked or open by default and how the answers given by the players should be validated, see 5.4.

Finally, a puzzle also stores an array of `AnswerOption` objects. This class represents an answer to a puzzle. An `AnswerOption` has a string, representing the answer, a boolean whether the answer is correct or not and an optional string 'feedback', which the players will receive upon attempting a puzzle with this answer, if the answer is wrong. If the players give an answer which is not in this array, they will receive a default feedback string, either set in the puzzle object, or in the event configuration.

The `Puzzle` class by itself is meant to be abstract and had multiple subclasses, which are described in 5.4.

TeamPuzzleStatus

The TeamPuzzleStatus collection holds `TeamPuzzleStatus` objects. These objects represent the many-to-many relation between a `User` and a `Puzzle`. They store the current state of the puzzle for a team, i.e. whether it is locked or open, solved or unsolved and available at the project desk or picked up. They also store references to the team, puzzle, schedule item and event they belong to.

Furthermore, every answer given by the team is stored in the form of a `GivenAnswer` object array. These objects store the answer and the time the answer was given. This information is used by the foremen to determine which teams need their help.

Rulesets

The RuleSets collection contains `Ruleset` objects. Rulesets have a name for in the UI, a reference to the event they belong to, a limit on how many times they can be run, and a counter for how many times it already has been executed. Naturally, a ruleset can only be run if the counter is less than the limit.

A ruleset also has three arrays, triggers, conditions and actions, storing `Trigger`, `Condition` and `Action` objects respectively. The structure of these classes is extensively documented in 5.7.1.

Server

The purpose of the Server collection is to store a single document providing server-wide settings. Its main function is to store properties belonging to the 'manifest.json' file the tablets need, in order to run the app as a Progressive Web App (PWA).

If this PWA is started on tablet, it will redirect the players to the event defined in this document.

5.3 Schedule

The schedule is the combination of different phases which make up the skeleton of the game event. Every item has a title, an event ID to link it to the event it belongs to, and a boolean 'skippable' which states whether or not the schedule item can be skipped. Skippable can be used in combinations with rulesets to skip stages.

Schedule items come in four different specialisations: The schedule group, which contains a list of IDs consisting of the different kinds of schedule item. The schedule text screen, which has a string of text. The schedule image screen, which is similar to the schedule text screen, but has a URL of a image instead of a string. Finally, there is the schedule puzzle, which is a schedule item which has a list of puzzle IDs. All schedule items are part of one schedule group, which is the main schedule in an event. Through rulesets different schedule items are linked, to transition from one into the other.

5.3.1 Schedule editor

Schedules are easily created using the schedule editor. Which allows the game-designer to add and customise any of the schedule items for use, such as puzzles and text screens.

In the left menu a schedule item can be selected, which will be opened in the editor on the right side. Schedule items can be moved using drag and drop within the left menu. Dropping a schedule item into its new position will cause a server method to be called, which will move the schedule item in the database, and thanks to Meteor, this update is then immediately reflected in the front-end. In the left menu, there is also a ‘new’ button, which is used to add new schedule items.

Because schedule items, including groups, can be nested within groups, a schedule item is a single Angular component, which includes its children recursively if the schedule item is a group or puzzle screen and has children. This way every level of nesting is supported. There is a slight issue with dragging items in and out of groups, as Angular is not aware of the nested ‘drop locations’, but this was mostly resolved by applying some boundary checks on the boundaries of the drop locations.

In the editor the properties of the schedule item are listed. In order to edit them they have to be clicked explicitly. This was done to ensure that schedule items are not changed by accident. Only a single property can be edited at a time. Clicking the save button or selecting another field to edit will save only the property that is currently being edited.

In order to reduce the complexity and duplication of the Angular code responsible for the editing of the fields, all the logic was extracted into a series of in-line Angular templates, which could then be included and passed certain variables, such as which field in the database should be displayed or edited when it is clicked.

5.4 Puzzle

Puzzles form the core of the system. Teams have to solve them in order to earn points and progress through the game. Puzzles come in three base forms: the open question, the multiple choice and the checkbox:

- Open question - The UI will give the players a text input field

- Multiple choice - The UI will give the players a series of buttons representing the answers from the answers array. Only one answer can be given at a time. This serves no purpose as an actual puzzle, since every answer can be attempted and thus this puzzle can be brute-forced. However, it can be used to ask the players to choose an option from a predefined list, which could influence the control flow of the game.

- Checkbox - The UI will give the players a series of buttons representing the answers from the answers array. Multiple answers can be selected at once, and all correct answers must be checked, and all incorrect answers must be unchecked, for the puzzle to be considered solved.

The main part of a puzzle is the the question, the number of points that the puzzle is worth and a list of answer options. An answer option consists of an answer, a boolean indicating if this answer is correct and, if the answer option is incorrect, a hint can be linked to that specific wrong answer to provide the teams with feedback. Next to these elements, a puzzle also contains a schedule ID and event ID to link the puzzle to a specific schedule item and event. Each puzzle also has a title and a optional subtitle. If chosen, these titles can be given a colour. Further cosmetic options include a background image and a logo. A description can be provided to give the user extra information and notes can be added to provide extra information to the foreman. The boolean ‘open by default’ determines whether the puzzle is automatically available, or if something needs to be done to unlock it. The boolean pickup indicates if a puzzle requires extra items from the project desk in order to solve it.

Puzzles also come with a number of functions that help determine how specific the given answer needs to be. Puzzles can check answers for capitalisation, white space and/or punctuation, or whether the answer is a number or not. This means “5,0” will be interpreted as both ‘5’ and ‘50’, and only one of these needs to match the actual answer of the puzzle. This choice was made because JavaScript uses periods as a decimal separator, but Dutch uses commas, while commas are also used as a thousands separator. These checks are specified in the puzzle editor.

5.4.1 Submitting an answer

In the UI, players can attempt to solve puzzles. If this happens, the front-end calls an RPC with the puzzle’s ID and the players’ answer. On the server, this answer is then compared to to answers stored for that puzzle. Of course, it is first verified that puzzle is actually in a solvable state. This means that the puzzle is open and delivered by the Project Desk, if applicable. Furthermore, the event should not be suspended.

There are three possibles outcomes of comparing the players’ answer to the defined answer:

- The given answer exists in the array of defined answers, and the defined answer is marked as correct. The puzzle is then marked as solved, and teams can move on to the next puzzle, if one is available. Players will be able to see when a puzzle is solved, but they will not see the answer they gave to solve it.
- The given answer does not exist in the array of defined answers. The answer is wrong and the players are given a message telling them the puzzle is not correct.
- The given does exist in the array of defined answers, but it is marked is incorrect. The players will receive a message custom message, defined specifically for the given answer.

In any case, the given answer is stored, so the foreman can see if the team is getting close to solving the puzzle. Furthermore, if there is a ruleset with a trigger for this puzzle, that ruleset will be evaluated and executed (see 5.7.1). A visual overview of attempting a puzzle is given in figure 5.4

5.4.2 Puzzle editor

The puzzle editor is situated within the schedule editor, since puzzles are conceptually schedule items that belong to puzzle screens. The editing process for puzzles is the exact same other schedule items, which is outlined in 5.3.1.

Puzzles also appear in the left menu of the schedule editor. Since puzzles can only exist within puzzle screens, they can be dragged between puzzle screens, but not any other type of schedule item. When clicking ‘new’, by default the new Puzzle option is disabled until you select a puzzle screen. Otherwise the system would not know to which puzzle screen the new puzzle belongs to.

5.5 Users

During an event, users interact with the created system. A user can have one of 6 roles. The ‘Team’ role is used for the players, while the other 5 are used for the staff. These roles consists of the general staff role, and separate roles for a foreman, project desk member, game-host and admin. A visual overview is given by figure 5.5

A team can consist of multiple individuals, but can only interact with the game as one team. Multiple teams can be grouped together in a cluster. A cluster consists of a name, an event ID to link it to an event and a colour to distinguish it from other clusters. A team stores its cluster ID to link it to the cluster it belongs to.

The big part of a team consists of a user-profile. The core of the user-profile consists of an event and schedule ID to link it to an event and a specific schedule item, a code, which is used to log in to an event, a team name, which the team enters after logging in and a score attribute, which keeps track of the team’s earned points. A user-profile also has two date objects, ‘last correct answer provided’ and ‘last score change’. These fields are used to calculate the time since those dates to keep track of the progress of the team. Two arrays, hints and pop-ups, are also part of the user-profile. These arrays contain text hints and images to assist a team with a puzzle.

The staff users only have a small user profile, containing their login code. This login code can only be used in the admin login page. After a staff member logs in, he or she is redirected to the page

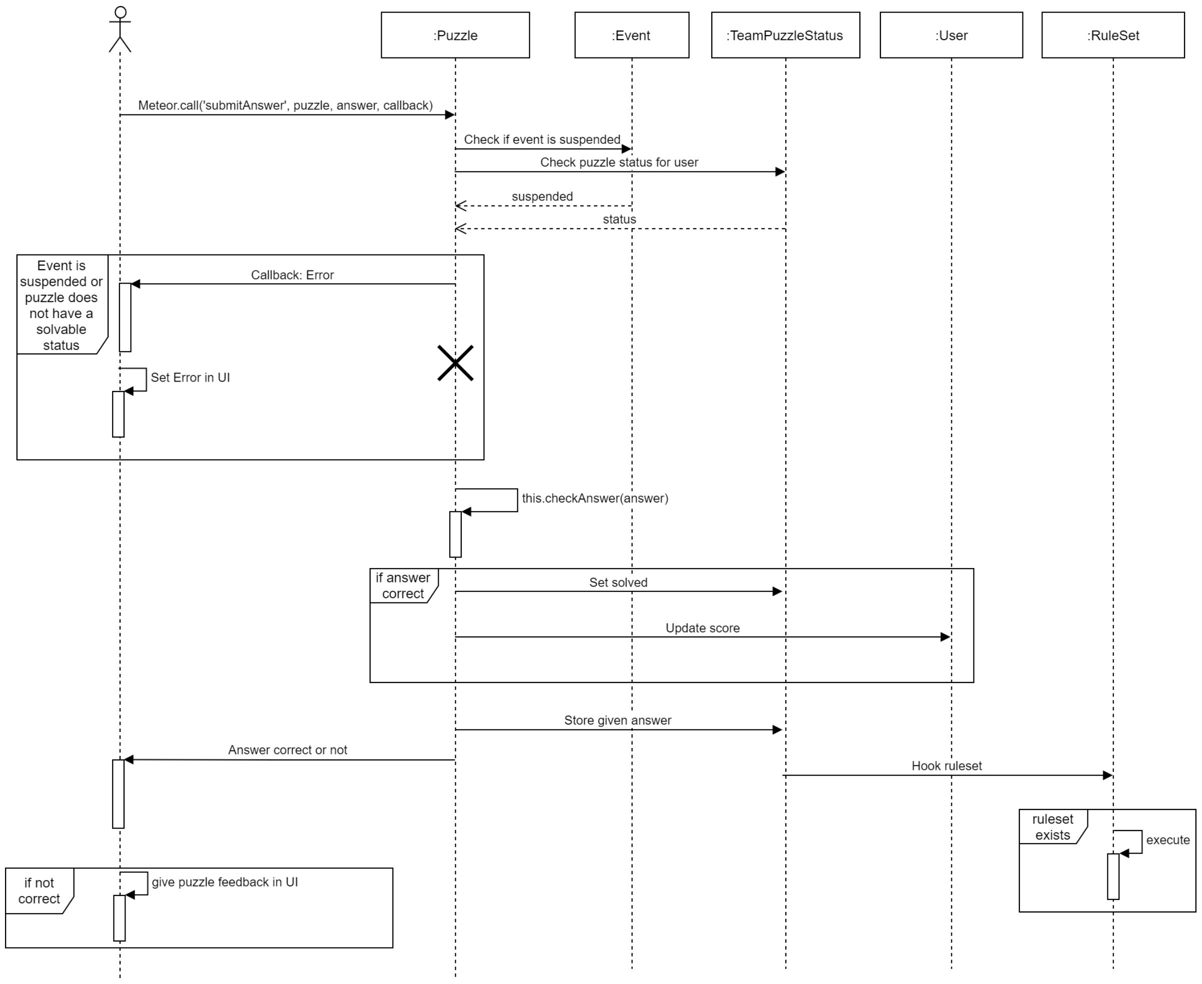
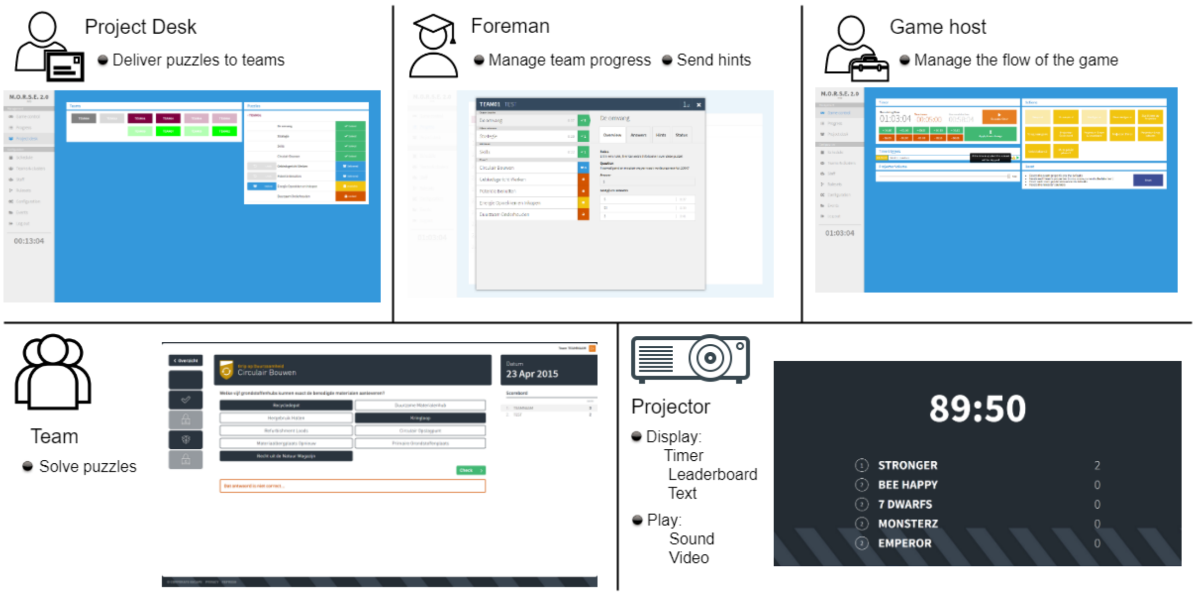


Figure 5.4: A UML sequence diagram of submitting a puzzle.



89:50

STRONGER	2
BEE HAPPY	0
7 DWARFS	0
MONSTERZ	0
EMPEROR	0

Figure 5.5: Overview of the different user roles in the system

corresponding to the staff role. Users with the 'Admin' role can access any page in the administrator environment and can manage multiple events. Users with the 'Game-Host' role can access most pages and can only manage the event they were created for. The 'Project Desk' role has access to the progress and the project desk screens, but can only interact with the project desk screen. The 'Foreman' role can only access and interact with the progress screen. The general 'Staff' role can see the progress screen, but cannot interact with it. All the screens in the administrator environment can be found in section 7.1.

A team's progress is tracked in a `TeamPuzzleStatus` object. The core of a `TeamPuzzleStatus` is the link between a team and a puzzle, since it keeps track of the progress of a specific team for a specific puzzle. When a team is created, the `TeamPuzzleStatus` for all puzzles are also created. A `TeamPuzzleStatus` keeps track if a team picked up the extra materials needed for a puzzle, if the puzzle can be attempted and if it solved. These are represented with a boolean. Furthermore, the given answers of a team are also stored in the form of an object. This object contains the answer given in the form of a string and the time it was given in the form of a date object. The `TeamPuzzleStatus` can contain another date object, indicating when the open field was changed to open. The properties 'solved', 'pickedup' and 'open' determine the way a team sees the puzzle. Each combination of these properties results in different screens, the logic of this can be found in figure 4.1. The given answers are used in the progress screen, here the date is used to compute the time since an answers was given and the answer is used to help the foreman monitor the teams progress.

5.6 Configuration

Because every escape event is tailored towards the company for which it is being held, every event looks and feels different. For this reason, there is an extensive configuration which allows the designer of an event to set up an event to their exact specifications.

Some examples of configurable settings are: the overall theme colour and font of the player screen, and how the timer should be displayed to the players and on the projector. There is even a configurable option for every string in the player screen, which allows the entire game to be played in a different language, or using different terminology (e.g. calling puzzles 'Projects' or 'Skills').

There are also settings which affect events in ways unseen to the players. For example, the option to only allow team codes that are defined upfront are allowed to sign in, or every team code is allowed to sign in. Furthermore, there are some settings which affect the foremen, such as after how many minutes they should get a warning, when a team they oversee have not solved a puzzle.

5.6.1 Implementation

The configuration is divided into separate categories, which contain single configurable options. These options are stored in a subdocument in the event document. This way, every client has access to them.

Some settings change the logic of the application, while others change the application's design. For these settings CSS variables[2] are defined in the front-end, which are overwritten by the values stored in the configuration.

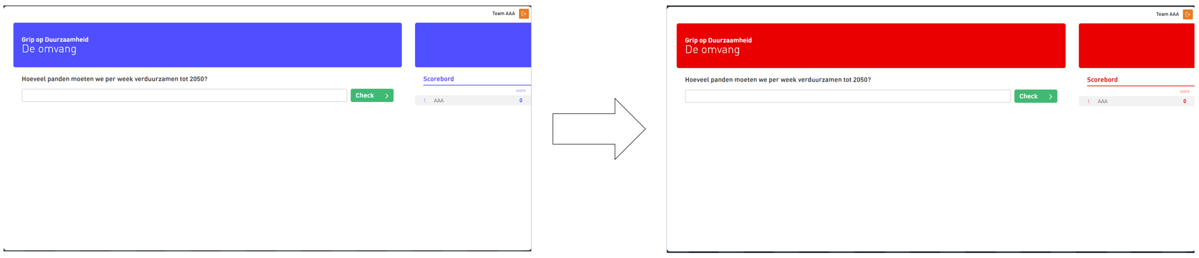


Figure 5.6: An example of a configurable setting

5.6.2 Editing the configuration

There are two ways to edit the configuration. The first one is through rulesets, which are described in 5.7.1. The second way is through the configuration editor in the front-end.

Just like the configuration in the back-end, the configuration editor is also divided in categories of options. The choice of using categories was made in order to prevent the editor from displaying to many options at once. By separating the options into groups, which can only be displayed one at a time, there is much more overview in the editor. Furthermore, the purpose and function of an option becomes clearer in a category of related options.

When the configuration editor is loaded, it stores the state of the configuration at that time. Because of Meteor's "reactive" functionality, it could happen that the configuration changes while it is being edited. If this happens, the editor will display a warning that the saved state of the configuration is no longer valid, and that the editor will need to be refreshed. This means that the changes made locally in the editor will be discarded, but the probability of two people editing the configuration at the same time is low enough for this not to be a problem.

5.7 Rulesets

In order to prevent hard coding the logic of an event, the M.O.R.S.E 2.0 system needed to be as flexible as possible. To this end, a ruleset structure was built into the system. The flexibility of the ruleset structure stems from the fact that it provides a lot of utility to the game designer. For example, it allows the game designer to change the game structure without needing to change the system's code. Rulesets are great for game designers to direct the flow of the game.

Rulesets consists of a list of 'actions', which are made up from smaller 'acts', such as making all teams in a cluster go from one puzzle to another puzzle or playing a video on the projector. A ruleset also has a list of 'conditions', a condition being a combination of restraints which need to be true in order for the action to occur, for example, a team must have given a certain answer to a certain puzzle or a puzzle needs to be solved before an action may occur. Finally, a ruleset has one or more 'triggers', triggers are similar to conditions in that they are a constraint that need to be true in order for the action to occur, the difference being that a trigger actually causes the ruleset to check on the conditions, performing the action if they are true. Triggers come in many shapes and forms, for example, they can be based on time, manually activated with a button or fire when teams (attempt to) solve a puzzle.

An overview of all triggers, conditions and actions is given in figure 5.7

5.7.1 Triggers

There are several triggers which allow the game designer to have control over the game. In the back-end, a ruleset will evaluate the conditions if one of his triggers is called. This means that the triggers of

[2]https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties

a ruleset are equivalent to a boolean *OR* operator, that determines whether the ruleset should check its conditions and run its actions.

Most of the triggers get triggered by calling the hook function at which the trigger should react. For example for a `PuzzleStatusTrigger`, which checks whether the status of a puzzle updates, this is once the puzzle status gets updated. It calls the hook function with all the current viable information, such as a reference to the event, user and the puzzle. The hook function checks, if there is a trigger in a ruleset to which the hook applies. If trigger and the hook match, the ruleset will be executed.

There are two types of triggers, namely triggers based around a team and triggers that are not. A trigger based on a team is for instance a `PuzzleStatusTrigger`, in which a team ID has to be given while calling the hook. A trigger which is not based around a team cannot have conditions and actions in which a particular team or cluster is targeted, an example of such a trigger is the `ManualTrigger`, which is triggered when the game host presses a button in his control panel.

Regarding triggers there are two special triggers, namely the `TimeTrigger` and the `WebhookTrigger`, because they are more complex and their structure is different from the other triggers, they will be explained separately in the following two paragraphs.

TimeTrigger

For a game designer it is useful to have the ability to execute rulesets at a certain point in time. The challenge that came with this is to create a time trigger in such a way that it would not create unnecessary extra server load.

One of the more naive ways to solve this problem is to keep track of all time triggers and check every second whether they would be triggered. However, this would create a job which the server has to execute every second. Another possible way of implementing this was by using cronjobs [3]. Cronjobs have a way of executing jobs at a certain time (interval), already have a structure for the timing, and does not add a recurring job to the server. However, cronjobs only have the ability to trigger on dates, hours and minutes, and not on seconds. Unfortunately this make cronjobs unsuitable to solve the challenge of triggering rulesets on a specific time.

Therefore the choice was made to built a system for the time triggers from scratch, which is based on a list of timeouts created from time triggers. Every time trigger gets converted to a timeout, which has as a callback the execution of the associated ruleset. For this a conversion from the current time to the event time is made, to determine the length of the timeout. Because the timeouts are running in parallel, it is very easy to have multiple time triggers. However, an issue occurs when there is a change in time, for instance when the time gets paused, the timeouts should not occur and the rulesets should not be executed, since the time is paused. This creates an additional challenge, which is dealt with by clearing all the timeouts when the game is paused an adding all of them again when the game is resumed or when the time is updated. The timeout methods used are the timeout methods from Meteor, which allows timeouts to be cancelled after setting them, making them very suitable for this problem.

When the time is updated and skips over the time trigger, the associated ruleset will not be triggered. When the game designer skips over a time trigger, (s)he will get a visual indication which warns them that they are skipping over it. At the same place as the warning, the game designer can manually trigger the time trigger in case skipping the time trigger was not wanted. Warning and letting a game host manually trigger a skipped time trigger was more desirable than simply executing them in case of a time change, since emphasising the execution of a ruleset for the game host is hard to do after the fact, but not before it.

The performance of the time triggers is good. This can be tested by creating a ruleset with a time trigger at 10 seconds and an action to increase the time with 1 second. This will then go on and loop infinitely. This shows that the implementation of time triggers is sufficient for the use case.

WebhookTrigger

The game designer also has an option to use a `WebhookTrigger`. This allows for dynamic ruleset triggering from an external source. This is especially useful, because some games consist of separate systems which will gain benefit from sharing data to each other. For example a button in a separate web page can be pressed, which caused a the execution of a ruleset in the M.O.R.S.E 2.0 system.

[3] `http://www.unixgeeks.org/security/newbie/unix/cron-1.html`

With webhooks, the M.O.R.S.E 2.0 system is capable of communicating with the Internet of Things, which brings great new capabilties for events. For example, the scanning of a physical QR code to solve a puzzle. Another use case is building a puzzle whose complexity goes beyond the types of puzzles the application supports, such as a regular open or multiple choice question. This puzzle could then take place in another web environment, and communicate with the main system through webhooks.

A `WebhookTrigger` has to have a specified endpoint, to which the it will react. The first part of the URL is always defined (/webhook/), what comes after is for the game designer to choose.

The game designer also has the option to only let the `WebhookTrigger` trigger when a key-value pair is given, which allows the game designer to have some structure in the way in which they manage the different endpoints. For example, for changing a config colour with a `WebhookTrigger`, the admin can create the route '/webhook/color' and create a ruleset for each colour which has to be accepted with as key value for the data of the webhook call 'color: <color>'.

Another option the game designer has is to require a `WebhookTrigger` to have a team ID. This means the call to the webhook should have a field called 'teamId' in which a valid team ID has to be given (this is checked before the webhook triggers). Having this in a `WebhookTrigger` allows the game designer to use conditions and actions which are in context of a team (such as `TeamPuzzleStatusAct`). The only problem is that the system that calls the webhook has to be aware what the team ID in the M.O.R.S.E 2.0 system is. For this a `WebhookAction` has been created (which can send the team ID) which will be explained later on.

With the addition of webhooks the M.O.R.S.E 2.0 system has a lot more possibilities and can be used in combination with other systems (or parts thereof), which make the system as a whole even more flexible.

5.7.2 Conditions

Once a ruleset is triggered, the list of conditions of the ruleset will be evaluated. Before a ruleset can execute its actions all the conditions have to evaluate to true, which means that there is an *AND*-structure.

The conditions are split into two groups, namely team-based conditions, named `GroupCondition` and a group of more general conditions which are checked game-based.

Game-based conditions

The game-based conditions all have a function which evaluates the condition to either true or false. The data which is needed for the game-based conditions is always retrieved from the event and therefore a condition is always tied to an event, which is the event of the ruleset.

Team-based conditions

Apart from game-based conditions there are also team conditions, which are applicable to a team. Only the children of a `GroupCondition` can be instantiated, which are `EveryoneCondition`, `ClusterCondition` and `TeamCondition`. These group-based conditions are checked for respectively everyone, a cluster and a team. A GroupCondition contains a 'constraint', which is similar to a condition, but different in the fact that they are interchangeable and can be used in every type of group condition. Constraints follow the Strategy design pattern, so a constraint is an abstract type and has several children that check a property for a team. For instance a `PuzzleStatusConstraint`, which checks whether a puzzle has a certain status for a team. A special boolean is added to the `EveryoneCondition` and the `ClusterCondition`, namely whether the evaluation of the group should be an AND or an OR evaluation, determining whether at least one in the group should satisfy the condition or whether all teams in the group should satisfy the condition.

5.7.3 Actions

If a ruleset is triggered and the conditions evaluate to true, all the actions of the ruleset will be executed in the order of insertion. An execute can also trigger another ruleset or itself, this way loops can be created, which the user needs to be vigilant of.

There are actions which target a group, named `GroupAction` and actions that target the game and its state. Besides those two types of actions there is also a webhook action and a more complex action to open puzzles in a puzzle stage. Both of those will be described after the game-based- and team-based actions.

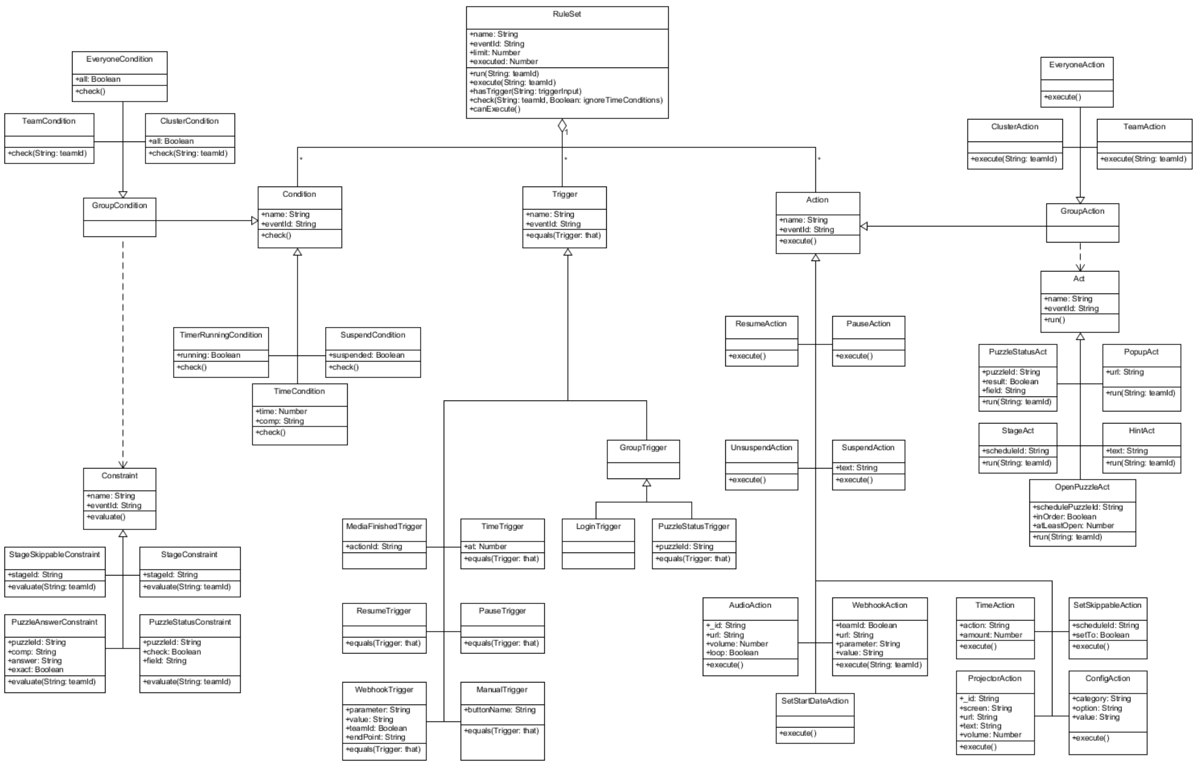


Figure 5.7: An overview of all the possible triggers, conditions and actions

Game-based actions

The game-based actions do not require a team context and are applied to the game and its state. An example of a game-based action is the `TimerAction` which can be used to update or set the timer. Using those actions the game state can be automatically altered on certain moments.

WebhookAction One of the more powerful actions is the `WebhookAction`. This allows to send a HTTP call to an URL. Because of this the M.O.R.S.E 2.0 system to communicate with external sources and link up with other components of other systems. A new link with another system can therefor be made without changes in the code. The WebhookAction is setup dynamically, it needs an URL and optionally extra data. The game designer can choose to add a key-value pair, which are always sent while calling the webhook. If the trigger has a team-context, the game designer could also choose to send the team ID in the call, such that teams can be tracked through various systems. Together with the `WebhookTrigger` data can be sent and retrieved with webhooks.

Team-based actions

The team-based actions are actions that apply to one or more teams. The actions do influence something that is team specific. Just like with the conditions, there are three team-based actions, namely a `TeamAction`, a `ClusterAction` and an `EveryoneAction`. Group-based actions are executed for everyone in the selected group (there is not an extra parameter like there is for `GroupCondition`) so therefore the execution is *AND*.

OpenPuzzleAct A more advanced team-based action is an action which contains an `OpenPuzzleAct`. For some events, there can be a phase where a set amount of puzzles needs to be open. This means that a once a team solves a puzzle, another puzzle should be unlocked. For that case the `OpenPuzzleAct` can open a defined amount of puzzles either in insertion order or in random order. Every time this Act gets called it will ensure that in the schedule there are at least the defined amount of puzzles open.

5.7.4 Ruleset editor

To give the capability to create and edit rulesets, game designers have access to the ruleset editor. In this editor, game designers can visually create a ruleset, add triggers, conditions and actions and configure the parameters for each of these.

To do this, the ruleset editor has a special data structure in the front-end to specify the possible triggers, conditions and actions including the configurable parameters. This data structure consists of three parts: a list of templates for all types of rules, a list of conversion functions to convert the template to a back-end rule and a list of conversion functions to convert existing rules from the database to editable rules in the front-end.

Templates

The list of templates defines how each type of rule is shown in the editor. This consists of a combination of strings and objects, where these objects represent parameters of the rule. The example below shows how this template is constructed for the action to update the time.

```
timeUpdateAction: [
    {
        name: 'change',
        type: 'dropdown',
        options: {
            increase: 'increase',
            decrease: 'decrease',
        },
        value: 'increase',
    },
    'the timer with',
    { name: 'minutes', type: 'number', min: 0, value: 0 },
    'minutes and',
    { name: 'seconds', type: 'number', min: 0, max: 59, value: 0 },
    'seconds',
],
```

This template is shown to the user as “[Increase] the timer with [0] minutes and [0] seconds”, where the user can alter the values of the dropdown menu and the number fields. Like the dropdown menu and number types, there is also a text type and puzzle, schedule and answer dropdowns as a way to reference other parts of the game.

Conversion from editable to rule

Because the data structure of rulesets in the back-end and database is completely different from the structure to edit them, a conversion between the two is needed. When a user saves a ruleset in the ruleset editor, each rule is converted to a rule in the back-end. This is done using a list of conversions. Based on the chosen template, the back-end class is determined. Attributes which have the same name are automatically filled in in the corresponding field, after which a predefined conversion function remaps the other attributes and sets remaining attributes.

For example, the template of the `timeUpdateAction` above is converted to a `TimeAction` in the back-end. This `TimeAction` has two attributes: an action, in this case *update*, and an amount of seconds. This is calculated based on the minutes and seconds and made negative if the change is *decrease*.

The resulting object is sent to a back-end method which saves the ruleset in the database.

Conversion from rule to editable

When the user opens an existing rule, the object from the database has to be converted to an editable object using the templates to be editable in the front-end. For this, the third list is used. This list maps back-end rule types to templates in the front-end and converts the attributes on the database objects to the parameters in the templates.

By using this data structure in the front-end, the ruleset editor is flexible enough to be extended with new types of rulesets and allows to make a distinction between the way rulesets are saved in the database and are presented to the user in the editor.

6 Testing

Throughout the project, the created system was tested in several different ways. From automated unit tests to testing the product during an actual escape event. This chapter will go over all the different ways the system was tested during its creation and describe how a high quality of code was maintained.

6.1 Unit testing

Throughout the development of the project, the code was tested via unit testing. The Mocha test framework was used for these unit tests, because it was recommended to be used in combination with Meteor [20]. To make sure that enough of the back-end code was tested, custom CodeCov[1] settings were created. At least 90% of the back-end needed to be tested at all times and this also applied to newly written code. The continuous integration ran the created unit tests before each pull request and blocked merging if one or multiple tests failed. In the end a coverage of 99% of the back-end code was reached.

6.2 Manual testing

The front-end code was tested by manual testing. Automated front-end tests has been attempted, but this was not feasible as it was not initially supported by the required dependencies. After discussing it with the coach, this idea was dropped completely. Furthermore, the 'hot reload' feature of Meteor allowed for quick manual testing of the front-end methods. The front-end code was tested regularly in two ways. During demonstrations to the client and coach and before each pull-request regarding front-end additions or changes. Before each pull-request, the changes were deployed and the reviewers checked the front-end of the system on a laptop or tablet to make sure the system still worked as intended.

Figure 6.1: An overview of the event [21]

[1]http://codecov.io/

6.3 Real-world testing

At the start of the project, it was known that there would be a few opportunities to test the developed system during an event of Escape Event. The aim was to have the system ready for the event on the 13th of June.

Before the system could be used during such an event, the client needed to approve the new system. This was done, to make sure that the system was capable of playing the event and had no unintended behaviour. After the system was approved, it was tested with a stress-test. During this test, 20 tablets interacted with the system at the same time. Actions like logging in together, spamming random buttons and answer questions at the same time where tested. The last test before the event, was demonstrating the system to the employees of Escape Event.

Figure 6.2: A team interacting with the system during the event [21]

After these tests were a success and some improvements were made, the system was ready for the event. During the event on the 13th of June, 75 participants interacted with the client side of the system and 5 employees from Escape Event interacted with the admin environment. There was a small issue, regarding a mismatch between a puzzle and its solution, but this was not system related. This could have been solved by having a schedule editor, but this was not realised yet. To solve the mismatch, the answer field of the puzzle document was updated in the database directly, and this update immediately took effect thanks to Meteor's reactive database queries.

Overall, the system performed as expected and a lot of feedback was gathered from this testing method. Due to the use during a real event, some small issues where found, which needed to be solved. The next day, each employee from Escape Event reflected on their use of the system and which additional features and improvements they desired. Generally, they were very positive about the system and provided some great feedback. The employees reacted positively to the newly deigned progress screen as it gave them a good overview. However, they would have liked to see some extra information in the foreman popup screen in the form of the question and some additional notes. The feedback list from the event en the feedback session can be found in appendix E.

6.4 Static analysis

6.4.1 ESLint

ESLint, a static code analysis tool, was used to improve the quality of the code. ESLint pointed out inconsistencies in the code-base, such as lines which were too long, or unused imports. A default configuration, recommended in the Meteor guide, was used and some extra rules were added throughout the project to ensure a higher code quality.

6.4.2 SIG

After 6 weeks, the master branch at that time was uploaded to the Software Improvement Group (SIG). SIG rated the code-base on code quality. The result of the first feedback is shown in appendix C. The code-base was rated with a score of 3.7 stars on a scale from 1 to 5. The biggest points of improvement were large methods and duplicate code. After receiving this feedback, the old code-base was updated, by removing duplicate code and splitting up large methods into multiple smaller ones. Also, a new rule was added to the ESLint configuration, namely that methods should not be longer than 20 lines. In the 9th week, a new upload was done to SIG, but at the time of writing there is no new feedback yet.

7 Final product

This chapter presents the final product as a solution to our Bachelor End Project (BEP). First, a detailed walkthrough of every screen in the application is given. Then, a comparison is made between the final product, and the original M.O.R.S.E system. Finally, our *MoSCoW* from chapter 3 is revisited and we reflect on how our applications meets the defined requirements. Finally, this chapter concludes with a section on the ethical implications of our application.

7.1 Walkthrough

In this section a walkthrough of the system will be given. It will go over each possibility every type of user has in the system and how these possibilities are used to control the game. For this section general knowledge about escape events is required, which is given in chapters 1, 2 and 3.

7.1.1 Player

Login

For the player there is a separate login page, displayed in figure 7.1. This page displays a number of input boxes, the amount of input boxes is equal to the length of team codes for the event. Once the player fills in their team code, and click on continue, the player then gets to fill in a team name. After that it depends on the configuration of the event what happens next. The separate possibilities will be explained in the next sections.

Base layout

This part will explain the base layout parts, which will be always visible after logging in. This consists of at the top right a logout button for the player to logout and at the bottom three different links, one to the website of the Corporate Escape, one to the privacy statement of Escape Event and one link that allows to refresh. The base layout is depicted in figures 7.2 and 7.5.

Leaderboard

At the right side of the player screen a leaderboard is visible, the scores of the team itself and other teams are displayed in order from highest to lowest. If two teams have the same score, they will have the rank, but the team that reached that amount of points first will show higher on the leaderboard. If there a lot of teams, the leaderboard has a scrollbar, in order to see all teams. A team's own entry in the leaderboard will be highlighted, so players can easily find their own score.

Timer

If a timer type has been set for the event, the players are able to see a timer at the top right of the screen. Depending on the setting this will be a date or a time. The time is synchronised with the time of the server across all the different devices.

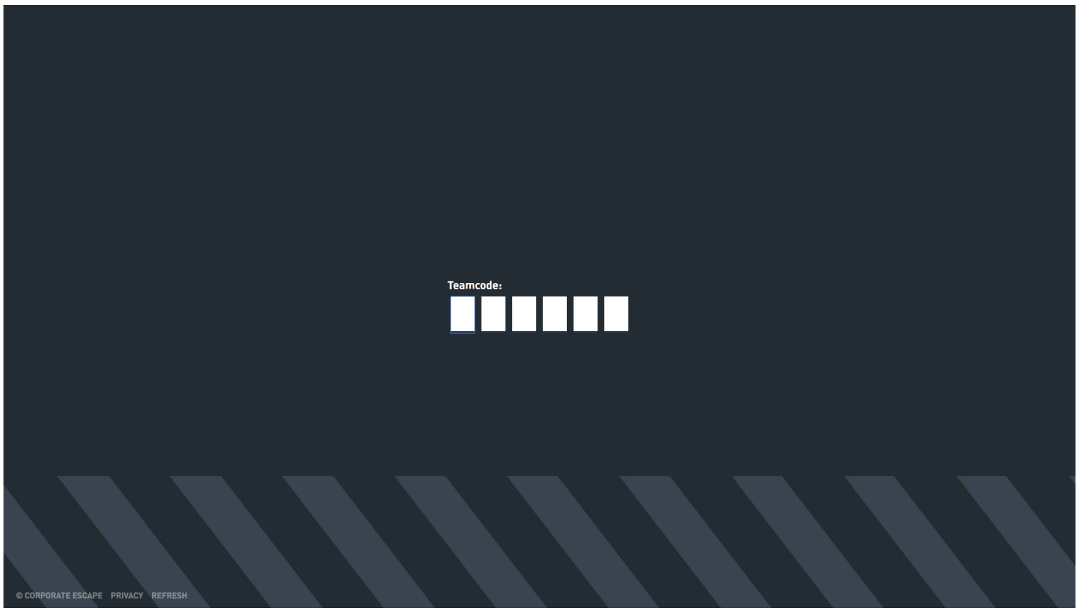
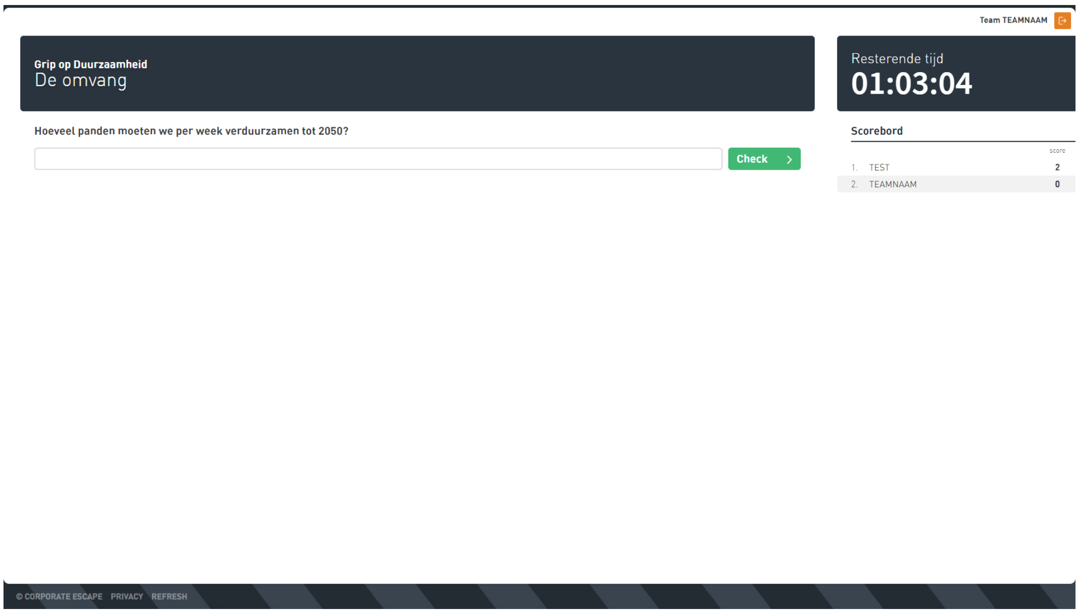


Figure 7.1: The player login screen



Grip op Duurzaamheid
De omvang

Hoeveel panden moeten we per week verduurzamen tot 2050?

Resterende tijd
01:03:04

Scorebord

		score
1.	TEST	2
2.	TEAMNAAM	0

Figure 7.2: Answering an open question puzzle

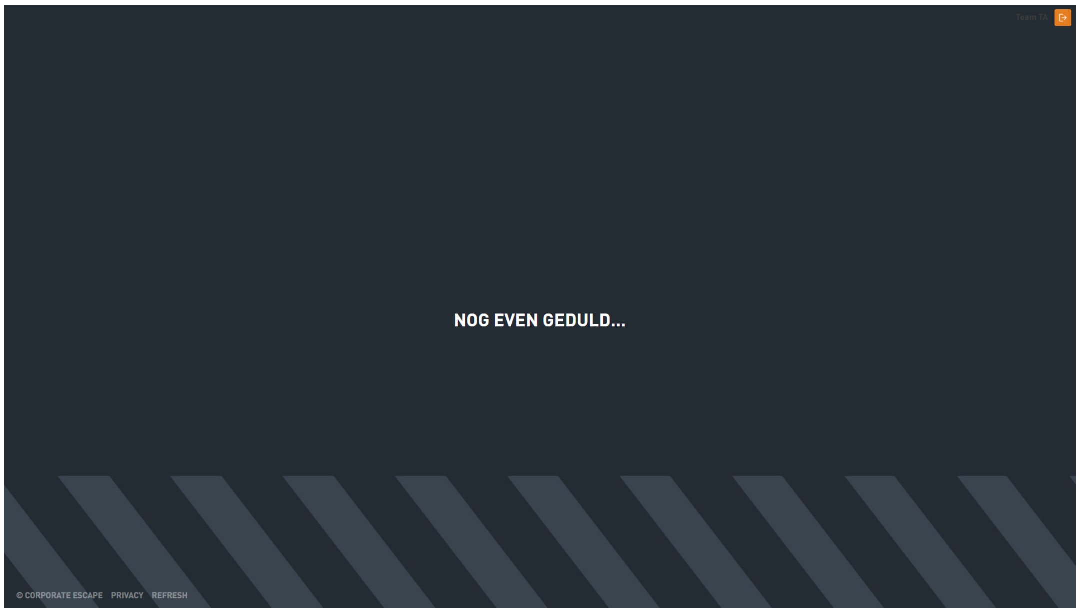


Figure 7.3: A suspended game

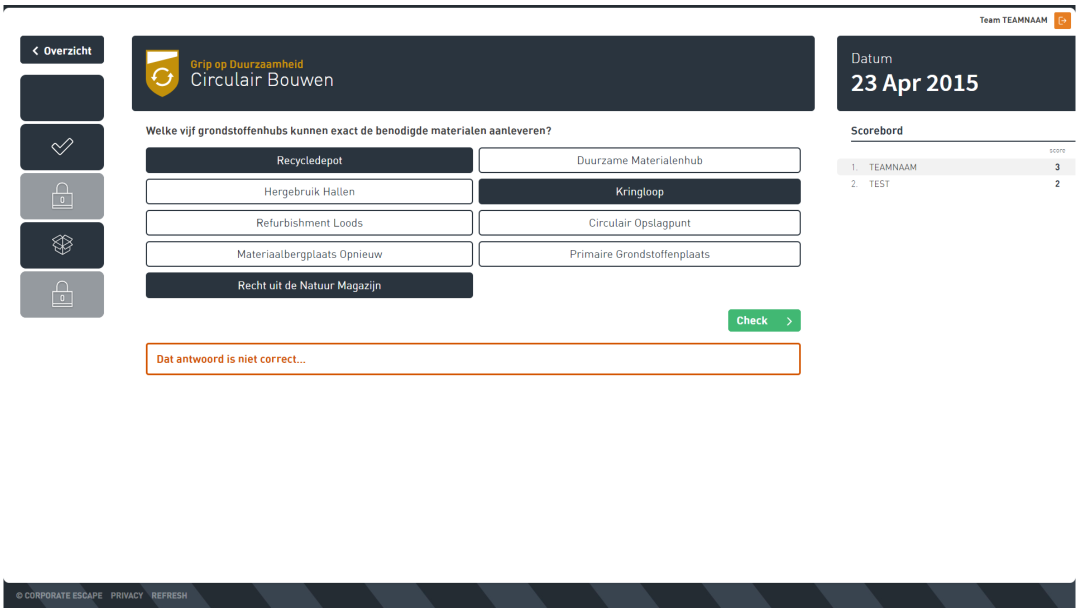
Simple screens
One of the possible schedule item types the player can be in is a simple screen type. This means the player gets to see an image or a text with which they can not interact. This is mostly used for when a video on the projector is playing or when interaction with the tablet is not required.

Also a text screen is displayed when a game-host or admin suspends the game, possibly with a message. In this way the suspension screen and the text screen act the same. This is depicted by figure 7.3

Puzzle screen
If the player is in a schedule item with a puzzle, the player will see the puzzle screen.. This screen has different looks based on the question. There are two separate question types, namely open questions (figure 7.2) and multiple choice questions (figure 7.4). Multiple choice questions have two variants, one of which only has the option to give one option as answer and the other that allows a combination of answers. If a wrong answer is entered a message will be shown in red indicating that the answer was not correct. If the answer is correct is also shown, but in most cases the player will be redirected to the next puzzle by rulesets. If the puzzle has predefined wrong answers and the players enters one of the wrong answers, players can receive an alternative feedback message, which can be set for the particular answer.

The design of the puzzle screen is defined in the puzzle editor. The title, the subtitle, description, question are all based on the question. As well as an optional logo for the question, this will appear left of the title. By default the block above the question is in the colour of the main theme, but if a background image is defined for a puzzle, that block will be filled with that image.

Group puzzle screen
The player can also be in a phase which consists of multiple puzzles, in that case a menu with blocks representing the puzzles will appear, as shown in figure 7.5. Each question has its own block, showing their background image and title, as well as their state indicated with an icon. All blocks are clickable, however if the player selects a puzzle which is already solved, should be picked up first or is still locked he/she will see a screen explaining why the player can not attempt that question. The puzzle screen itself is the same as the puzzle screen for a single question, except from the fact that a sidebar at the left will appear from which they can switch to the other puzzles or back to the overview page of all the puzzles. Also a CSS animation is added to the blocks in the overview, to give the user the impression that they should click the blocks.


Team TEAMNAAM

Overzicht

Grip op Duurzaamheid
Circulair Bouwen

Datum
23 Apr 2015

Welke vijf grondstoffenhubs kunnen exact de benodigde materialen aanleveren?

Recycledepot	Duurzame Materialenhub
Hergebruik Hallen	Kringloop
Refurbishment Loods	Circulair Opslagpunt
Materiaalbergplaats Opnieuw	Primaire Grondstoffenplaats
Recht uit de Natuur Magazijn	

Check

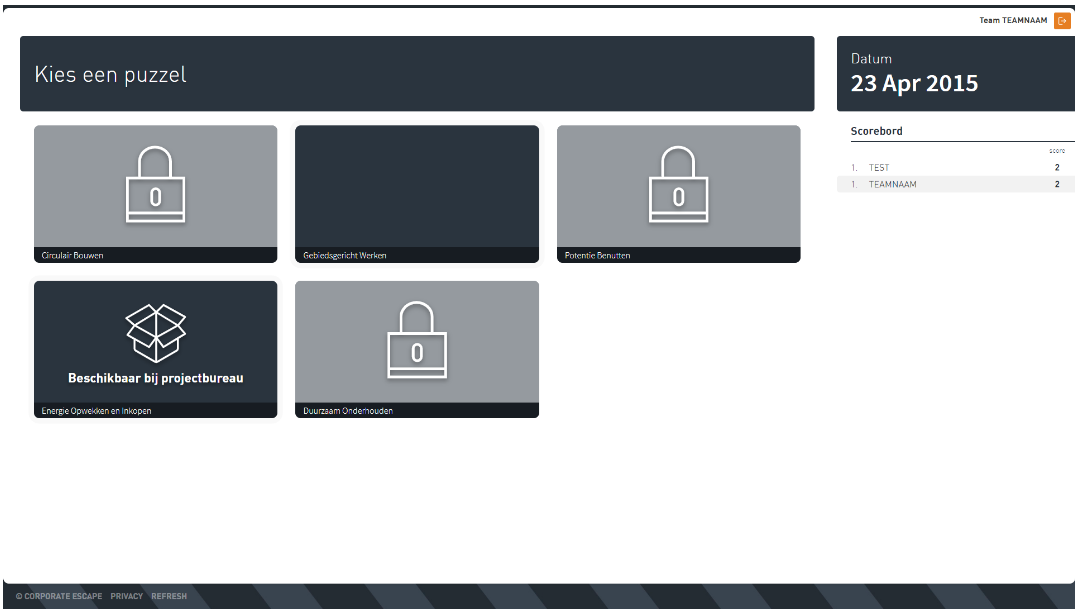
Dat antwoord is niet correct...

Scorebord

		score
1.	TEAMNAAM	3
2.	TEST	2

© CORPORATE ESCAPE PRIVACY REFRESH

Figure 7.4: Answering a multiple choice puzzle


Team TEAMNAAM

Kies een puzzel

Datum
23 Apr 2015

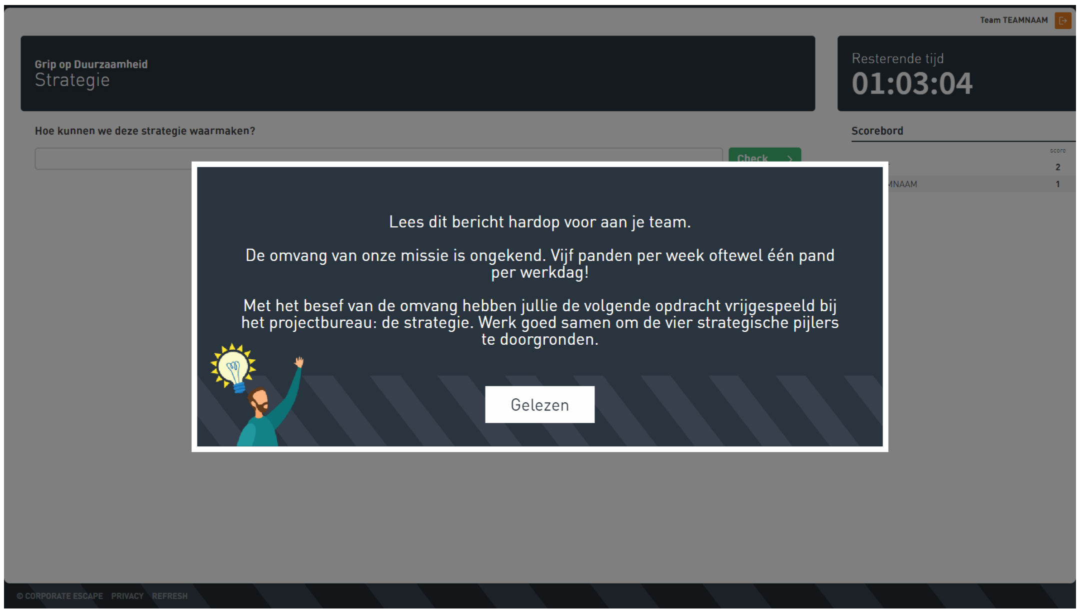
Circulair Bouwen
Gebiedsgericht Werken
Potentie Benutten
Beschikbaar bij projectbureau
Energie Opwekken en Inkopen
Duurzaam Onderhouden

Scorebord

		score
1.	TEST	2
1.	TEAMNAAM	2

© CORPORATE ESCAPE PRIVACY REFRESH

Figure 7.5: Overview of puzzles



Hoe kunnen we deze strategie waarmaken?

Lees dit bericht hardop voor aan je team.

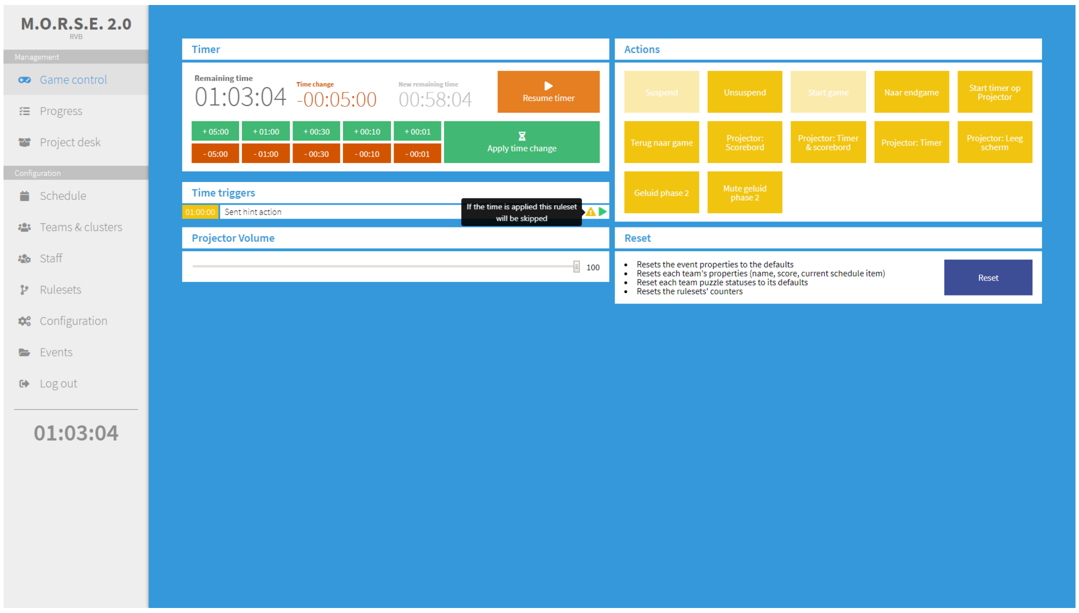
De omvang van onze missie is ongekend. Vijf panden per week oftewel één pand per werkdag!

Met het besef van de omvang hebben jullie de volgende opdracht vrijgespeeld bij het projectbureau: de strategie. Werk goed samen om de vier strategische pijlers te doorgronden.

Figure 7.6: A hint for the player

Hint
If a staff member sends a hint, the user receives their hint on their tablet, as shown in figure 7.6. A popup will cover part of their screen, the remaining space will be covered by a semi-transparent overlay, and nothing in the background is clickable anymore. A message indicating that a hint is received is shown first, with a button to ‘open’ the hint. Once that is clicked a similar screen is shown with the actual text of the hint. After 3 seconds a button to confirm that the hint has been read appears, this is done so the hint is not clicked away by accident, without being read.

Popup
Popups act similar to hints. A popup is an image which can be clicked in order to be dismissed. With popups it is possible to give the player some visual information. While the entire image is clickable, this might not be communicated clearly to the players. It is therefore highly recommend to simply include a button within the image that is shown. Players will then get the impression that the button dismisses the popup.



- Resets the event properties to the defaults
- Resets each team's properties (name, score, current schedule item)
- Reset each team puzzle statuses to its defaults
- Resets the rulesets' counters

Figure 7.7: The game control screen

7.1.2 Admin

Login

To access the administrator environment the user needs to be logged in and have a staff role. This can be done by going to the administrator login page. Do mind, however, that even though the team login page and the admin login page look the exact same, they are not actually the same page. At the admin login admin codes are accepted and once a correct code has been entered the user will be redirected to the administrator environment.

Admin sidebar

When logged in into the administrator environment there is always a menu bar at the left which allows quick navigation between several parts of the system. Depending on the role of the user, not all options are visible.

The menu is divided up into Management and Configuration. With Management being more focused on a running game and tracking progress and Configuration being focused on setting up an event and editing flexible options. At the top of the sidebar the name of the system is displayed, namely M.O.R.S.E 2.0 and also the name of the event the user is in at that moment. At the bottom of the menu the timer for the game is shown, such that it is also visible in screens that are not the game control screen.

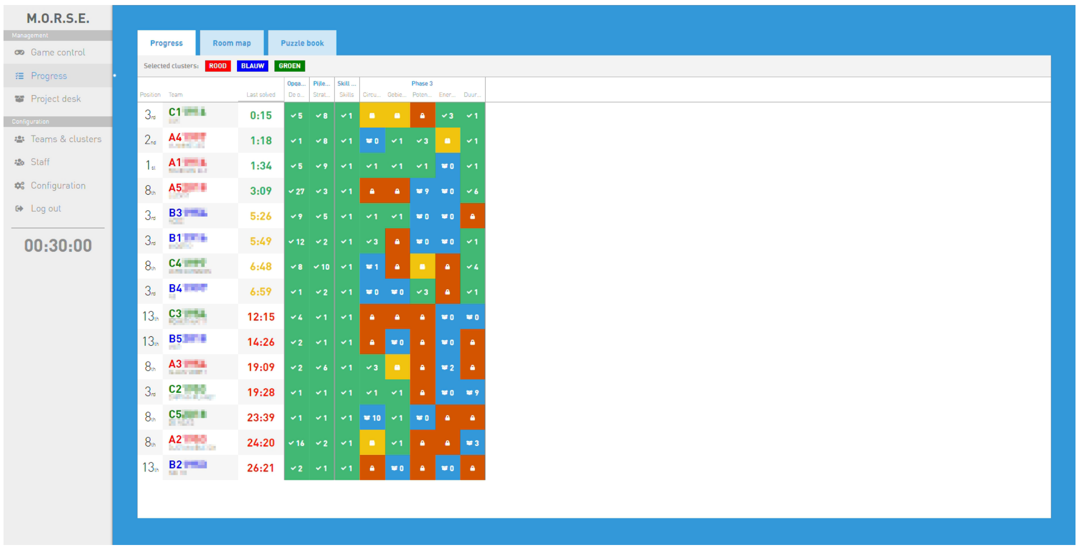
Game control

When logged in as a game host or administrator, the first screen that will appear is the game control screen (figure 7.7). In this screen the flow of the game can be influenced by one of the four components, namely: editing the timer, triggering manual rulesets, managing time triggers, altering the projector volume and resetting the event. The functionalities of those five components will be briefly explained.

Time control The first part of the game control is a block in which time can be configured. It shows how much time is still on the timer and what the current end time is in real life time. The time can be paused and resumed with the button to the right of the time.

The lower part of the time control is dedicated to changing the time. This could be done while paused but also when the time is running. When editing the time multiple buttons can be clicked to edit the time to the wanted value. While editing the time can also be seen. Once the time is ready to be updated, the user can click ‘Apply time changes’.

Manual ruleset triggers At the right side of the time control a grid of buttons can be seen. These buttons correspond to the manual triggers which are set in the rulesets. Clicking a button will result



			Opga...	Pijle...	Skill ...	Phase 3				
Position	Team	Last solved	De o...	Strat...	Skills	Circu...	Gebie...	Poten...	Ener...	Duur...
3rd	C1 [redacted]	0:15	✓ 5	✓ 8	✓ 1	🔒	🔒	🔒	✓ 3	✓ 1
2nd	A4 [redacted]	1:18	✓ 1	✓ 8	✓ 1	⏰ 0	✓ 1	✓ 3	🔒	✓ 1
1st	A1 [redacted]	1:34	✓ 5	✓ 9	✓ 1	✓ 1	✓ 1	✓ 1	⏰ 0	✓ 1
8th	A5 [redacted]	3:09	✓ 27	✓ 3	✓ 1	🔒	🔒	⏰ 9	⏰ 0	✓ 6
3rd	B3 [redacted]	5:26	✓ 9	✓ 5	✓ 1	✓ 1	✓ 1	⏰ 0	⏰ 0	🔒
3rd	B1 [redacted]	5:49	✓ 12	✓ 2	✓ 1	✓ 3	🔒	⏰ 0	⏰ 0	✓ 1
8th	C4 [redacted]	6:48	✓ 8	✓ 10	✓ 1	⏰ 1	🔒	🔒	🔒	✓ 4
3rd	B4 [redacted]	6:59	✓ 1	✓ 2	✓ 1	⏰ 0	⏰ 0	✓ 3	🔒	✓ 1
13th	C3 [redacted]	12:15	✓ 4	✓ 1	✓ 1	🔒	🔒	🔒	⏰ 0	⏰ 0
13th	B5 [redacted]	14:26	✓ 2	✓ 1	✓ 1	🔒	⏰ 0	🔒	⏰ 0	🔒
8th	A3 [redacted]	19:09	✓ 2	✓ 6	✓ 1	✓ 3	🔒	🔒	⏰ 2	🔒
3rd	C2 [redacted]	19:28	✓ 1	✓ 1	✓ 1	✓ 1	✓ 1	🔒	⏰ 0	⏰ 9
8th	C5 [redacted]	23:39	✓ 1	✓ 1	✓ 1	⏰ 10	✓ 1	⏰ 0	🔒	🔒
8th	A2 [redacted]	24:20	✓ 16	✓ 2	✓ 1	🔒	✓ 1	🔒	🔒	⏰ 3
13th	B2 [redacted]	26:21	✓ 2	✓ 1	✓ 1	🔒	⏰ 0	🔒	⏰ 0	🔒

Figure 7.8: Progress overview

in the ruleset being executed. The names of those buttons are defined in the ManualTrigger. If the conditions of the ruleset are not all met, the button will still show up but appear greyed out and not clickable.

Time trigger overview The third part of the game control is the time trigger overview. It displays all the time triggers which are defined at the rulesets. It shows the ruleset name, the in-game time at which the trigger will be activated and, if the conditions are met, an execute button. If the time of the time trigger has already passed, the trigger will be greyed out, however it can still be executed (again) if need be.

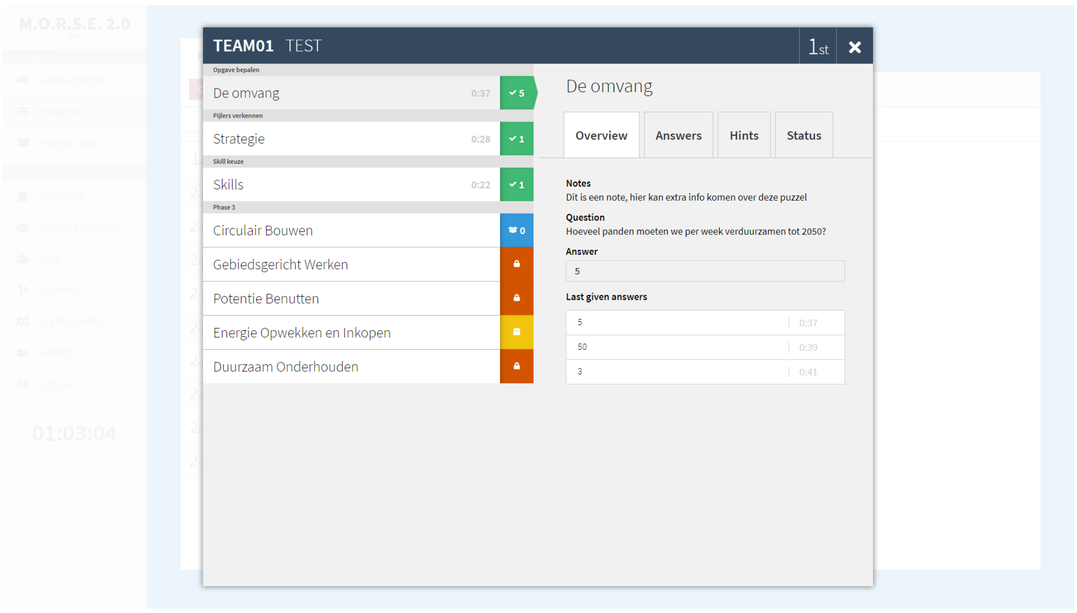
While editing the time it is a possibility that if the remaining time gets lowered, then some time triggers will be skipped. Because of this, a warning icon will appear while editing the time, warning the user that the ruleset will be skipped. In that case the user can decide to execute the ruleset manually using the execute button. Also, if time gets added and this results in going over a time trigger, that time trigger will show a warning in which it says that the ruleset might occur again if the timer reaches that time again.

Projector volume For easy of use for the game-host a volume slider is added to the game-control screen, this can be used for the game-host to quickly turn up/down volume during the game.

Reset button Before an event is actually run, it might come in handy to do a test run to ensure that all puzzle answers are correct and all ruleset logic is correct. After this test run, the event would needs to be reset in order to ensure it functions properly during the real event. The reset button accomplishes this goal, and clicking it resets the rulesets, puzzle statusses and teams. This is also explained next to the button, so the user is aware what is being reset exactly. Furthermore, after clicking the button, the user will be asked to confirm whether (s)he really wants to reset the event, in order to prevent accidental resets during an event.

Progress

For the foremen as well as for the game-host there is a page on which they can easily monitor progress for a team. It is designed for the needs of the foremen, so they can monitor, influence and control the progress of a team. It has also two more tabs in which a floor map and a puzzle book (containing hints and answers about the puzzles in the event) can be added. The separate components will be explained in the next paragraphs.



Opgave bepalen		
De omvang	0:37	✓ 5
Pijlers verkennen		
Strategie	0:28	✓ 1
Skill keuze		
Skills	0:22	✓ 1
Phase 3		
Circulair Bouwen		0
Gebiedsgericht Werken		
Potentie Benutten		
Energie Opwekken en Inkopen		
Duurzaam Onderhouden		

De omvang

Notes
Dit is een note, hier kan extra info komen over deze puzzel

Question
Hoeveel panden moeten we per week verduurzamen tot 2050?

Answer
5

Last given answers

5	0:37
50	0:39
3	0:41

Figure 7.9: The foreman popup

Progress overview The first thing that is visible is a table that directly gives insight into the progress for all teams. It shows the puzzles and the teams in a table, and every cell has a visual indication of the status of a puzzle for a team as shown in figure 7.8.

The top part of the table is a row in which a filter can be applied for certain clusters or for teams that are not in a cluster. This is useful for foremen as they can select the cluster for which they are responsible. This way irrelevant information is filtered out. The buttons toggle on click, if they are solid in their colour it means the cluster is not filtered out, if they only have an outline, it means that the cluster is filtered out of the progress overview.

The table of the progress is below the cluster filter. This table has as its rows the teams and as its columns the puzzles. Each team has a position on the leaderboard, a team code (displayed in the colour of the cluster which they are in) and a team name which is made up by the team themselves. Also, a coloured time is shown besides the team code and name, this indicates how long it has been since the last puzzle was solved for the team. The colour is to indicate whether the team needs attention, it starts off with green if the team recently solved a puzzle, then goes to yellow if it takes a while longer and it turns red when if it takes even longer, indicating that the team might need help. The moments that the colours switch from green to orange and from orange to red are configurable in the configuration editor.

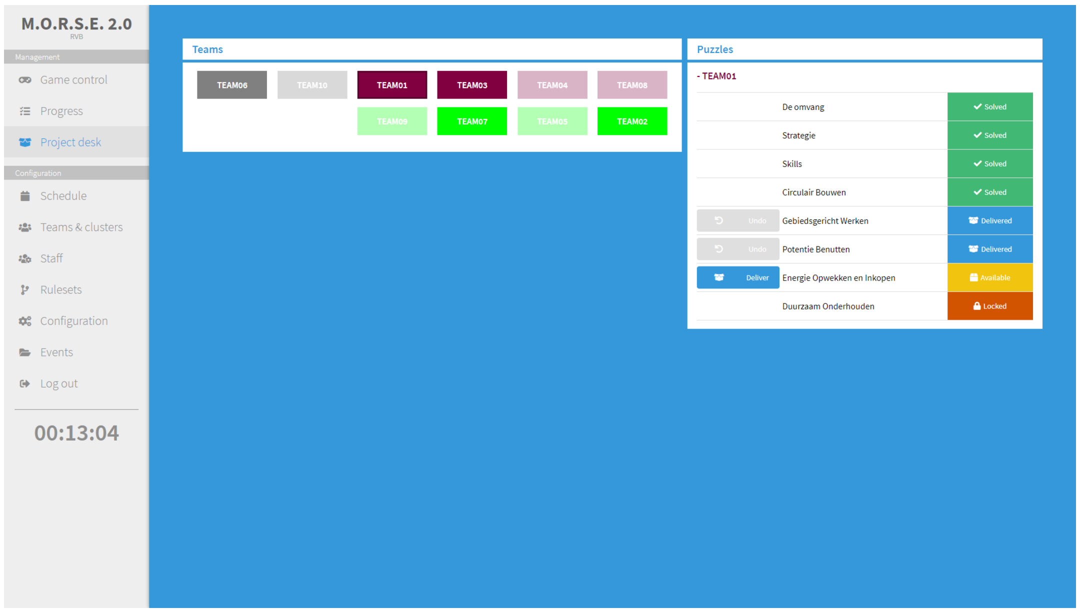
Each puzzle has a column, these columns are grouped into the separate phases which are configured by the game designer. This way a foreman can easily see in which phase the team is.

Each cell between the puzzles and the teams has an indication of the status for that puzzle with an icon. Red with a lock icon means that the puzzle has not been unlocked yet. Yellow with a closed box icon means that the puzzle needs physical attributes which can be picked up at the project desk, but are not yet picked up. Blue with an open box means that those attributes are picked up and the team is able to attempt the puzzle. Green with a check mark means that the puzzle is solved. The cells can also have a number next to the icon, which indicates how many attempts the team has made at the puzzle.

Progress overview popup The teams, as well as the cells are clickable. When clicked they open a popup in which more info about the team and the puzzle are given (figure 7.9). At the left side of this puzzle a list of puzzles is given. The puzzle can be selected by clicking. Once selected the right part of the screen is dedicated to the selected puzzle for the team of the pop-up.

The right side has 4 tabs, which will be briefly explained in the next few paragraphs.

The first tab is for the general overview of the question. It shows notes which are set to read for the foreman, containing info which can help them to help the team to solve that particular puzzle. It also shows the question and the answer. Underneath the answer there is a list of given answers and for



- TEAM01

	De omvang	✓ Solved
	Strategie	✓ Solved
	Skills	✓ Solved
	Circulair Bouwen	✓ Solved
Undo	Gebiedsgericht Werken	Delivered
Undo	Potentie Benutten	Delivered
Deliver	Energie Opwekken en Inkopen	Available
	Duurzaam Onderhouden	Locked

Figure 7.10: The project desk

each of the answers there is a time indicating how long ago they tried that answer, to ensure that the foreman does not have to scroll this list is capped to the last three. At the bottom of the page a list of wrong answers with their hints, this means that if the team gives an answer from that list that they get the predefined hint that shows in the list.

The second tab contains all the given answers for a team. This list is not limited to a maximum of three. This allows the foreman to see and track the progress of the team and in which direction the team is trying to find the solution.

The third tab is dedicated to hints. A foreman has an input field in which a hint can be typed and sent. If the puzzle has predefined hints (sample hints set by a game designer to help foremen), there is a dropdown menu option to select one of the texts, which will be automatically filled into the input field.

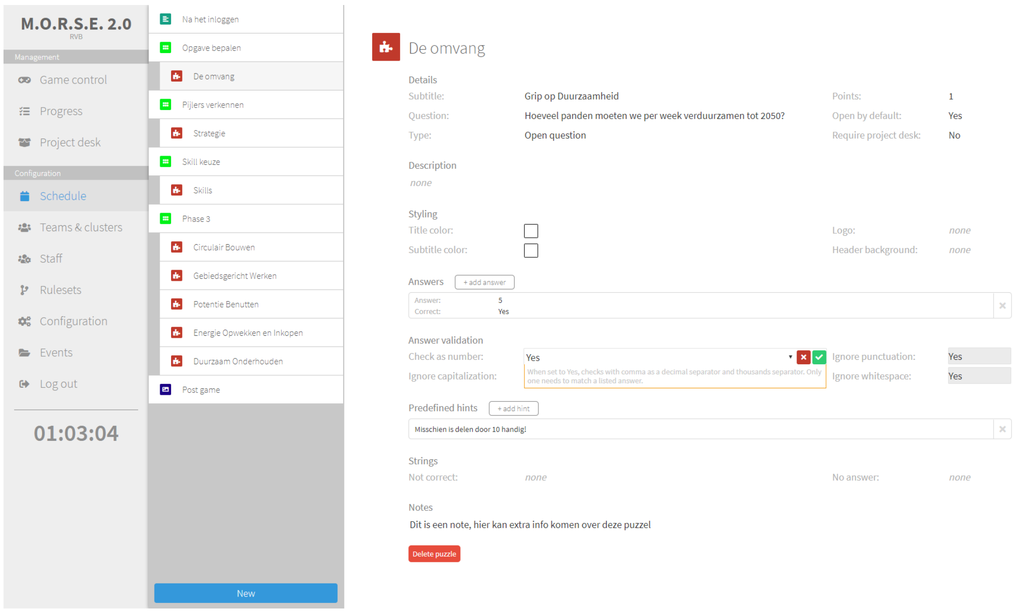
The fourth and last tab is for editing the status of the puzzle for a team. This allows a foreman to change a status in case something went wrong. This could be for example that the team gives the right answer, but due to an error the answer is marked as wrong, in this case the foreman can go to this tab and click the solved button.

Project desk

The project desk screen (figure 7.10) belongs to a separate role in the game, namely the project desk. The project desk has to change status of puzzles for certain teams. One of the most important requirements for the project desk is that the task can be done fast and in an efficient manner. The project desk screen has no added functionality compared to the progress screen, however, for ease and speed of use the same functionality is presented in an entirely different way for the project desk.

Teams overview To change the puzzle status for a certain team the project desk has to select a team first. For each team there is a button in the colour of their cluster which shows the name of the team and their team code. If a player is retrieving a puzzle the project desk member needs to find out the team of the player. One of the best identification methods is to narrow it down to a cluster by finding out their cluster colour (this differs for the different types of events, for example, some events have the players wearing a safety vest in the colour of their cluster). Once the cluster is found a team needs to be determined, this is done by either asking the team code or team name. For the second selection it is important that teams in the same cluster are adjacent to each other, this can be achieved by alternating the rows from left to right to right to left, this makes it easier to quickly go over the teams in a cluster.

Also if a team has nothing to pick up at the project desk, they will be greyed out, in that way the project desk only has to check for the teams that actually have something that can be picked up.



De omvang

Details

Subtitle:	Grip op Duurzaamheid	Points:	1
Question:	Hoeveel panden moeten we per week verduurzamen tot 2050?	Open by default:	Yes
Type:	Open question	Require project desk:	No

Description

none

Styling

Title color:	☐	Logo:	*none*
Subtitle color:	☐	Header background:	*none*

Answers

Answer: 5
Correct: Yes

Answer validation

Check as number:	Yes	Ignore punctuation:	Yes
Ignore capitalization:	When set to Yes, checks with comma as a decimal separator and thousands separator. Only one needs to match a listed answer.	Ignore whitespace:	Yes

Predefined hints

Misschien is delen door 10 handig!

Strings

Not correct:	*none*	No answer:	*none*

Notes

Dit is een note, hier kan extra info komen over deze puzzel

Figure 7.11: The schedule editor, editing a puzzle

However, if a team comes which does not have any items to be picked up, the project desk can still select them to change a status.

The combination of the ordering of the teams and greying out the teams that do not have anything to pick up, make it for the project desk easy and fast to select the right team.

Questions overview Once a team has been selected at the right side a list of question will be shown. Each question will have, next to the title, an indication of the status, with a colour and icon (which have the same meaning as the icons described in the progress screen). In addition to that, if a question has pick up available, a blue button with text 'Deliver' will be next to the title. Clicking this will set the status of the puzzle for the team as delivered. If the puzzle is delivered, but not solved yet, the project desk has the option to undo the delivery, this is useful if by accident the wrong puzzle was marked as delivered.

Schedule and puzzle editor

To allow a game-host to create a schedule for the game and add puzzles there is a flexible schedule editor (figure 7.11). The page is divided into two parts, the left part being the list of schedule items and the right part the editor for the selected schedule item or puzzle.

List of the schedules and puzzles At the left hand a list of the puzzles and schedule items can be found. With the add button at the bottom of the page new items can be added to the current schedule.

The first is a schedule group, indicated by a folder icon and a light blue colour, this is a schedule item which is made to have other schedule items inside of it. It can be used to create a stage which contains other items in it. Once made other schedule items can be dragged into it creating a nested structure.

The second option is a puzzle screen, which is indicated by a green icon with a grid of 6 white icons. A puzzle screen is used for showing the user puzzles, if there is only one puzzle in the screen then the user directly gets to see that puzzle if in that schedule item. If there are more puzzles the user will see a menu in which (s)he can pick what puzzle to start with.

The next option is a dark green icon with a list in it. This is the text screen schedule item option. If a user is inside this schedule item, the user will see a screen with a predefined text.

Apart from displaying text on the screens there is also the option to display an image, this schedule item is indicated with a dark blue icon with a white picture icon in it. This schedule item has to have a predefined image.

The last option is to adding a puzzle. This option is indicated with a red icon with a puzzle piece in it. Puzzles can only be added underneath a puzzle screen, and a puzzle screen must be selected before this option becomes available.

All schedule items can be dragged and dropped within the schedule, giving the game-host control in a dynamic way over the event. One important thing to mention is that the order in which the schedule items are displayed here are not automatically the way the user will traverse through the schedule, this has to be done with the rulesets. By default the teams will start in the top-most schedule item, at the time of creating the teams. It is therefore recommended to create the teams after the (top-most) schedule is final. If not the game host's reset button can be used to ensure the correct starting schedule.

Editor Once an item in the left list is selected (or a new item is created), then on the right part of the screen the edit menu for that item will be opened. All options will be briefly described.

Each field be shown and when the game-host hovers over the field, an edit option will appear which will change the field to an input field if clicked. Some of the fields have an additional explanation, which will be shown when editing the field. If the field is edited and the green check mark is pressed, then the update is automatically saved. But the value is also saved if another property to edit is selected.

The first edit screen is the one of the schedule group. Since schedule groups are added for structure and are not a leaf item in which the user can reside, it only needs a title for the structure. The second edit screen is for the schedule puzzle group, which has a title but also the property skippable of which the workings are described in 5.3. The next edit screen is the text screen. It also has a title, skippable property and of course the text which has to be displayed for the users if the user is in that schedule item.

The edit screen for the image has more properties because it has a title, skippable property, image source, a 'fit to screen' option and background colour. This allows the user to choose whether an image should be expanded to the width of the device the user is using or whether it should not (both options preserving aspect ratio). When it is not stretched to fit the screen, then there is a possibility that there are empty spaces, those will be filled with the defined background colour in the schedule item.

The edit screen for the puzzle editor is divided up into seven parts. The first part is the details part and contains some basic information and parameters about the question. After that there is a possibility to add a description, which will be added above the question. Under the description there are a couple of fields in which the styling of the question can be changed. The next part is a list of answers, to which new answers can be added. Each answer has a string of what the answer is, an indication whether it is a correct or wrong answer and, if it is a wrong answer, a text which should be displayed for the players when they try this answer. A wrong answer simply indicates that if the user gives that answer, they get the custom defined message instead of the usual message. Then a list of predefined hints can be made. These hints will be options for the foreman to select and send. After that there is a dedicated section for the validation of the given answer, there are several self-explanatory options such as ignore capitalisation. And the last option is to add notes to a question, those notes can be read in the foreman progress screen when opening the question. Notes could, for instance, contain hints for the foreman on how the puzzle should be solved.

Team and cluster editor

In the cluster and team editor (figure 7.12), clusters and teams can be created, edited and updated. The screen consists of three parts, namely the create team block, the create cluster block and the management block to manage both the created teams and clusters.

Create teams Teams can be created using the input field on the right side of the page. Multiple teams can be added at once by having separate team codes on each line, the team codes should be the same length as the defined length in the event. The teams which are added this way will not be assigned to any cluster. In the input field a cluster name could also be added, using a comma or semicolon separated format. If the cluster name exists then the team gets added to that cluster, if it does not exist the cluster will be created with a random colour or, if the cluster name is a valid colour, that colour will be used for the cluster. If the cluster name is not a colour, a colour could be added as third column of the insert query. If the cluster already exists, this colour should match with the colour of the existing colour. If there are any mistakes made while inserting, the teams that could not be inserted will not be removed from the input field.

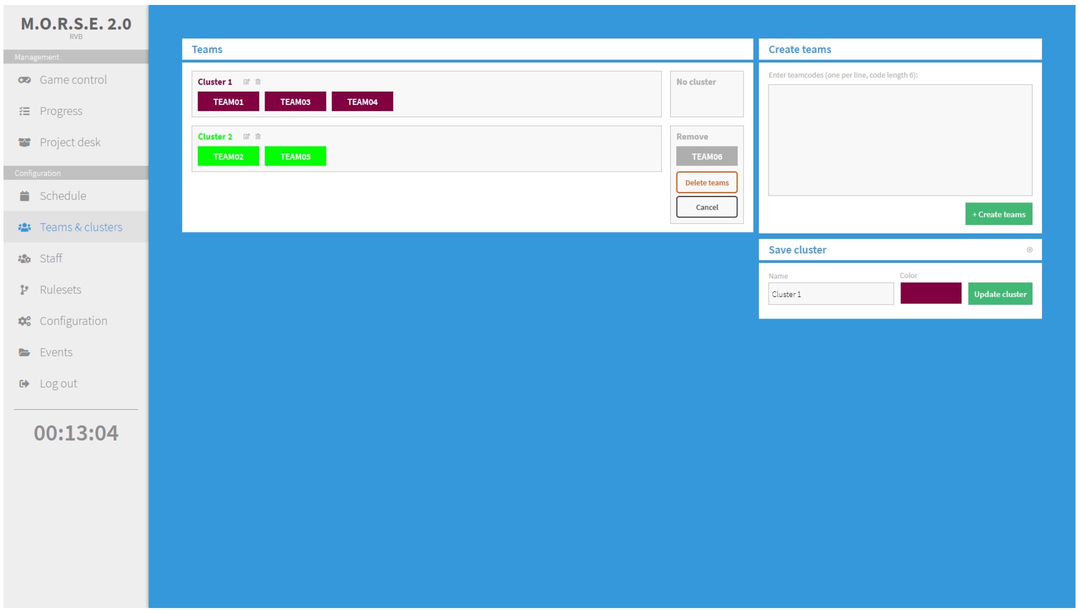


Figure 7.12: The team/cluster editor

Create cluster Besides from creating a cluster using the input field for teams, there is also a way to manually create a new cluster, the option for this is displayed underneath the create teams block. This block also is used to edit clusters if one is selected for edit in the overview block. For each cluster a name and a colour is picked, the colour can be picked using a colour picker.

Manage clusters and teams The main part of the team and cluster editor is the management block for the teams and clusters. This block consists of various blocks, one for the teams without a cluster, one for removal (used as a bin) and one for each existing cluster. The teams are displayed as cluster-coloured blocks inside those blocks, inside the team code is shown. Teams can be dragged and dropped between clusters and no cluster, once dropped it is automatically saved. Each cluster also has two buttons, one for editing and one for deleting. Dragging teams into the remove section will mark them for removal, they are not actually removed until 'Delete teams' is pressed, then all teams which are in the removal section are removed permanently.

Staff editor

On the staff page (figure 7.13) the game-host can add codes for certain roles. This determines which pages in the admin environment a staff member is able to see. The roles are Admin, Game host, Foreman and Project desk. An admin and game host can see and do everything the foreman and project desk can including all other features. The only thing the admin can which the game-host can not is manage multiple events.

The staff editor works the same as the cluster an team management. The only differences are that only team codes can be added and no cluster/colours. Instead of clusters the blocks are dedicated to the four separate aforementioned roles and can be dragged and dropped around just like teams could be changed from clusters.

Ruleset editor

In the ruleset editor screen (figure 7.14) rulesets can be created, updated and deleted. It has an intuitive design and feel, making it easy to create rulesets. Just like the schedule editor, it consists of a list, in this case of ruleset and if a ruleset is selected, then at the right side the edit panel appears in which the rulesets can be modified.

The list is a simple list displaying the titles of the rulesets. At the bottom of the list is a create option, which opens a new empty ruleset which can then be saved.

The edit part consists of several sub parts. The first being the title, the title will be used to get more overview while editing and is shown at the time triggers at the game control screen. Below that there

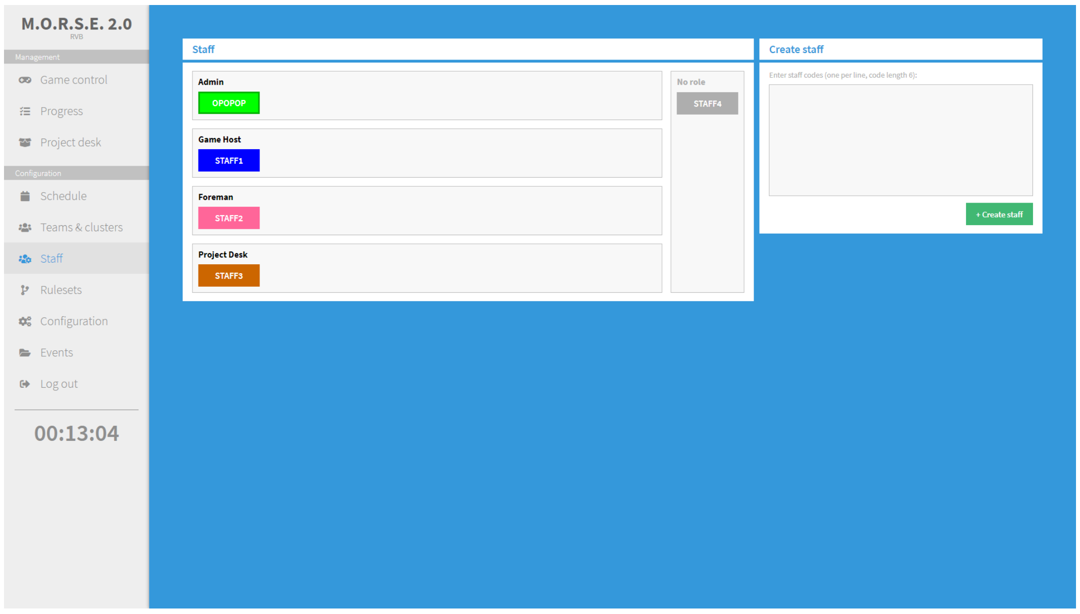


Figure 7.13: The staff editor

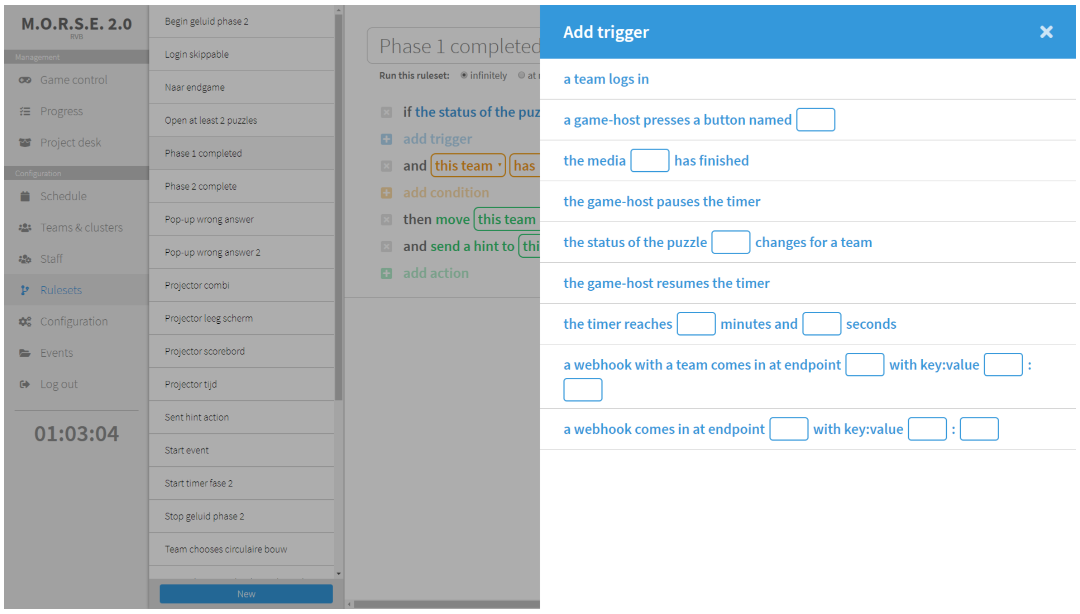
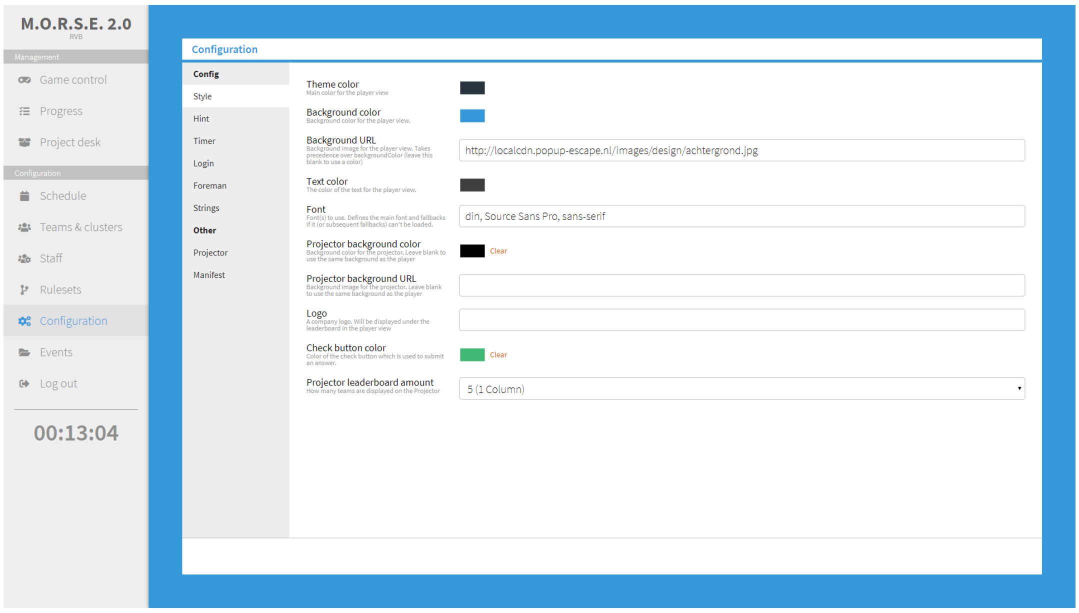


Figure 7.14: The ruleset editor



Configuration

Config
Style
Hint
Timer
Login
Foreman
Strings
Other
Projector
Manifest

Theme color Main color for the player view	■
Background color Background color for the player view.	🟦
Background URL Background image for the player view. Takes precedence over backgroundColor (leave this blank to use a color)	http://localcdn.popup-escape.nl/images/design/achtergrond.jpg
Text color The color of the text for the player view.	■
Font Font(s) to use. Defines the main font and fallbacks if it (or subsequent fallbacks) can't be loaded.	din, Source Sans Pro, sans-serif
Projector background color Background color for the projector. Leave blank to use the same background as the player	■ Clear
Projector background URL Background image for the projector. Leave blank to use the same background as the player	
Logo A company logo. Will be displayed under the leaderboard in the player view	
Check button color Color of the check button which is used to submit an answer.	🟩 Clear
Projector leaderboard amount How many teams are displayed on the Projector	5 (1 Column)

Figure 7.15: The configuration editor

is an option to choose how many times the ruleset should run at most, if it does not run infinitely or is not disabled. Next to that option there is a counter which shows how much time the ruleset has been executed already, with next to it the option to reset this counter back to zero.

Underneath the run settings and title is the main part of the editor, namely the composition of triggers, conditions and actions. For clarity each of them has a different colour. Each ruleset should be composed of at least one trigger and at least one action. Since the part of adding, deleting and editing is similar for triggers, conditions and actions, the structure will be explained for triggers, but can also be read as conditions or actions.

Each trigger has its own row, starting with a grey cross mark icon, with which the trigger can be deleted. Next to that the text for the trigger is displayed. This is a short text from a predefined template which explains what the trigger does. Most triggers have additional parameters, which will be displayed in the template as either a dropdown menu or an input field, these can then be edited to the right values.

To add a trigger the last row of the triggers section is not a trigger itself but is named 'Add trigger' and instead of a grey cross mark icon it has a blue icon with a plus in it. Once clicking that a menu will open up which displays all the triggers which can be picked from as seen in 7.14. Once clicked they will be added and if it contains parameters those will display empty.

A special constraint for a ruleset is that if a team-based condition or action is used that all the triggers should be team-based, otherwise the condition and action do not have a team or cluster for which they should check or execute. If there are team-based actions or conditions and a not team-based trigger the ruleset can not be saved (red text ruleset invalid), but also the team-based actions and conditions will have a red error icon indicating that the combination is invalid because there are triggers which are not team-based.

At a ruleset there are three options, delete, save and cancel, which will close the ruleset and forget the changes which were made.

Configuration editor

The configuration editor (figure 7.15) is an editor which can change various things inside the event. This has direct influence on what is happening and the layout. It helps keeping the events as dynamic as possible since a lot can be edited.

The screen consists of a list of categories with their sub options, all the sub options are clickable and open several options on the right part of the screen. Each option has a name, explanation underneath and the actual input. The save button has to be clicked if the values are changed and have to be saved. At pressing the save button the inputs will be validated, if the inputs are not valid it will not be saved an show a popup that the updating has failed, with the option to refresh to how the option used to be.

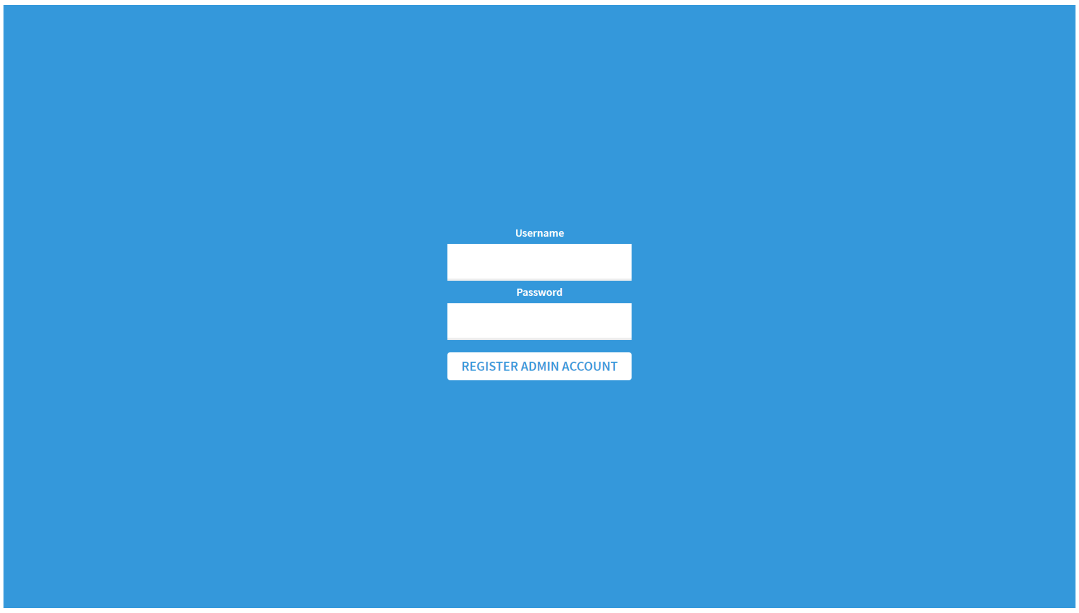


Figure 7.16: The administrator register screen

Besides that if one user is updating the configuration and another person is also on the same page making changes, the second user will get a warning that the configuration options were updated by another user and a possibility to refresh to make sure (s)he is updating the most recent version of the configuration.

As for input fields, there are several types of input fields, namely text, number, colour picker, slider and dropdown menus.

The styling, font, colours and texts of the player screen can all be configured here.

Event editor

There should be an ability to manage multiple events, switch between them, create, update and duplicate them. Therefore the event editor (figure 7.17) has been created. There are two different way to get into the event editor, both of which will be described below.

The first way is when there is not an admin account yet. Then an admin account needs to be registered. At most one admin account can be registered. The page '/events' will redirect to a register page (figure 7.16) at which a username and password can be filled in. After that the user is logged in as admin and sees the event editor. The next time that a not logged in user goes to the '/events' page it will show a login prompt for the admin which was registered the first time. Other codes or registering a new code will not work here.

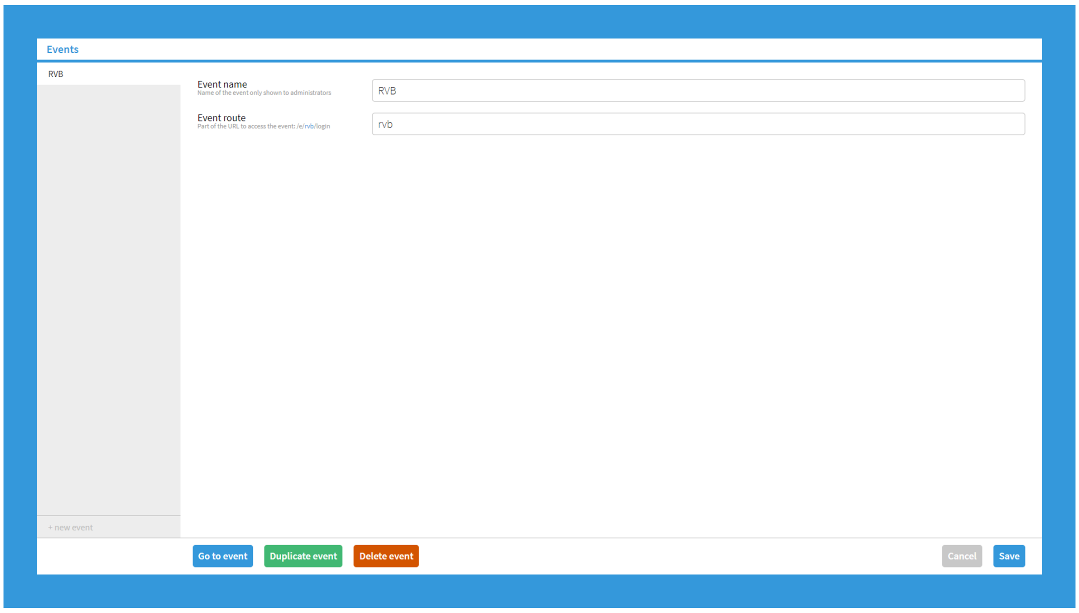
If there already is an event and the user logged into the admin panel, the event editor can also be found in the sidebar menu of the admin panel.

At the left side of the event editor all events are listed and there is an ability to add a new event. At the right side is the edit screen for an event. When creating a new event the route, name, team code length and optionally an initial admin code should be defined. When editing only the route and the name are editable (the team code length and admin options are then moved to the admin panel of the particular event).

There is also a button which allows duplicating an event. It requires a new route, and all other things are copied, except for the teams. Furthermore, there are buttons to delete the event, or visit its admin panel.

7.1.3 Projector

The projector (figure 7.18) is a separate screen used to give players more information during an event. For this purpose it can show text or an image, play a video or play a sound. Likewise, the sound can also be used to play some music to set the mood of the game. Furthermore, the projector can also show the timer and the leaderboard, or a single screen with both options.

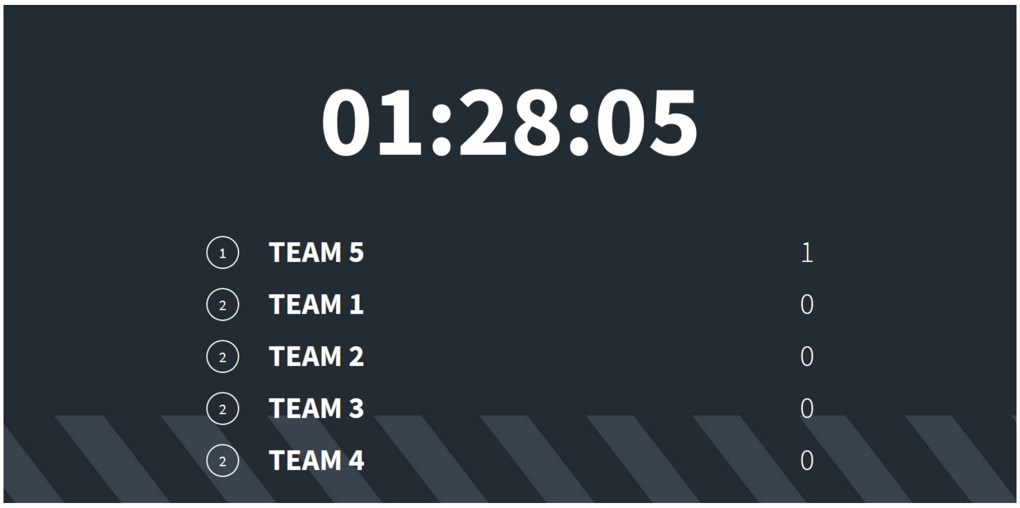


Event name
Name of the event only shown to administrators
RVB

Event route
Part of the URL to access the event: /e/rvb/login
rvb

Figure 7.17: The event editor

The Projector can be controlled with rulesets or manually from a menu located in the configuration editor. Additionally, the volume of the sound or video played through the projector can be controlled with the volume slider in the game control as shown in figure 7.7.



01:28:05

1	TEAM 5	1
2	TEAM 1	0
2	TEAM 2	0
2	TEAM 3	0
2	TEAM 4	0

Figure 7.18: The projector showing the timer and the leaderboard

7.2 Comparison

In this section the M.O.R.S.E system will be compared to the M.O.R.S.E 2.0 system based on different aspects.

7.2.1 Functionality

First, the functionality of the user interaction will be compared for each role.

In general, the user interaction has improved significantly compared to the original system. Where in the old system features were scattered across pages in a nonsensical fashion, the same features are now available in more logical locations. Furthermore, the usability of the system has improved for people without past experience with the old system, as we have learnt from the feedback we have received.

Admin

The admin in the developed system has the ability to create, duplicate and delete instances of events. This is done by logging in to a separate admin part. Here the admin can also edit the route. This functionality compared to the old system is entirely new. In the old system there was not a way to duplicate, create or delete instances of the event, the entire database was based on only one event. The only way to delete, create and duplicate events was by doing those operations on the database. This means that the new system has a better way of managing the events.

Compared to the old system, there was not a role like this, being able to manage events in the front-end, so this is actually an improvement, especially because it does not have to be done in the back-end any longer. Another advantage is that nearly every event can be set-up only using the web application and without changing any code.

Game designer

In the previous system, the game-host had a lot of ways in which the game could be influenced with buttons. The options the game-host had were controlling time, switching phases and starting sound clips or videos. In the newly created system those functionalities are also available, some of which through the rulesets, which gives a lot more flexibility. A game designer can arrange the game control screen using the `ManualTrigger` in rulesets. Also, in the old system the phases were hard coded, in the developed system the phases are dynamic and could be switched between using buttons.

The old system has a configuration page were several important parameters could be saved. The follow-up system has a lot more parameters which can be edited real time. One of the more impact full additions to the configuration is the ability to change the entire style of the client, as well as all the texts at the client side.

The game flow has also significantly changed from the old to the new system. In the old system there was a predefined path beforehand defined in the code the players would go through. This is now controllable in a dynamic fashion real time using rulesets. The rulesets can also be used to create easier a more dynamic schedule based on time or progress of the teams.

The functionality for the game-host has been changed significantly. The game-host has way more control of the game by using the configuration editor and the rulesets.

Foreman

The foreman has its own page in the old version of the M.O.R.S.E system. On this page he could see the status of each team of his cluster and was able to send (predefined) hints. In the newly developed system the foreman has its own page too and also has a page on which (s)he can she the progress of the teams, but in the newer version there are some important filters added, including a coloured time which indicates how long ago that team solved their last question, which is an indication on whether they need help. The overview is also a lot clearer now and it takes less time for a foreman to understand what is going on.

If the foreman selects a team there is a lot more info to find about their given answer and when those answers were given, some general info about the question and the ability to override the status of a puzzle for a team. Those functionalities are new compared to the older system.

The functionality for the foreman has changed quite a lot, a lot of extra information/overviews have been added. Everything is also shown in a more clear way, allowing to have a better grasp of the progress of the teams in the cluster.

Project desk

In the old system the project desk already worked very efficient and very well. The new system has copied some aspects of the design of the project desk and tried to increase the efficiency even more. This is done by greying out teams which do not have any thing to pick up and instead of the normal ordering, the project desk teams in the new system do alternate in rows from left to right to right to left, to create a kind of 'snaking' effect. This ensures that teams in the same cluster are always adjacent to each other. The M.O.R.S.E 2.0 system also does not show the undo deliver immediately after delivering, but after 1.5 seconds, because it is not realistic that a puzzle delivery gets undone on purpose within 1.5 seconds, but can happen on accident occasionally.

The project desk functionality has been changed a bit, but not a significant amount.

Player

The experience for the player for the M.O.R.S.E system has not changed a lot. There are some things that changed are added, but the main idea is the same. One of the new features is the ability to receive hints as a player, it opens as a popup indicating that you got a hint, when clicked, the actual hint is shown. This used to be done with an integration that allows foremen to send a hint to a phone, however this functionality was not required any longer, so therefore the ability to receive hints on the tablet was added.

One of the more significant changes is the change in the puzzle overview screen. All the icons of the puzzles that are available to solve will now have a subtle flash effect, indicating that they are clickable. Also, if a player is solving a puzzle in a group of puzzles, the other puzzles in the group are shown on a sidebar on the left. The unavailable puzzles are also clickable but will show the reason why they are not solvable yet. Compared to the old system this is an improvement, since there the puzzles could only be all seen at the overview page and the puzzles did not feel clickable.

Another change compared to the old system is that the player can get predefined feedback when a certain wrong answer is given. This allows players to get a hint if they are close to an answer but not yet there.

The experience for the player has not been changed in a drastic way, the main difference is some quality of life upgrades.

7.2.2 Maintainability

The new system has been designed with maintainability in mind, because once it is finished it should be able to run on its own without any further edits in the code. For the old system that has not been the main goal, since the knowledge to edit the code was available. Due to the ruleset and schedule editors, events can easily be created and edited. Furthermore the config editor allows all visible text fields in the team client to be altered. These editors cause the system to be more maintainable compared to the previous system, since the changes can be made in the administrator panel rather than in the code.

7.3 Reflection on requirements

At the start of the project, requirements were established to build a product which would incorporate the needs of the client. This section will discuss how these requirements have been implemented and what choices were made with regards to implementing or not implementing a feature.

7.3.1 Functional requirements

Each of the stakeholders of the new system had a list of needs they had for using the system. These needs were translated to functional requirements (see 3.3.2) and prioritised using the *MoSCoW* method [9].

Team

For the team, i.e. the client side which is used on the tablets, all *must haves* and *should haves* are implemented. The *could haves* are not. The overview of hints for a client is something that has not had priority, since the way hints are sent now forces players to read the hint before dismissing it. The feedback questions after the game also did not have priority, since the company now uses methods outside of the system to gather feedback.

Projector
All *must haves* for the projector have been implemented. The requirements in the *should haves* are also met, but there is no specific view to see only the winner. This can however be done using the leaderboard view. The automatic statistics views have also not been implemented.

Game designer
The game designer did not have any *must haves*, since the puzzles could also be hard coded instead of being editable in the UI. This editor was part of the *should haves*, which have all been implemented. In addition, most of the *could haves* have been implemented. There is not a possibility to create new input templates from the admin interface because of the way input templates are implemented. Like on the player- and projector-side, evaluation questions and statistics have not been implemented. Also, exporting the answers is not possible using a feature in the admin interface. This can however be done by exporting the MongoDB database directly, which is beyond the scope of the system.

Game host
All *must haves* and *should haves* for the game host are implemented in the new system. Also, gamehosts have the capability to change puzzle statuses for teams. One of the *could haves* was the capability to set phone numbers for teams, but since hints using mobile phones are not used anymore, this is not implemented. Hiding teams which are not participating is not implemented as special feature, however, teams can be hidden by deleting them or moving them to a different cluster, which is then filtered away from the progress screen.

Foreman
The foreman has a completely new overview compared to the overview in the old system. This is why an adaptive overview based on the stage of the game was not necessary. All other *must* and *should haves* for the foreman are implemented. The tool to simulate the player login is not implemented, although most of the information a player can see is also available in the foreman overview.

Project desk
All features described in the *must haves* and *should haves* for the project desk are implemented. The sorting on likelihood of teams coming to pickup puzzles is not implemented. However, there is a highlight for teams who have something to collect, making it easy for project-desk staff members to find the teams coming in to collect.

7.3.2 Non-functional requirements
As well as functional requirements, non-functional requirements follow from the needs of the client and describe what should be taken into account when developing the system.

Maintainability
All code in the project is subject to linting, resulting in a consistent code style. In addition, practically all back-end functionally has been tested. This provides the assurance that functionality does still working after changing parts of the code.

Modularity
The system is split up in small modules and small methods making it easy to change or correct parts of the code. This can easily be extended and parts of the system can be replaced without affecting the complete system.

Usability
The system has been tested in a real-world event, in which actual users have used the system successfully. They had a positive experience with the usability of the system. Note that no players have received prior explanation of the system, and that all players were able to understand the application and participate in the event.

Deployment
Deploying the system and all of its components to the Galaxy hosting can be done by using the Meteor deployment procedure. This comes down to one command, making it easy to deploy after changes.

Scalability
The scalability of the system is mostly dependant on the amount of resources provided to the server. With Meteor, this can be scaled up before the events.

Performance
In the testing done before the event and during the actual event, which both used around 30 connected devices, there were no performance issues at all. Also, the CPU and memory usage charts did not rise more than 10% during the usage of the system. From this, we can conclude that the system could be used with more than 30 devices.

Robustness
When processing input of the user, erroneous input is dealt with. In addition, any errors that might occur during the event do not result in the system crashing, but only in the attempted action failing.

Security
From the players perspective, there are no ways in which a user can perform actions that are not intended. Information which should not be available to the player is not published, e.g. the locked puzzles, puzzle answers and answers of other teams. Players can not perform actions from the admin environment and can not perform attempts on puzzles they do not have access to.

Logging into the admin environment can be done using defined codes. Depending on the role of the code, the logged in user can then perform a set of actions. Other actions are then restricted. A foreman can not control the time for example. Furthermore, when a player tries to access the admin environment, they are redirected to the player page via router guards.

7.4 Ethical implications

In this section the implications of the M.O.R.S.E 2.0 system with regards to ethics will be explained. Possible ethical issues will be described and evaluated.

The system is used at escape events at which participants are grouped in small groups. Those small teams log in using an unique code which can be found on their table. In the system the team has to enter a team name and after that they can answer questions. No personal data is gathered or asked, only the team name and their answers to puzzles. This means that there is no possible way to trace back towards a person using the data which is saved. Therefore the system does not pose any ethical violations based on privacy.

This statement is further supported by the privacy statement of the company this project is carried out for[1]. The first part of the privacy statement explains that team codes (also in the old version of the M.O.R.S.E system) are not linkable and not linked to personal details and that there is no way to to link a team code to a person.

[1]`https://www.escape-event.nl/privacy-policy-event/`

8 Process

In this chapter, the team will look back at the process of creating the final product. First the choice for Scrum will be explained, followed by what each group member contributed to the created system. Next a reflection on these choices and the rest of the process. The last part of this chapter looks back on the internal communication between the team and external communication with the client and the coach.

8.1 Choice for Scrum

Scrum is a well-known agile managing framework, it is commonly used in software development. The aim is to improve the prototype product each iteration, or so called sprint.

During the project, weekly sprints were held according to the Scrum methodology. At the start of every week the goals for that week were established and split up into smaller issues during the sprint planning.

Each day started with the daily Scrum, a daily stand-up where each team member explained what they had done the day prior and what they would do that day. The rest of the day consisted of working on the project.

At the end of the week, both the work that was completed and not completed was reviewed. This part is known as the sprint review. The client and coach were not present during these reviews, but separate meetings with them were held, where progress of the sprints were discussed and a demonstration of the current product was presented. Furthermore, a short verbal sprint retrospective was held at the end of the week. The feedback from the retrospective was used to make the next sprint even more productive. For example, this led to some team members preparing the issues before the sprint planning, so that the whole process of planning took less time.

An important part of any Scrum process is the sprint backlog. In plain terms, this is the list of work that must be completed before the end of the next print. A JIRA[1] board was used to keep track of the issues, it could be tracked to who an issue was assigned to, and in what state the issue was: ‘to do’, ‘in progress’, ‘in review’ or ‘done’. JIRA has an integration with GitHub, so when a branch related to an issue on JIRA was merged with the master branch an issue was immediately closed on JIRA. The same principle applies to issues in progress and in review.

The choice for Scrum was a simple one. As the project had a very defined time-schedule and clearly defined requirements by the client. Scrum was a natural fit, as everyone in the team had experience with the methodology and the agile yet structured shape of it created a reliable working environment. These facts made Scrum the tool to use for the project.

8.2 Distribution of work

Throughout the project, the work was distributed amongst the members of the team. Everyone had assigned roles and worked to the best of their strengths. Next to their strengths, each team member also attempted tasks outside their area of expertise. Furthermore, everyone was responsible for a few tasks outside of their programming tasks. These can be found in the project plan in appendix A.

[1]https://www.atlassian.com/software/jira

The design of the UI was done by Tim. Most of these designs were then realised by Alexander and Tim. The back-end code was mostly written by the other three. Ayrton, who focused a lot on the puzzles, users and their interactions, Matthias, who is the main architect behind the rulesets and Wouter, who worked on time and staff related tasks.

The team was quite happy with how this distribution worked out. The process was smooth and the order in which tasks were completed connected well. Finally, the team also had the idea to start work with the rulesets in mind, with every step came the thought: “What will this do, does it affect the rulesets and if so, how can it be made so that it complements the rulesets best?”. Overall the initial idea of rulesets did not affect the process a lot, because the system was designed in such a modular way, that the initial lack of rulesets was not blocking the other development.

8.3 Looking back at the process

When looking back at the progress made during the sprints, we were quite happy with the results. The consensus was that Scrum allowed for some fine work and that it created productive working weeks.

In meetings with both the client and the coach, they both stated some concern with the large amount of work planned every week and they urged the team to reevaluate the planned work as they would rather have a more modest but working project. However, later successes managed to soothe those concerns. This was accomplished by having a clear overview of what needed to be done. Before the programming phase of the project, we made a list of high-level components which needed to be finished each week. Nearly all components where finished according to the created schedule. The editors took a little longer than expected, however we left some room in the planning. This allowed us to finish everything on time. Furthermore, improvements were found by the continuous sprint retrospective, mentioned in 8.1, these improvements smoothed out the process and made for a better distribution of resources. This better distribution improved the time in which issues were completed and also improved the general flow of the sprints.

However, a point of contention were the so-called story points. These points often low-balled the amount of time required to complete an issue and as such, there were some issues in the first three weeks which were not resolved in time. In the those couple of weeks, sometimes people had to wait until another team member was done with their work before they could continue. This caused both the client and the coach questioned whether the team could actually succeed at finishing the project in the given timeframe. This was resolved by smarter use of time and better prioritisation of work. During some sprints, the client asked if some extra functionalities could be implemented. This led to us altering the current sprint to fit these needs. Sometimes this caused issues to be moved to the next sprint, but only if this would not cause problems with the overall planning. Additionally, the team also asked the CTO of Raccoon Serious Games, Jan-Willem Manenschijn, what other features should be implemented in the system, if they were to continue using it after the project.

Before a team member could merge his work into the master branch, a pull-request on GitHub had to be reviewed by at least two other members. Both the code and a running version of the product was tested before each merge. This was very useful for us, since it prevented many bugs from entering the master branch and increased the overall quality of the product.

All in all, we had a successful use of Scrum and learned a lot from the whole process. Everyone was satisfied at the end of the project about the job done. The experiences gained from the project will be carried by each member into different future projects.

8.4 Internal communication

During the project, most of the internal communication was done in person since we as a team worked on location almost every working day. During the day, all members were up to date with what everyone was working on. If anyone had a question or a subject he would want others’ idea on, this could be asked directly.

This was beneficial since very few misunderstandings could occur by communicating face-to-face. In addition, all expertise was in one place which meant that potential problems could be identified and corrected quickly. This also has a disadvantage, since the continuous involvement with each others work might have been unproductive in some moments.

In addition to the communication face-to-face, Slack and WhatsApp was used to communicate with each other. This regarded mostly practical aspects like presence and appointments.

8.5 External communication

One of the aspects of the project was communication between the team and other stakeholders. In this project, the main stakeholders were the client, Raccoon Serious Games, and the coach, Prof. dr. M. Specht.

8.5.1 Client

The team did not know the client at the start of the project. Nevertheless, the communication between the team and the client went smoothly from the start. The client provided a clear insight in the world of escape events and in the existing system, giving a solid base for what the requirements for the new system would be. The client was proactive in brainstorming about how the system would best solve the problem of the client and provided access to resources and people which had a connection to the system. For example, in the first weeks the team got to interview two employees who would use the system from different roles.

During the development of the system weekly meetings with the client provided a way for the client to keep up to date with the progress and for the team a way to gather feedback and verify that the product was heading in the good direction. Because most of the work was performed on location at the client, any questions during the development could be asked immediately. This was useful, since the team was therefore not stalled by questions for the client.

In addition, the client provided the ability to test the system in a real-world setting. This resulted in a lot of insights and feedback and was a perfect way to put the product to the test.

8.5.2 Coach

After the coach was found, an initial Skype meeting provided both the team and the coach with a clear insight in the expectations from each other.

Over the course of the project, the team met with the client six times. During these meetings, the team provided an update on the progress and often had several subjects they wanted to discuss with the coach. The coach provided helpful feedback, ideas and comments with regards to where to put the focus.

9 Conclusion

At the beginning of the project. Raccoon Serious Games wanted to have a reworked or upgraded version of their original Massive Online Reactive Serious Escape (M.O.R.S.E) system. This system is part of the foundation of the business operations of Raccoon Serious Games, and is used to host so called Escape Events. In these events, a large group of players, divided into smaller teams, work to solve escape-room-like puzzles on tablets running the M.O.R.S.E system. The progress of the teams can be monitored by staff members within the same system.

When Raccoon Serious Games started to grow, it became apparent that the original M.O.R.S.E system was lagging behind in terms of usability and maintainability, due to the low modularity and flexibility of the code. The code of the system had to be reworked for every event, and this was especially difficult for employees without a technical background.

In ten weeks a new Massive Online Reactive Serious Escape 2.0 (M.O.R.S.E 2.0) system was built entirely from scratch as part of a Bachelor End Project (BEP). During this project, the new system has been designed, implemented, tested and deployed. Furthermore, the system has already been used in a real escape event which was hosted for an actual client of Raccoon Serious Games.

From the highly positive feedback of this event, and the other feedback from throughout the project, it can be concluded that the new M.O.R.S.E 2.0 system has shown great improvement over the original system, when it comes to modularity, maintainability, functionality, flexibility and usability. From the facts that all must haves, which were equal to the functionality of the old system, and nearly all should haves, which were all new functionality, were reached, and the system has already been used successfully, and will continue to be used in the future, we conclude that choosing to start over from scratch was the better option for our BEP, and that this project has been completed successfully.

10 Recommendations

This chapter will discuss which parts of the system might need future improvement, how the system can be extended and give tips on the maintainability.

10.1 Future improvements

This section will provide ideas for future improvements to the existing system. During the development of the system various decisions were made about the way features are implemented. Some of these choices could be made differently with the knowledge after the project. Also, some parts of the system are not built as flexible as others because of limiting factors such as time.

10.1.1 Question types

There currently are two types of questions: open questions and multiple choice questions. While these types both provide flexibility to change the way answers are checked (e.g. ignoring whitespace or allowing multiple answers to be correct), the way these types are defined is not modular enough to easily create a new question type. This could be improved to create a more modular structure of question types which would be extendable.

10.1.2 Automated UI testing

The team made the choice not to write automated tests to test the UI. Most of the functionality can be tested manually without much effort. It could be beneficial for the quality and reliability of the system to add automated UI tests for the most crucial parts of the system.

10.1.3 Access control

Since security was not the most important part of the old and also not the new system, the way access control for administrators is implemented is not as robust as it could be. For example, users with a role in one event automatically have this role in all other events. Additionally, logging in is done using the code without additional password verification. This could be changed in the future.

It should be noted, however, that Angular's Router guards provide some layer of security. They ensure that regular players are redirected away from admin pages, and that once a different event is accessed than the one the user belongs, the user is logged out (this applies to all roles, except for administrators, which can manage multiple events). Neither of these situations is likely to occur, but they were taken into account during the development.

10.2 Extending the system

This section will give ideas on how the system can be extended with extra functionality. In the limited duration of the project, not all wishes from the start of the project have been implemented. Also, during the development of the system, new ideas for features came up which could not be implemented yet.

10.2.1 Hint overview

One of the ideas from the start was the ability for teams to look back at hints they received in the form of a hint overview or notification centre. This would allow users to read the hints they received in a later phase of the game, instead of having to ask for the information again.

10.2.2 Game statistics

After the game, the game host presents a couple of interesting statics to the crowd. This includes things like the team who solved a specific puzzle the fastest or which puzzle had the best success ratio. These statistics could be implemented in the system, either as insight for the game host or as views on the projector.

10.2.3 Evaluation

After the game, the players could be asked about their experience in playing the game. This is currently done using whiteboards outside of the room, but this could be integrated in the system. Teams could for example get a question to grade their experience at the end of the game.

10.2.4 Extra rules

The ruleset structure is set up to provide flexibility in creating events and defining the flow during the game. While most of the components of the system are controllable using rulesets, this can still be extended to include more possible rules. For example, rules with regards to stages could be implemented: a trigger when a team attempts a puzzle in a stage or finishes a stage. Also, triggers, conditions and actions with regards to the score could be implemented. This allows for more complex game logic such as score bonuses for the first team that solves a question.

Another improvement for the ruleset structure is a possibility to narrow down how actions affect teams. This could be done by allowing to add conditions to specific actions. An example of what could be done with this is moving all players that solved a specific puzzle to a stage but leaving all other players.

10.2.5 Connecting external websites

With the introduction of the webhook triggers and actions, more possibilities are opened for more immersive games where players might need to access other websites to solve puzzles. For this to work properly, a way to connect external websites to the current system would be needed, i.e. external websites should know which team is logged in. An idea was to setup an endpoint where users could be redirected to a specified URL with their team ID as parameter which can them be used by the external website to call webhooks.

Another option to preserve teams across different websites, is to make the external website also use Meteor, along with a connection to the same database, and pass through the user accounts access token [22].

10.3 Maintainability

This section will give recommendations on the maintainability of the system. During the development, the team used various methods to keep the quality of the code and the usability of the product of a high standard. These methods can also be used when improving or extending the system.

10.3.1 Testing

All back-end code has been tested to ensure that the methods work in the intended way. Although it costs some time to write these tests, it helps with detecting bugs and unwanted behaviour. Therefore it is recommended to keep on testing newly created methods. If this seems too time consuming, writing tests for large methods is highly recommended. In that case smaller methods could be skipped, but there is a bigger risk of unintended behaviour in the code base.

10.3.2 Static analysis

The style of all written code, including test code, was checked using ESLint, using Meteor's recommended configuration. This made sure that the code was written in a consistent way and problems like large methods were avoided. It is recommended to keep on writing code to this configuration to ensure that future code is also of high quality.

10.3.3 Code review

Every pull-request made during this project was reviewed by at least two other team members. This was to ensure that the changes made did not have unwanted side-effects or other problems. Also, the continuous integration tested if all tests still succeeded and no ESLint code conventions were broken. It is recommended that future groups keep on reviewing and have some form of continuous integration to ensure that the master branch is of the highest quality possible. If it is not a viable option to have the code reviewed, it is still recommended to check for failing tests and broken code conventions.

Bibliography

[1] Redactie. Explosieve groei escaperooms in Nederland. *Het Financieele Dagblad*, 2019. URL https://fd.nl/ondernemen/1283944/explosieve-groei-escaperooms-in-nederland.

[2] Juho Hamari. Do badges increase user activity? a field experiment on the effects of gamification. *Computers in Human Behavior*, 71:469 – 478, 2017. ISSN 0747-5632. doi: https://doi.org/10.1016/j.chb.2015.03.036. URL http://www.sciencedirect.com/science/article/pii/S0747563215002265.

[3] ”Harri Sarsa” ”Juho Hamari”, ”Jonna Koivisto”. Does gamification work? – a literature review of empirical studies on gamification. *Proceedings of the 47th Hawaii International Conference on System Sciences*, page 3025–3034, 2014. doi: doi:10.1109/HICSS.2014.377.

[4] ”Christopher Cunningham” ”Gabe Zichermann”. Introduction. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps (1st ed.)*, 2011.

[5] Meteor Development Group Inc. *Meteor*, 2018. URL https://www.meteor.com.

[6] Meteor Development Group Inc. *Blaze*, 2018. URL http://blazejs.org.

[7] MongoDB, Inc. *MongoDB*, 2019. URL https://www.mongodb.com/what-is-mongodb.

[8] Clayton Grassick. *Minimongo*. mWater, 2019. URL https://www.npmjs.com/package/minimongo.

[9] Dai Clegg and Richard Barker. *Case Method Fast-Track: A Rad Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994. ISBN 020162432X.

[10] I. Fette and A. Melnikov. The WebSocket Protocol. RFC 6455, RFC Editor, December 2011. URL https://www.rfc-editor.org/rfc/rfc6455.txt.

[11] Feathers contributors. *Feathers*, 2019. URL https://docs.feathersjs.com/.

[12] Google. *FireBase*, 2019. URL https://firebase.google.com/.

[13] Sacha Greif. What’s Going On With Blaze, React, and Meteor? *Discover Meteor*, November 2015. URL https://www.discovermeteor.com/blog/blaze-react-meteor/.

[14] Facebook Inc. *React*, 2019. URL https://reactjs.org/.

[15] Google. *Angular*, 2019. URL https://angular.io/.

[16] Evan You. *Vue.js*, 2019. URL https://vuejs.org/.

[17] Guillaume Chau. Vue integration for Meteor. *GitHub repository*, 2019. URL https://github.com/meteor-vue/vue-meteor-tracker.

[18] Spec India. React vs angular vs vue.js: A complete comparison guide. *Medium*, Augustus 2018. URL https://medium.com/front-end-weekly/react-vs-angular-vs-vue-js-a-complete-comparison-guide-d16faa185d61.

[19] Meteor Development Group Inc. Comparing performance of blaze, react, angular-meteor and angular 2 with meteor. *Meteor Blog*, July 2015. https://blog.meteor.com/comparing-performance-of-blaze-react-angular-meteor-and-angular-2-with-meteor-c650c913d3f8/.

[20] Meteor Development Group Inc. *Meteor mocha testing*, 2018. URL `https://guide.meteor.com/testing.html#mocha`.

[21] Escape Event. *Photos Escape Event*, 2018. URL `https://photos.escape-event.nl`.

[22] Meteor Development Group Inc. *Sharing Accounts*. URL `https://guide.meteor.com/structure.html#sharing-accounts`.

A
Project Plan

Version 1.2 - June 28, 2019

Matthias Bakker, Ayrton Braam, Wouter Morssink, Tim Nederveen, Alexander Sterk

A.1 Introduction

This project plan defines our bachelor project and describes the objectives of said project. The plan will serve as a guideline during the project and as an agreement between the project team, the client, the coach from TU Delft and the bachelor project coordinators.

In this plan, we will give a description of the company and the events our problem originates from. Furthermore, we will formulate the problem of the client. We will describe the scope of our project. Finally, we will specify the way the project is organised and establish a global timeline for the project.

A.2 Project scope

A.2.1 Company description

Popup-escape is a young company started by a Delft Computer Science student three years ago. They have designed and made more than 50 escape rooms over the past three years and at least 7500 people have played their different escape rooms.

Next to regular, small scale escape rooms, Popup-escape also hosts larger events for groups of 200-400 playing escape rooms. These events generally take place in a corporate setting and are used to convey a message and foster teambuilding.

A.2.2 Project description

During these events, players engage in the story and the puzzles using laptops, tables and mobile phones. This is currently done using the M.O.R.S.E. (Massive Online Reactive Serious Escape) System.

This system started out as a specific solution for one escape room, and grew each time the system was used, adding functionality over time. It currently works sufficiently, but it is not user-friendly for administrators and game-hosts, nor is it modular in the back-end.

Popup-escape is on the brink of expanding and the software needs a 2.0 version. It should be easy to use for new game-hosts, easy to maintain and it should have the ability to dynamically change between different escape-games.

In the first two weeks, we will research the requirements of the system. For this, we will meet with our client. First of all with the owner, who developed the current system and is the most intensive user of the system. Second with other employees who use the system and might have thoughts about how the 2.0 version should look. We will also investigate what technologies are suitable to solve the problem of the client. We will investigate if the 2.0 version should be based on the current code or should be built from scratch.

A.3 Project management

A.3.1 Client

For this bachelor project our client is the company Popup-escape. The client advisor for this project is Jan-Willem Manenschijn. He founded Pop-up Escape a few years ago. Jan-Willem built the current version of the M.O.R.S.E. system. The clients main objective is a ready to use version of the M.O.R.S.E. System which complies with the stated requirements.

A.3.2 Coach

For this project, Marcus Specht will coach us with regards to the interest of the TU Delft. His primary role is to guiding the project team in applying their knowledge within the project. He supports the project team in decisions in software development methodology, approves the project plan and gives feedback during the preparation of the final report and the final presentation.

A.3.3 Project team

The project team consists of five people, all third-year Bachelor Computer Science students. Each team member contributes to all parts of the project, but each member is assigned roles to guide the project in the right direction. The coach meets regularly with the project team to support in keeping the project running and converging to a timely, finished project.

Role	**Member**
Team leader	Wouter Morssink
Lead programmer	Matthias Bakker
Lead UI	Tim Nederveen
Lead testing	Wouter Morssink
Lead code quality	Matthias Bakker
Lead infrastructure	Alexander Sterk
Scrum master	Ayrton Braam
External communicator	Tim Nederveen
Secretary	*Alternating*

Team leader
The role of the team leader is to make sure that all deadlines are reached and every group member performs their tasks.

Lead programmer
The lead programmer is responsible for all decisions regarding the structure of the code base.

Lead UI
The lead user interface is responsible for the decisions regarding the design and implementation of the user interface.

Lead testing
The role of the lead testing is to make sure that the created code is well tested. Furthermore the lead testing is responsible for a test plan and planning live testing.

Lead code quality
The lead code quality is responsible for ensuring that the created code is well-written and is in line with a set of pre-determined guidelines.

Lead infrastructure
The lead infrastructure is responsible for all decisions and issues regarding the pipe-lining, continues integration and version control.

Scrum master
The task of the scrum master is to keep the scrum board up-to-date, to ensure that all issues are resolved properly and to guide the sprints to completion.

External communicator
The external communicator controls all communication between the team and the coach, client and other external parties.

Secretary
The secretary is a rotating function, meaning that each week a different member of the team takes up the role. The task of the secretary is to take notes of the different meetings and to setup the agenda for the next week.

Week	Member
1	Wouter Morssink
2	Tim Nederveen
3	Matthias Bakker
4	Ayrton Braam
5	Alexander Sterk
6	Wouter Morssink
7	Tim Nederveen
8	Matthias Bakker
9	Ayrton Braam
10	Alexander Sterk

A.3.4 Meetings
The bachelor project spans over the course of 10 weeks in which we develop a system for the client. In these weeks there are several different meetings which will be discussed in this paragraph.

Team
As team we decided to meet and work together on a 9 to 5 schedule. This ensures that we keep on track with our schedule and can easily communicate with each other when something is not clear. Apart from the 9 to 5 schedule we allowed each other to have 4 hours a week in which a team member does not have to be present. The team meetings are at the location of the client, namely in the shared room of Popup-escape.

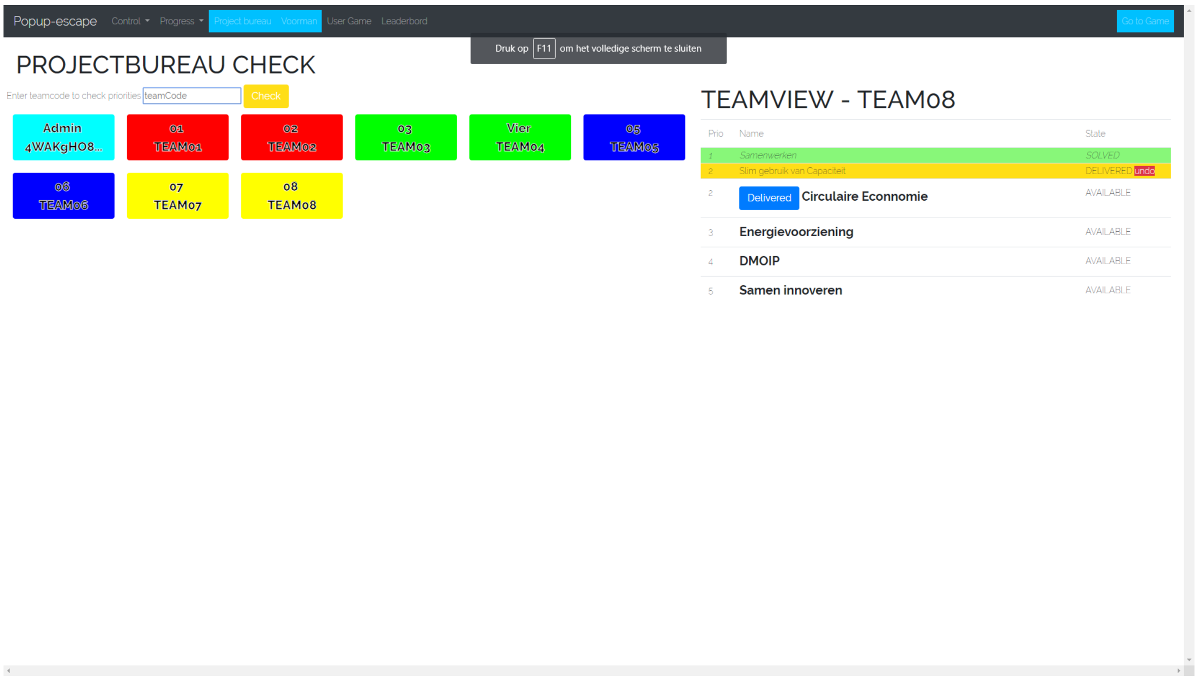
Client Advisor
In the first week we have two meetings with the client advisor, Jan-Willem Manenschijn, the first one before we started exploring the current situation and the second one about expectations. From this point on we decided to have a weekly meeting on each Monday 16:00 to 17:00. At those meetings we will present our progress and planning and if possible a demo, based on this information our client advisor could give feedback.

A.4 Project timeline

Our project timeline consists of tasks and deliverables. The tasks are what we, as a team, will generally do in that week. The deliverables are deadlines which need to be met for the bachelor project. Furthermore, we added planned escape room events, these events are moments when the old M.O.R.S.E. system will be used. However, if we have a working version and our client advisor has trust in that version, it could be used and tested in a real environment.

Week	Tasks	Deliverables
1: 22-4 - 26-4	Creating project plan Start research report	
2: 29-4 - 3-5	Finishing research report Setup tooling Prepare first sprint	3-5: Research report
3: 6-5 - 10-5	Develop first working version: client-side and data model	
4: 13-5 - 17-5	Start of further development of the system (using SCRUM) 16-5: First test opportunity	
5: 20-5 - 24-5		
6: 27-5 - 31-5		First SIG upload
7: 3-6 - 7-6	Finalise development of event ready system Process SIG feedback	
8: 10-6 - 14-6	Prepare system for test event 13-6: Second test opportunity	
9: 17-6 - 21-6	Process feedback from testing	Second SIG upload
10: 24-6 - 28-6		25-6: Report finished 25-6: Info sheet finished 28-6: Final deliverable
11: 1-7 - 5-7	4-6: Third test opportunity	Presentation

B
Old UI



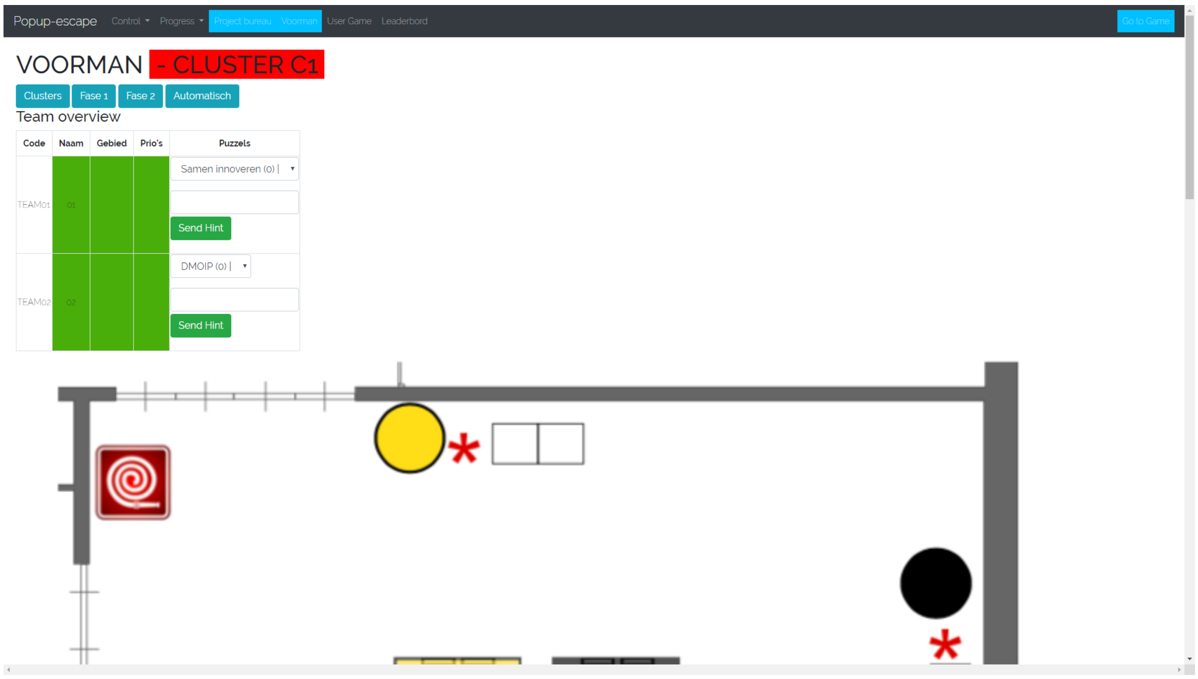
PROJECTBUREAU CHECK

Enter teamcode to check priorities teamCode

TEAMVIEW - TEAM08

Prio	Name	State
1	Samenwerken	SOLVED
2	Slim gebruik van Capaciteit	DELIVERED undo
2	Delivered Circulaire Econnomie	AVAILABLE
3	Energievoorziening	AVAILABLE
4	DMOIP	AVAILABLE
5	Samen innoveren	AVAILABLE

Figure B.1: Screen of the project-desk part

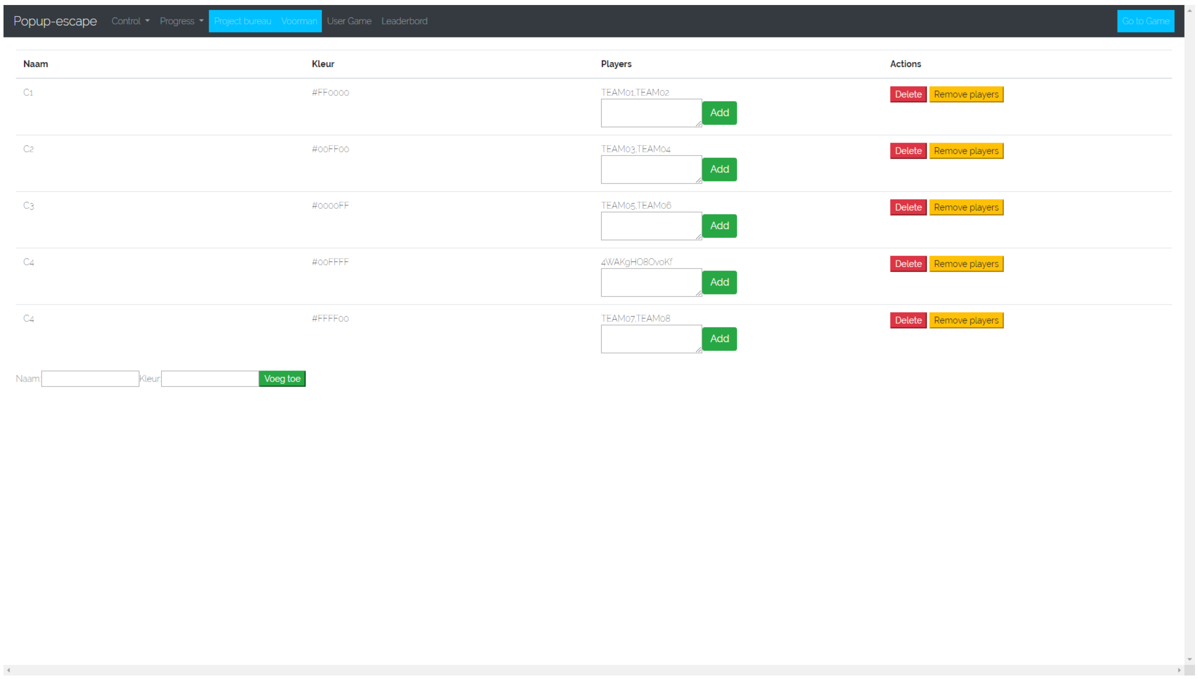


VOORMAN - CLUSTER C1

Team overview

Code	Naam	Gebied	Prio's	Puzzels
TEAM01	01			Samen innoveren (0) | ; Send Hint
TEAM02	02			DMOIP (0) | ; Send Hint

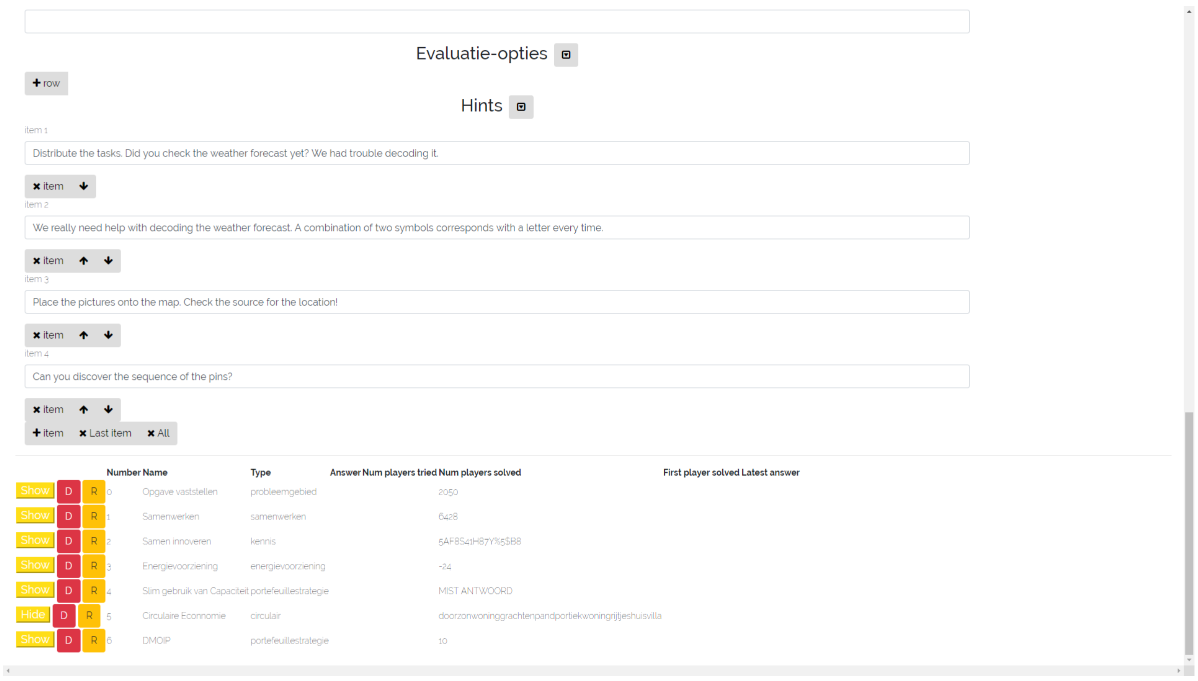
Figure B.2: Screen of the foreman part



Naam	Kleur	Players	Actions
C1	#FF0000	TEAM01,TEAM02 Add	Delete Remove players
C2	#00FF00	TEAM03,TEAM04 Add	Delete Remove players
C3	#0000FF	TEAM05,TEAM06 Add	Delete Remove players
C4	#00FFFF	4WAKgHO8OvoKf Add	Delete Remove players
C4	#FFFF00	TEAM07,TEAM08 Add	Delete Remove players

Naam Kleur Voeg toe

Figure B.3: Cluster overview screen in the admin area



item 1

Distribute the tasks. Did you check the weather forecast yet? We had trouble decoding it.

item 2

We really need help with decoding the weather forecast. A combination of two symbols corresponds with a letter every time.

item 3

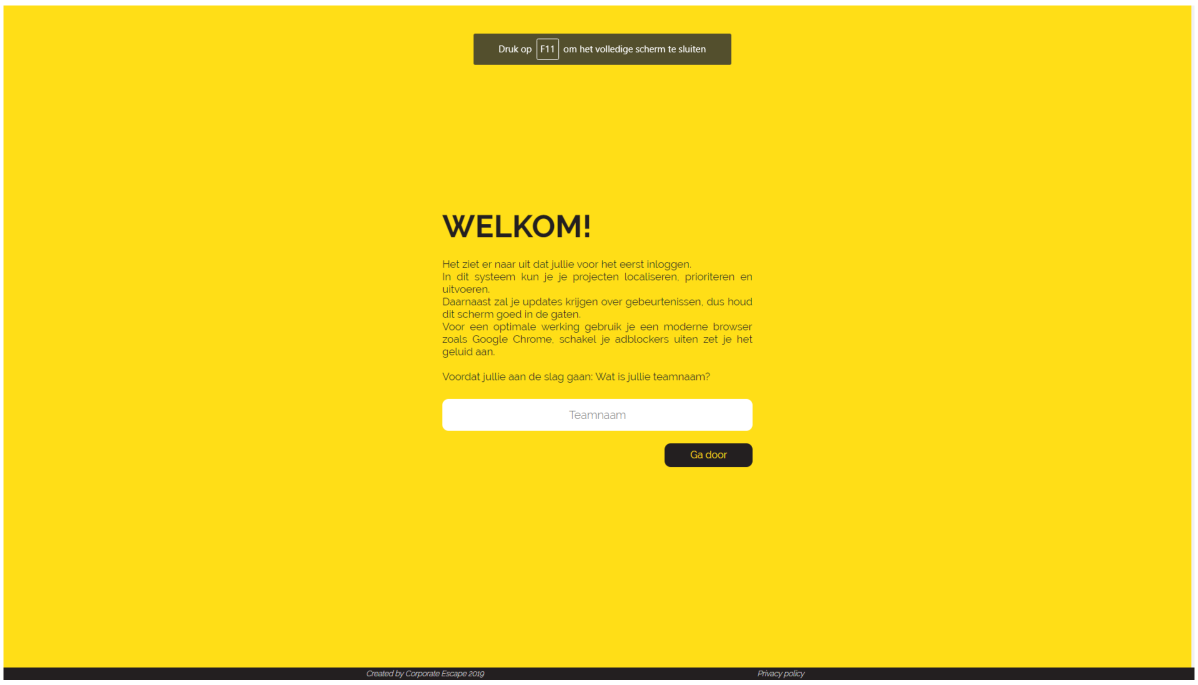
Place the pictures onto the map. Check the source for the location!

item 4

Can you discover the sequence of the pins?

			Number	Name	Type	Answer	Num players tried	Num players solved	First player solved	Latest answer
Show	D	R	0	Opgave vaststellen	probleemgebied			2050		
Show	D	R	1	Samenwerken	samenwerken			6428		
Show	D	R	2	Samen innoveren	kennis			5AF8S41H87Y%5$B8		
Show	D	R	3	Energievoorziening	energievoorziening			-24		
Show	D	R	4	Slim gebruik van Capaciteit	portefeuillestrategie			MIST ANTWOORD		
Hide	D	R	5	Circulaire Econnomie	circulair			doorzonwoninggrachtenpandportiekwoningrijtjeshuisvilla		
Show	D	R	6	DMOIP	portefeuillestrategie			10		

Figure B.4: Puzzle overview in the admin area

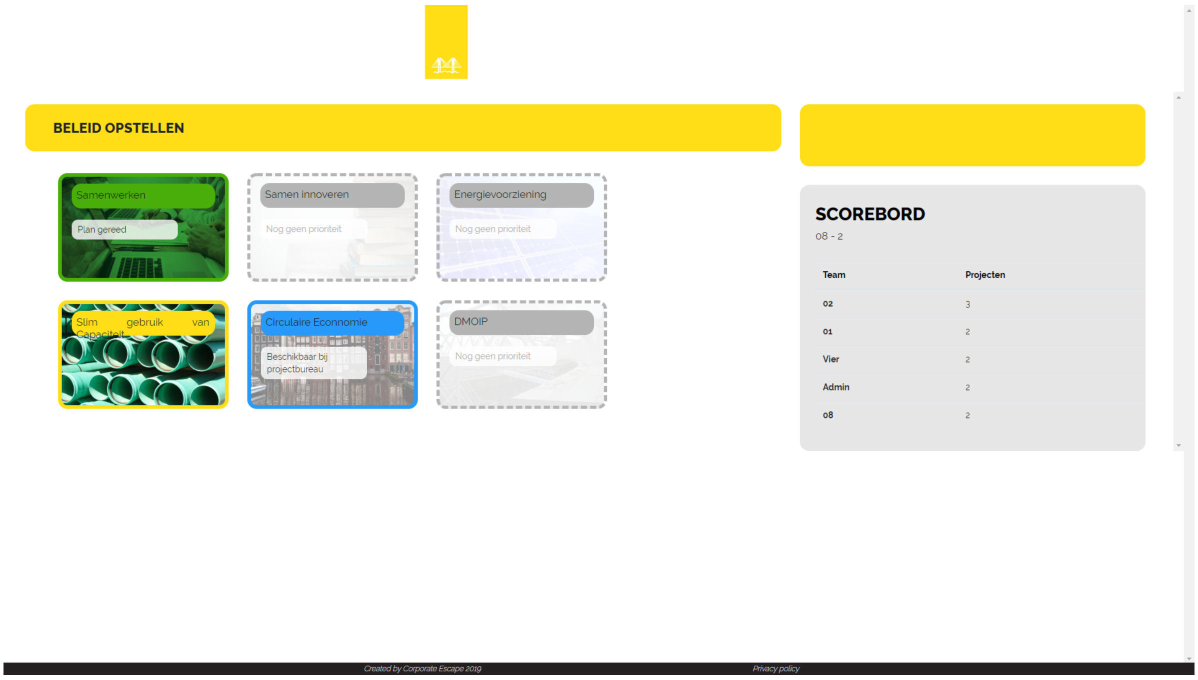


WELKOM!

Het ziet er naar uit dat jullie voor het eerst inloggen.
In dit systeem kun je je projecten localiseren, prioriteren en uitvoeren.
Daarnaast zal je updates krijgen over gebeurtenissen, dus houd dit scherm goed in de gaten.
Voor een optimale werking gebruik je een moderne browser zoals Google Chrome, schakel je adblockers uiten zet je het geluid aan.

Voordat jullie aan de slag gaan: Wat is jullie teamnaam?

Figure B.5: Welcome screen in the client

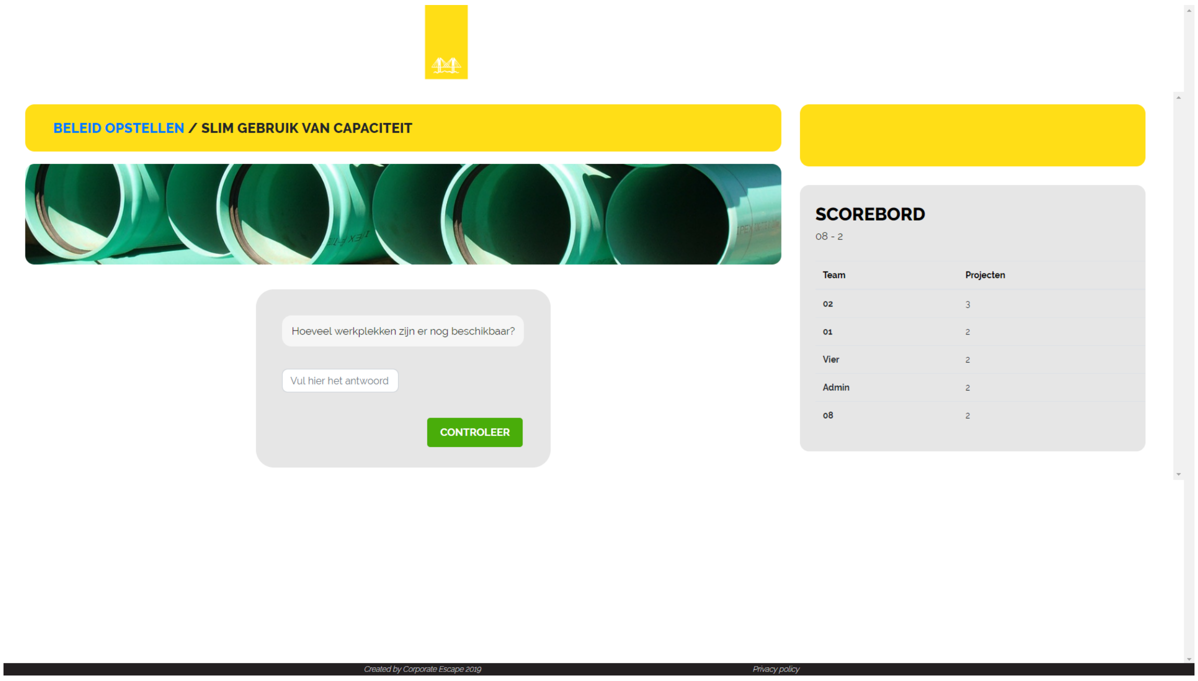


SCOREBORD

08 - 2

Team	Projecten
02	3
01	2
Vier	2
Admin	2
08	2

Figure B.6: Screen of a question menu in the client


BELEID OPSTELLEN / SLIM GEBRUIK VAN CAPACITEIT
Hoeveel werkplekken zijn er nog beschikbaar?
Vul hier het antwoord
CONTROLEER
Created by Corporate Escape 2019
Privacy policy

SCOREBORD

08 - 2

Team	Projecten
02	3
01	2
Vier	2
Admin	2
08	2

Figure B.7: Screen of a question in the client

C SIG Feedback

C.1 First review

De code van het systeem scoort 3.7 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Size en Duplication vanwege de lagere deelscores als mogelijke verbeterpunten.

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.

Voorbeelden in jullie project: - LoginComponent.processTeamCode - GameComponent.ngOnInit - TeamsComponent.drop

Bij Duplication wordt gekeken naar de hoeveelheid gedupliceerde code. We kijken hierbij ook naar de hoeveelheid redundantie, dus een duplicaat met tien kopieën zal voor de score sterker meetellen dan een duplicaat met twee kopieën. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om de hoeveelheid gedupliceerde code zo laag mogelijk te houden. Na verloop van tijd zal de gedupliceerde code moeten worden aangepast. Dit leidt niet alleen tot extra werk, aangezien op dat moment alle kopieën tegelijk moeten worden veranderd, maar is ook foutgevoelig omdat de kans bestaat dat één van de kopieën per ongeluk wordt vergeten.

Voorbeelden in jullie project: - morse-2-master/client/imports/app/player/pages/game/game.component.ts en morse-2-master/client/imports/app/projector/parts/timer/timer.component.ts

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

D
Configuration

In this appendix is an in-depth documentation of the configuration.

D.1 Back-end

The set configuration of an event is stored in a subdocument of the event itself. However, the overall structure of the configuration is defined as a single JSON object, which is interpreted by Astronomy to create a similar structure in the database. The Config object has the following properties: ‘options’, ‘get’, ‘set’ and ‘getAstronomyFields’.

options

The options property is another JSON object which defines the structure of the configuration. Its keys represent categories and the values are JSON objects with every key representing the name of an option within the category, and the value another JSON object for the properties of the option.

An option has one required field: ‘type’. This field defines what kind of data the set value should be, such as: ‘boolean’, ‘number’, ‘date’ or ‘string’, etc. The rest of the fields are optional:

- ‘allowed’: An array of JSON objects, which are the only values this setting can be. The keys of these objects are a textual description of the given value. This property is mainly used to generate dropdown selectors in the configuration editor.
- ‘clientSide’: A boolean stating whether this option is available in the player screen. Some options are irrelevant to the players and Config.get() will always return undefined for them, if the caller does not have a Staff role.
- ‘default’: A default value. If this option is not set, this value will be used.
- ‘description’: The description of this option for the editor.
- ‘integer’: Whether the value should be an integer, instead of any number.
- ‘min’: The minimum value, if the type is ‘number’.
- ‘name’: The name of the option, for the editor.
- ‘needs’: A JSON object with ‘option’ representing an option in the same category and ‘values’ an array of the possible values. This is used in the editor to disable certain options if they require other options to be set (to a certain value).
- ‘optional’: Whether this value is optional, i.e. allowed to be undefined.

Adding new configurable options is as simple as adding another category with a new option (or adding a new option to an existing category), and setting the required fields.

get
A function requiring an event ID, category and option name. This will query the database for the value of the option in the given event, and return it. If it cannot be found, it will return the default or ‘undefined’, if the value is optional or not clientSide while you are not logged in as a Staff member.

set
A function requiring an event ID, category, option and new value. This will set the given option to the new value in an event.

getAstronomyFields
A function that interprets the ‘options’ into a structure understandable by Astronomy[1]. Astronomy acts as a Model Layer between our database and application. Astronomy also provides the validation of the configuration, such as options being the proper type in the database, and only allowing accepted values from the ‘allowed’ property.

D.2 Front-end - Implementation

In the front-end, ‘Config.get()’ is used to get all the different options and apply them where needed. For example, for the strings category, the configuration is accessed directly in the Angular templates. For the style category, however, first CSS variables are defined in the main HTML file, which are set/overwritten in the Angular Component.

D.3 Front-end - Editor

Because the configuration back-end is so modular, the configuration editor (figure 7.15) is designed with this modularity in mind.

On the left there is a list of the categories, acting as tabs in the editor. On the right there is the actual editor, which is a list of all the options in the selected category. To generate the editor, ‘Config.options[category]’ is interpreted into a list of options and their properties. For example, the ‘type’ property defines what kind of input field is used in the editor, and ‘needs’ is used to mark input fields as disabled if their conditions are not met.

The editor supports the following types:

- ‘color’: Gives a colour input, and stores the value as a Hex colour.
- ‘number’: Gives a number input. Can be used in combination with the ‘integer’ and ‘min’ properties.
- ‘date’: Gives a date input.
- ‘url’: Gives a URL input.
- ‘boolean’: Gives a checkbox.
- ‘choice’: Gives a dropdown.
- ‘numberChoice’: Gives a dropdown of numbers.

Any other type will give a regular text input field.

In the editor, changes to the back-end are only made once the save button is pressed. Different options can be changed at the same time, but the ‘Save’ button will not appear until a single change is made. There is also a reset button, to cancel any pending changes.

When the editor is loaded, the current state of the configuration is saved and compared to the state whenever a change to the event is detected. If the two states are not the same, the changes made in the editor become invalid and the user needs to refresh the editor. This implies that two people cannot edit the configuration at the same time, because if one person saves their changes, the other person’s

[1]`http://jagi.github.io/meteor-astronomy/`

editor will become invalid. This is intended, since doing it this way is much easier than tracking which options have changed between event updates. Furthermore, the probability of two people changing the configuration at the same time is rather low, as it would imply multiple game designers/hosts.

The editor itself does not provide any input validation, besides the validation provided by the HTML5 input tags, such as checking that the value of a URL input field is actually a URL. The back-end takes care of validation, and if there is an error, the editor will display an error message and needs to be refreshed.

D.4 ConfigAction

Because of Meteor's realtime functionality, changing an option and saving the changes will immediately update and apply the new value(s) everywhere. For example, a change to the theme colour will immediately become visible to the players. For a more dynamic approach to changing the configuration, rulesets (see 5.7.1) with a 'ConfigAction' can be used.

A ConfigAction takes a category, option and new value, and when executed will set the option in the category to the new value. However, the names of options as they are displayed in the editor and how they are in the back-end varies. As such, here is a list of all the categories, options and their types that are available in the configuration.

- style
  - themeColor : colour (string of colour in Hex format)
  - backgroundColor : colour
  - backgroundUrl : url (string of url)
  - textColor : colour
  - font : string (comma separated list of font names)
  - projectorBackgroundColor : colour
  - projectorBackgroundUrl : url
  - logo : url
  - checkColor : colour
  - projectorAmount : number (allowed values: 5 & 10)
- hint
  - popupBackgroundImage : url
  - popupBackgroundPosition : string (CSS: background-position)
  - popupBackgroundSize : string (CSS: background-size)
  - popupBackgroundColor : colour
- timer
  - type : string (allowed values: hms, ms, date, none)
  - secondsPerDay : number
  - endDate : date (JavaScript Date object)
- login
  - codeLength : number
  - allowAllTeams : boolean
- foreman
  - warning : number (min = 0)
  - danger : number (min = 0)
  - endTime : number (min = 0)

  - puzzleBook : url
  - roomMap : url
- strings (all are type string)
  - remainingTime
  - currentDate
  - leaderboard
  - score
  - check
  - noAnswer
  - notCorrect
  - solve
  - projectDesk
  - solved
  - goodJob
  - atProjectDesk
  - goToProjectDesk
  - locked
  - solveOthers
  - hintReceived
  - openHint
  - read
  - team
  - overview
  - overviewHeader
  - teamCode
  - teamName
  - start
  - notAValidCode

E Feedback test event, 13 June

E.1 Tijdens event

- Teams verwijderen
- Teams verbergen
- Game control onder timer plaatsen
- Het sorteren op puzzlestatus gaat fout met opgehaald / niet opgehaald
- We kunnen bij het sorteren op puzzlestatussen bij gelijke status ook de tijd meenemen
- Projectbureau update niet altijd als mensen een puzzle beschikbaar krijgen
- Projectbureau is traag met laden van teams

E.2 Feedback session

- De undo knop in het project desk scherm verschijnt te snel => per ongeluk weggeklikt wordt
- 'Tijd sinds' verbergen na x uur
- Toggle selected clusters
- Bij meerkeuze is niet te zien welke optie is geselecteerd (alleen index)
- Hint bij verkeerde selectie rood in balk
- Goede antwoorden zijn niet te zien in de detail pop-up.
- Twee rijtjes projector: top x / twee rijtjes
- Antwoorden zijn lang in team detail popup
- Mensen kunnen brute forcen (maakt niet uit)
- Clusters sorteren op clusternaam in project desk: sorteren binnen kleur
- Laatste antwoord bovenaan
- Hoeveel items af te leveren
- Active kleur van puzzle in popup

Supporting large-scale escape events with a modular system

Making the M.O.R.S.E. system modular

Raccoon Serious Games hosts so called escape events. These events are similar to an escape room in which teams work together to solve puzzles, only on larger scale. Different kinds of challenges arise when hosting these events, such as monitoring the progress of the teams and managing collection of physical puzzles. For an escape event, different kinds of staff help the teams to have the best experience possible. The teams need to be able to submit their answers to puzzles, while the staff should provide them with the right materials and monitor their progress. To facilitate these needs, Raccoon Serious Games created the M.O.R.S.E. (Massive Online Reactive Serious Escape) system. However, the old system lacked in modularity and flexibility, while having the event hardcoded. These were the main challenges of the project.

To solve this problem, we created a new system to improve the lesser points of the old system while also improving the usability and and maintainability. Over the course of ten weeks, we researched and created the M.O.R.S.E. 2.0 system. The first part of the project consisted of researching the previous system to find the requirements for a new system. After that, a new system was designed and implemented from the ground up using Scrum.

M.O.R.S.E. 2.0 contains all the functionality of the old system, while also improving the structure of the administrator panel. Furthermore, the final product comes with several editors, allowing for designing the puzzles, schedule and logic in the form of rulesets for an escape event and allowing multiple events to be created and managed. Furthermore the configuration editor allows the employees of Raccoon Serious Games to adjust different types of settings and cosmetics. A possible extension would be different kind of puzzles to allow for even a more diverse event. The product was successfully tested during an escape event and will be used by Raccoon Serious Games for future events.

Project team members

Ayrton Braam
Scrum master & back-end developer
Enjoys Arbitrage cryptocurrency bots, Dungeons and Dragons, Formula 1 and Boxoffice tracking

Matthias Bakker
Lead programmer, lead code quality & back-end developer
Enjoys programming, gaming and football

Wouter Morssink
Team leader, lead testing & back-end developer
Enjoys playing draughts, going to the movies and computer games

Tim Nederveen
Lead UI, external communicator & front-end developer
Freelance webdeveloper, enjoys watching series

Alexander Sterk
Lead infrastructure & front-end developer
Enjoys playing Smite and watching superhero movies & shows

All team members contributed to preparing the final project report and presentation

Client

Jan-Willem Manenschijn
CTO of Raccoon Serious Games

Coach

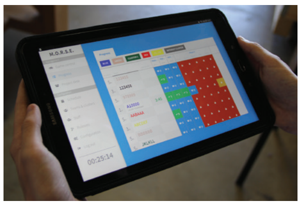
Prof. dr. Marcus Specht
Professor for Digital Education, TU Delft
Director Leiden-Delft-Erasmus Center for Education and Learning

Contact

Wouter Morssink
woutermorssink@hotmail.com

Tim Nederveen
tim@tim365.nl

Alexander Sterk
ajhsterk@gmail.com


Progress overview

System being used at an escape event

RACCOON SERIOUS GAMES

TU Delft

Final presentation date: July 5th, 2019

The final report for this project can be found at repository.tudelft.nl

Glossary

AJAX Asynchronous JavaScript and XML. 20

API Application Program Interface. 20, 25

BEP Bachelor End Project. iii, 1, 5, 45, 69

CSS Cascading Style Sheets. 21, 22, 35, 47, 88, 89

DOM Document Object Model. 11, 26

HTML HyperText Markup Language. 11, 21, 88, 89

JSON JavaScript Object Notation. 10, 11, 28, 87

M.O.R.S.E Massive Online Reactive Serious Escape. v, 5, 8–12, 21, 45, 61, 62, 64, 69

M.O.R.S.E 2.0 Massive Online Reactive Serious Escape 2.0. v, 25, 26, 28, 35–38, 50, 61, 62, 64, 69

Meteor Open source JavaScript platform for developing web. 11, 21, 25

NPM Node.js package manager. 11, 20, 21

ODM Object Document Mapping. 28

OOP Object-Oriented Programming. 28

PWA Progressive Web App. 30

RPC Remote Procedure Call. 25, 32

SIG Software Improvement Group. 17, 42, 43

SMS Short Message Service. 9

UI User Interface. 8, 10, 31, 32, 63, 66, 71