BACHELORPROJECT

Hunting Happy Families
Final report

Ruben Band,
Koen du Buf,
Bob Dorland,
Quintin van Leersum,
Emma Sala

TI3806 Bachelorproject
Faculty of Electrical Engineering, MAthematics and Computer Science
Delft University of Technology
July 14, 2020

Presentation:	Thursday July 2, 2020	
Client:	Marcel de Bruin	Cityhunter
Supervisor:	Gosia Migut	TU Delft
Coordinator:	Otto Visser	TU Delft
	Huijuan Wang	TU Delft
	Thomas Overklift	TU Delft

Preface

This is the final report of the TI3806 Bachelor Project course for Computer Science and Engineering at Delft University of Technology. This report describes the development of the game Hunting Happy Families, created for the company named Cityhunter. The project lasted ten weeks, in which a digitized version of the game has been made for the Android platform.

This project could not have been done without the help of different people, including Marcel de Bruin, who created the original game in the first place and provided the team with his opinion on different matters in gameplay and design. Secondly, the wonderful Gosia Migut, the supervisor who provided the team with advice and encouragement. Next, the developers would like to thank every single person that helped test the game by playing it against each other and thereby discovering small details that could be changed in order to improve the design and experience of playing the game. Lastly, the team would like to thank the coordinators of the course for regulating the project and answering every question asked.

Summary

Cityhunter developed the idea of transforming their card game, Hunting Happy Families, into a game that can be played on tablets in a same manner that their other game, Hunt the Hunter, can be played. The aim of this project is to develop a game that can be played in Schiedam on an Android platform. An additional requirement is that the game can be adapted after the project is finished. That is, the client must be able to add new card sets and new locations through an interface without having to make changes in the codebase. Another part of this project includes a game master interface, in which one person can keep track of all teams in a game, including their cards, points, locations and chat messages. Lastly, a server was created for storing and sending information between teams and the game master.

This report defines the standard game play of a game of Hunting Happy Families. The main goal of the game is to learn about the history of the city of Schiedam. By walking around, teams can collect cards from other players, and identify locations associated with the various sets to try and gather as many points as possible. The team with the most points at the end of the game, wins.

The system features a server connecting the teams and the game master using a string message system and lambda functions. Upon sending a message using this system, the server handles the requested data, and answers the request by forwarding the provided information, or relaying the information requested.

The system was thoroughly field tested, including a playtest with external players. The verdict on the game by these players is positive, though the system needs a slight amount of fine tuning to ensure the best experience. That said, the game is completely ready for use, and Hunting Happy Families is bound to carve its niche in the market of group events.

Contents

1

Introduction

Created in 2017, Cityhunter is a small company based in Schiedam, which focuses on creating enjoyable experiences that allow you to learn about the city. To do so, the company created the game Hunt the Hunter, in which you have to find your target based on clues collected throughout the city. Now, Cityhunter wants to expand their list of games with a new game inspired by Happy Families, called Hunting Happy Families.
Hunting Happy Families, from here on referred to as HHF, is a card-based variation of the game Happy Families, in which players compete against each other through collecting sets of four cards. In HHF, teams gather and trade cards with the other teams in order to find landmarks in Schiedam with the clues on the cards. By collecting more cards of the set, it becomes easier to hunt down the location. Teams are competing against each other by earning points through completing sets and finding as many locations as possible.
Initially, the game was playable with physical cards using QR codes and websites. Teams would start with eight cards and traded a card with any team they come across. When identifying a location, the team would take a selfie and save it on their phone. At the end of the game, the game master reviewed the hands and images the teams have, and awarded points accordingly. Instead of playing this game with real cards, Cityhunter wants to create a mobile application that is playable on Android.

This project intends to create this game from the ground up. What that means is first creating a design based on the ideas provided by the client, and then implement said design into a well-functioning system, after which it is tested and finally released for public play. The basic structure of the created system can be found in figure 1.1.

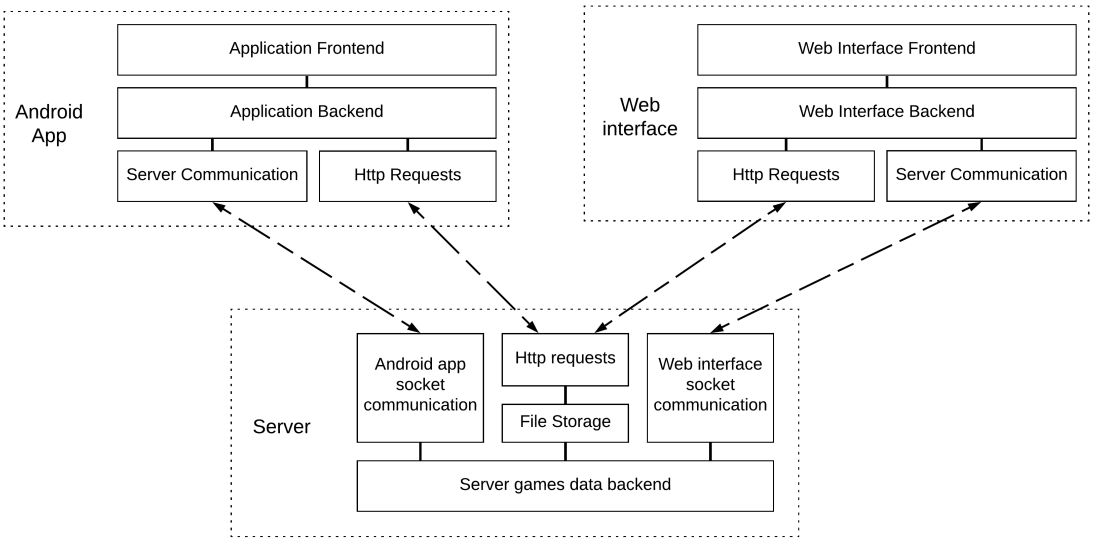

Figure 1.1: Overall structure

As further described in chapter 4, this product consists of three modules: The android app, the web interface and the server.
The android app and web interface each consist of a front- and back-end, the latter of which includes functionality for server communication and sending and receiving HTTP requests. The server includes a socket for communicating with the various android applications, a socket for facilitating the web interface, and a back-end functionality which includes file storage and processing the sent HTTP requests and socket communication.

1.1. Document structure

This document will first provide the problem statement in chapter 2. Afterwards, the design of the game is presented in chapter 3. Next, the implementation of this design is detailed in chapter 4, after which the testing of the implementation is defined in chapter 5. Then, the final product and the overall process are evaluated in chapters 6 and 7. Finally, future recommendations for this project will be given in chapter 8, and a conclusion will be provided in chapter 9.

In appendix A, the research report made in the first two weeks of the project can be found, including its respective appendices. This is then followed by appendix B, which goes over each requirement as decided upon in the beginning of the project, explaining what requirements have been met, why some haven't been met and why and how others have changed since their creation. Appendix C contains the project description as found on Bepsys, and lastly, appendix D showcases a quick and simple info sheet of the project.

2

Problem statement

The challenge of turning Hunting Happy Families into a digital game which can also be managed by the game master consists of three main parts.

First, The game requires development of a mobile application. This is the app used by the players of the game, containing the functionalities that the players can enjoy. The app should be intuitive, so no problems arise when playing.

Second, The game master requires the creation of a web interface. This interface needs to have an simple structure and must be compact, so that the interface is intuitive for the game master, which then leads to efficient management of games.

Finally, the two discussed components need to be linked together via a server. Data required by the game master or a team should be available to these respective parties, such that they can easily communicate with each other.

3

Hunting Happy Families: The game

HHF is intended to create a fun and educational experience for people of all ages. This chapter explores how a game of HHF is played, and how the game's design has progressed into the game play realized in the final product. This includes the initial design and the changes during implementation.

3.1. Gameplay

This section defines a standard playthrough of a game of HHF, starting at the creation of a game, until its deletion. A game of HHF involves two parties: The teams and the game master. The teams, the amount of which ranges between 4 and 10 per game, try and achieve the goal (3.1.1) of the game as best as possible. The game master is a single person responsible for managing the ongoing game.

3.1.1. Winning the game

By walking around the city the game takes place in, collecting cards from other players, and identifying locations associated with the various sets, teams try and gather as many points as possible. In doing so, the teams can learn about the history of the city. The team with the most points at the end of the game, wins.

3.1.2. Before the game

Before starting a game, a game must be created. A game consists of the following elements:

- The time the game will last;
- The amount of teams the game will have at most;
- The amount of points a set is worth;
- The amount of points identifying a location is worth;
- The amount of cards a team obtains at the start of the game;
- The city the game is played in;
- The sets to be used in the game.

Most of these are predetermined for the game master, but are customisable at their discretion. Once the game has been created, the game master can start managing this game, and teams can start joining
Before starting a game of HHF, all teams are provided with a tablet with the application of HHF installed and loaded. The game master has created a game and has opened it up for teams to join. Teams cannot exit the app. The teams type their name on the provided tablet, as is seen in figure 3.1. Once a team has decided upon a name, they are then sent to the main menu when the team has been added to the game. An example of this can be seen in figure 3.2.
Teams are given a predetermined amount of cards, eight by base, and these are added to their collection. A team can never obtain a complete set from the start. Remaining cards are added to the stack.

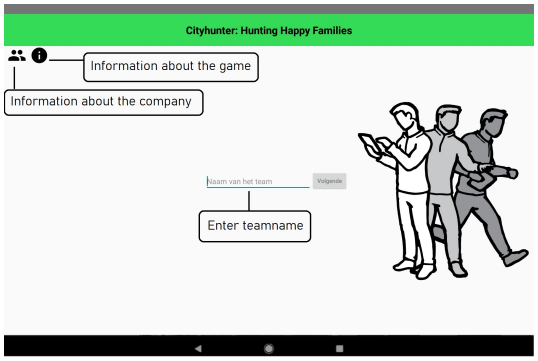

Figure 3.1: Welcome screen

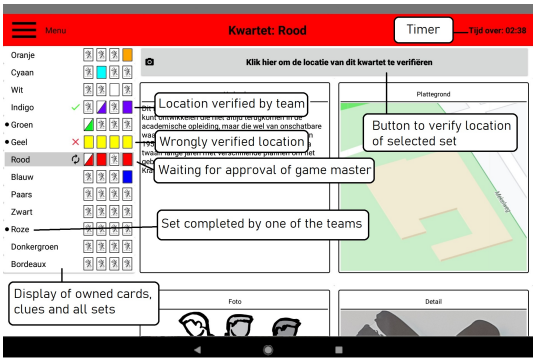

Figure 3.2: The main screen, which showcases the currently owned cards and information

3.1.3. Navigating the menu

While the game has not yet started, and even once the game has commenced, teams can browse through the menu, as showcased in figure 3.3. This includes a screen for the map of the play area (3.4), a chat screen for communicating with other teams and the game master (3.5), a screen showcasing the ranking of the teams and any activity that has happened between teams (3.6), an FAQ screen, and the options to trade with other teams and to verify locations. The latter two are still disabled while the game has not started.

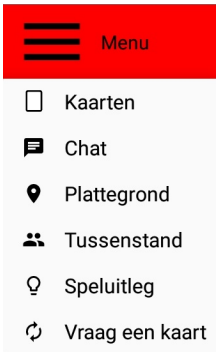

Figure 3.3: The menu on the main screen

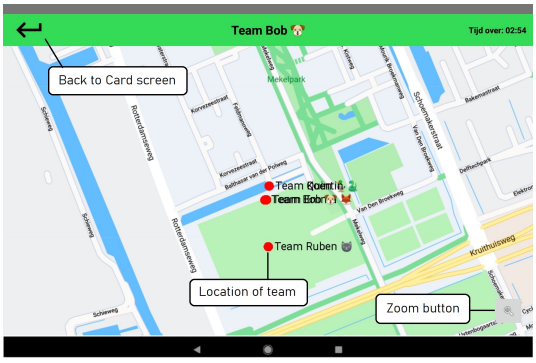

Figure 3.4: Map screen

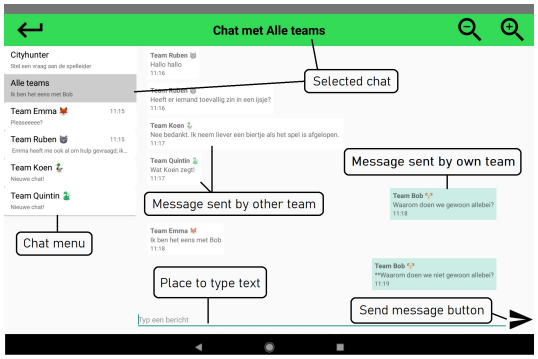

Figure 3.5: Chat screen

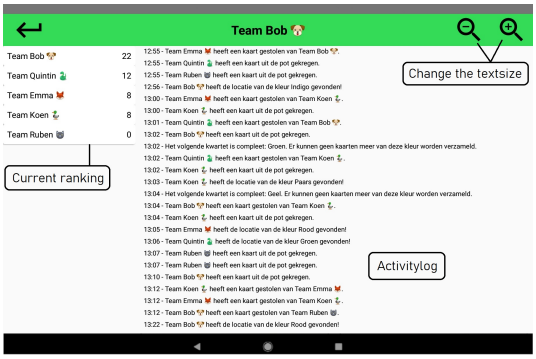

Figure 3.6: Ranking screen

3.1.4. Starting the game

Once all teams have joined a game, the game master can start the game. The game timer will start, teams will receive a pop-up and can start walking around the city. This also means they are now capable of getting new cards, and getting points through completing sets and verifying locations.

3.1.5. Getting cards

Any team can obtain new cards by 'trading' cards with another team. When two teams are within range of one another, are not already in a trade, and have not recently traded with each other, they can perform this trade. They select the team in question (3.7), and are then directed to the trading screen (3.8). However, a team can also refuse to ask the other team for a card when it is asked to trade. This does not stop the other team from asking a card back. If both teams decide to trade, each team selects a card that they do not own. This card has to be of a set of which they already have at least one clue. If the other team owns the card that is asked for, they 'steal' it from that team. If the other team does not own that card, this team instead draws a card from the stack.

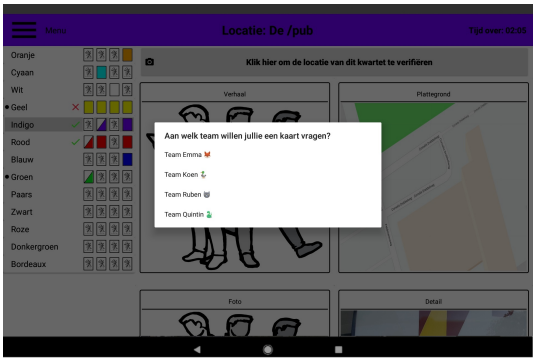

Figure 3.7: Popup to start a trade

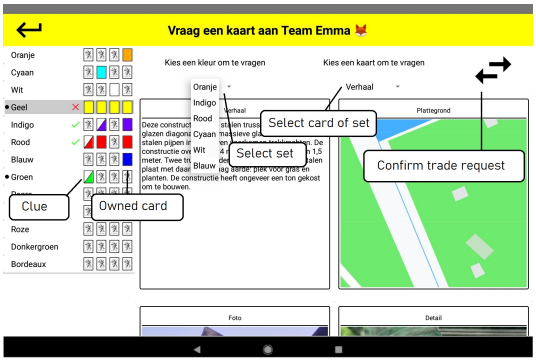

Figure 3.8: Trading Screen

The information of a card

Cards that have been stolen from a team are no longer owned by that team. However, the information contained within those cards is kept by the team for the rest of the game. Those cards are referred to as clues. This way, the knowledge of stolen cards can still be utilised for identifying locations.
As soon as the stack is empty, instead of receiving a card from the stack after asking for a wrong card, teams receive a clue that they don't yet have. Consequently, teams can keep on puzzling.

3.1.6. Getting points

There are two methods of obtaining points: completing sets and verifying locations.

Completing a set

Completing a set means owning all four cards of a color. Only having the information of a card does not count as owning a card. Once a team has completed a set, the information on the cards can no longer be distributed among other teams.

Verifying a location

The main source of points, however, is verifying locations. To verify a location, a team selects to verify a location in the main menu of the selected color (3.2), take a picture of themselves at said location, and then send that image to the game master (3.9 and 3.10). The game master then approves the picture if they can confirm the location is correct. If the photo is approved, the team is awarded points depending on the amount of cards and clues they needed to verify the location. That is, for every clue a team owns, they are awarded less points when verifying a location.

3.1.7. Finishing the game

In the last thirty minutes of the game, the ranking is made invisible for all teams. Once the game has ended, teams can no longer verify locations or ask for cards. However, teams can still use the chat to communicate with each other. The end of a game is announced to teams through a pop-up asking them to return to the original starting location to gather with the game master. The game master then announces the final scores, and the game is ended. The game master is then free to delete the game from the server.

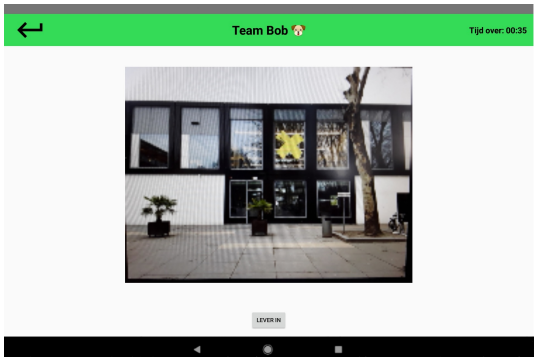

Figure 3.9: Image verifying screen

Figure 3.10: Game master interface

3.2. Initial design

Prior to implementation, a few alternative ideas were considered for the final product. This section will explore these initial ideas and why these were ultimately not decided upon.

Disabling the card screen

In the very first iteration of the design, the team considered the possibility of removing the ability to see your cards when nearby another team. This was to prevent possible cheating by sharing the screen with the other team. When proposing this to the client, he noted that he would prefer to not see this functionality. In retrospect, this design choice would likely interfere with the intuitiveness of the product.

Location verification

In the same iteration of the design, the product intended for locations to be verified using the coordinates of the team that wants to verify a location. This way, the game could be run without any required influence of the game master, aside from the chat system. However, the client made clear that he does want the game master to be active during the game, and liked the aspect of sending selfies to the game master from his non-digitized version. Thus, the team changed the design to reflect this requested change.

3.3. Changes during implementation

During the implementation phase, there were three other major design choices executed. These were separating the card screen from trading and verifying, as well as a structure change for the home page, and finally, the addition of other cities as potential locations for the game.

Separating trading and verifying

The functionality for trading and verifying images would initially be performed on the same screen as the trading screen. Both trading and verifying would always be available on the screen. Trading would only be enabled if a team was close enough, and verifying was possible at all times. This design choice partially stemmed from the first design, where teams would still verify using the location of their device, rather than a picture.

During implementation, it became clear this design would lead to a cluttered card screen, which in turn would make the app more difficult to use. To remedy this, both trading and verification were moved to separate screens, which could initially be invoked from the cards screen, and, at the final design, be selected from the main menu.

Front end restructure

The final design of the application's interface has had another small but significant change compared to the design envisioned after the research phase. Initially, a main page was created (3.11), with the intent of adding a map to that page. On it, there were menu options redirecting to a page for the cards, chat, map, ranking & activity, FAQ and camera page respectively. When showcasing this to the client, he instead requested a removal of the main menu, and to change the card page to be the main page instead (3.2). This way, the main menu was moved to a toggleable sidebar, since the used design of the card page, as provided by the client, had little room for a menu on the page itself.

Other cities

Lastly, the game has been expanded such that it can be played in Delft as well as Schiedam. Initially meant for testing purposes, this design choice showcases the capability for the game to be deployed in any city, as the creation of a new game is only dependent on creating a new map and new sets of cards for locations of the city.

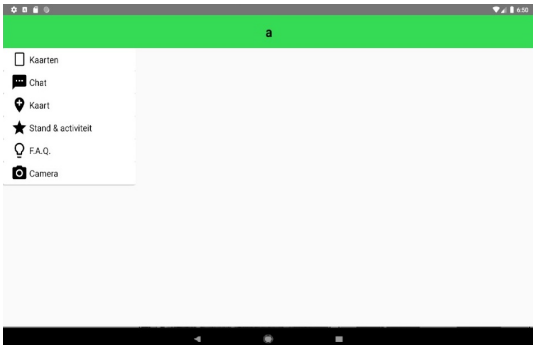

Figure 3.11: Old main screen

4

Implementation

To implement the game as set up in chapter 3, three system components must be created: The application, the web interface and the server. This chapter delves into the required functionality and the implementation of each of these aspects.

4.1. The server

To properly centralize data, as defined as an expectation in section A.3.3, there needs to be a central location to handle the process of a game, from start to finish. To do so, a central server is created for the game. This server, written in JavaScript, is the nerve center of the game. All information of past, present and future games that have been created, is stored here. Any communication between teams and with the game master is accomplished through the server as well. In short, the server is responsible for the following aspects:

- Managing the information of all games, the game master and its teams. This includes:
  - The location, sets, duration and amount of teams in a game;
  - The team names, points, collected cards and clues, locations, ongoing and recent trades, and recently sent pictures of each team;
  - Communication between teams and/or the game master;
  - The ability to trade between teams.
- Providing all aforementioned information to the client and the game master, when needed.

Generally, the server does not perform anything without being invoked. If a team provides or sends information, the server handles it accordingly. If a team or the game master requests information in a valid manner, it also provides the requested data.
This section further explains how connection with the server can be invoked. In chapter 5, the utilisation of the created error log will also be detailed.

4.1.1. Connecting with the server

Upon calling the server.js file, the system sets up a server on the designated IP address. Afterwards, an IPv4 socket is set up for players to connect to, and a different IPv4 socket is opened up for the game master to interact with. Once these connections are set up, the server is capable of lasting as long as required.
Sending and receiving messages requires specific message formats, depending on the prefix. The prefixes available also vary based upon the sender (game master or player), as both of these have different functionality required for their respective systems.
After a message is sent, its prefix is compared to the list of keywords defined in the system. The associated lambda function is then called on the server to handle the message sent, if all fields are in accordance with the expected format. Thanks to utilising the lambda function, new keywords can easily be added without increasing unit size and code complexity.
A full explanation of the message sending functionality is explained in the next section, 4.2.

4.2. The app

Teams must be capable of playing the game to its fullest extent, having the most optimal experience in doing so. For this reason, the application is the most crucial element of the system. The application of Hunting Happy Families is created for the tablets provided by the client, but will work on any Android device provided it is running on the required API or higher. The app is written in Kotlin, and uses XML for its front end functionality. Preferably, the application created provides little inconvenience and instead augments the players' experience. That is, the application is easy to understand, and players can focus entirely on playing the game through the created interface.

The client side functionality has the following responsibilities:

- Showcase the user interface. That is, provide an interface for:
  - Managing cards and clues;
  - Trading;
  - Verifying locations;
  - Chatting;
  - Showcasing the playing field and player locations;
  - Showcasing the ranking and activity log.
- Communicate changes to these elements to the server.
- Showcase sent changes from the server.

Whereas it might be the responsibility of the client side to provide an interface for the cards, trading, chatting and so on, it is the server that has to keep track of all the data. This section highlights the connection between the server and the application interface.

Every change that has been made through an action by one of the teams is sent to the server as a string. The string contains a keyword for identifying what action the server needs to take, followed by the required information. This information often includes the id of the team that went through the action, as well as the id of the team that will be affected by the action.
For example, when sending a chat message to a team, the server requires information on the sender, receiver and content of the message as well as the time it was sent. When the information has been received by the server, the server takes the appropriate actions to pass on the chat message to the other team.
Another example is when team A asks team B for a card. Here, both team ids are once again necessary, alongside the details of the card that has been asked. When the server receives this information, it checks whether team B indeed has this card. If they do, the server removes the card from team B's owned cards, and provides it to team A. In doing so, team B does retain the information of said card. If they do not, the server determines if the draw pile is empty. If it is not, the server pops the first card from the draw pile and adds this card to team A's collection. If the draw pile is empty, it instead gives the team a random clue. This way, the application on the tablet only needs to keep track of the information their own team has, as well as the ids of all other teams. Other information is not required, since the server keeps track of all that.

As a result, if one of the tablets were to lose their connection or even crash, no information would be lost upon reconnecting or restarting the game. Once the tabled has reconnected with the server, it will send all required data to the tablet. Re-sending this information to a team behaves the same way as a team sending information to the server. The server sends a string to the tablet, of which the first word is the keyword that determines what action needs to be taken. The remainder of the string contains all information required for handling the function associated with that keyword.

However, not all actions need to be decided by the server. For example, the start of a cool down needs to be sent by the server, but the tablet keeps track of when the cool down period is over. Once it is, the two teams can once again ask each other for cards again.
Similarly, the code on the client side decides whether a card can be asked by one team from another team, by going through the list of completed sets, owned cards and clues. However, the application is incapable of checking whether the other team has this card on its own.

4.2.1. Code structure

The app is comprised of three folders, being the data, resources and display folder. During the project, each class that only had to contain data, such as Card.kt or Team.kt, was placed in the data folder. For the layout of screens, menus, text, images and so on, every file (mostly .xml files) was placed in the resources folder. Lastly, each class that had to interact with both the data classes and the resource files, was placed in the display folder. These classes contained the function for clicking on a button, showcasing all owned cards, trading with other teams and so on.

4.3. The web interface

As mentioned in section 3.1.2, the game master is responsible for managing the game, before, during and after it is played. For this, they need a web interface to easily manage all ongoing games. This web interface, written in JavaScript and utilising HTML and CSS, includes functionality for:

1. Starting, cancelling and deleting a game;
2. Having an overview of the data of each team. That is, their owned cards and information, as well as their location and amount of points;
3. Having an overview of the data of each game. That is, the time remaining,
4. Chatting with all teams;
5. Showcasing the locations corresponding to all sets in a game;
6. Comparing and consequently approving or disapproving photos sent by a team;
7. Adding extra sets to an ongoing game.
8. (Separate screen) Creating a new game
9. (Separate screen) Editing card sets

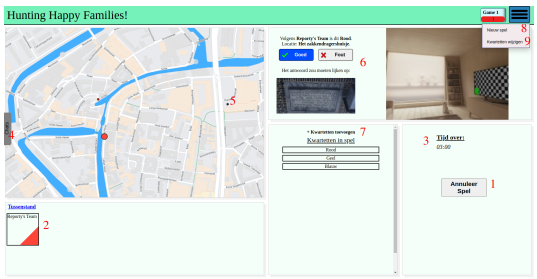

Figure 4.1: Basics of the web interface for an ongoing game

The web interface gives the game master control over all ongoing games. The basics of the main web interface can be seen in figure 4.1, with additional annotated numbers corresponding to the numbers in the enumeration above. this figure does not show all of the web functionalities however, as part of the functionality is created using pop-ups or overlays. To elaborate on the specific options, this paragraph will quickly go over this functionality.
First, a game can be created using the 'nieuw spel' option from the drop down menu in the top right. When creating a game, the game master can determine the game duration, amount of teams, which sets to include, points given for certain actions, and the city to play in. A created game is displayed as a rectangle to the left of the drop down menu button. Figure 4.1 showcases one created game, though more games would be added as similar boxes. Selecting a game box will bring up the information for that game, occupying the rest of the screen.

In the game screen, all the information about that specific game can be seen. This includes the map on the left side, showing the team locations as a large colored dot, and locations for the card sets as smaller dots. Underneath the map, the team information is displayed. Here, every team in this game is represented as a

box. Clicking a team box will show all information about that team, including their current points, verified locations and owned cards and clues. The 'tussenstand' button shows an ordered ranking list of all teams and their points.
On the right side of the screen, a box is displayed for showcasing images as sent by the teams, as well as an extra image which shows the result images associated with that location. Additionally, there are two buttons, 'goed' en 'fout', giving the game master the option to approve ('goed' button) or disapprove ('fout' button) the image. This, in turn, lets the team know whether they were wrong or right, including awarding the team points if they were correct.
In the bottom right two more boxes can be seen; the left box shows the card sets that are currently in the game, and gives an option to add more card sets during a running game. Clicking a card set will highlight the location of that card set on the map. This way the game master can quickly find any card set location.
The other box in the bottom right contains buttons to change the game state between created, started, finished and deleted. Only the applicable button is shown, to ensure the game can only change into each state once.
Lastly, the chat system of each game can be inspected and used by the game master using the collapsing display on the left side of the screen. Clicking the chat button will show a WhatsApp-like interface where the game master can read or take part in chats with the playing teams.

Aside from the screen shown in 4.1, a second web page can be accessed by clicking the 'kwartetten wijzigen' button in the drop down menu on the top right. This will bring you to a page where the game master can add sets to, or edit or delete any card sets within the system, to be used in upcoming games. Any changes made in this screen are directly changed in the server file system.

5 Testing

To ensure the highest quality for the final product, the system must be thoroughly tested on low and high level. This chapter provides an overview of the used testing methods for this product. This includes continuous integration and its static analysis, as well as unit, scenario and field testing. Lastly, the error handling and logging of the product will be discussed.

5.1. Continuous integration

Newly created code must be tested within the workflow of a project. Luckily, Gitlab provides continuous integration (CI) for all of its repositories. This software structure executes tasks in pipelines on Docker images. When a pipeline in one of these images fails, the person to start the build is notified of the failure, and must adjust their code to be accepted. The CI is configured using the file named .gitlab-ci.yml. In the case of this project, it is created such that static analysis (that is, checking the format of the code) and test execution are performed on every push done to the Gitlab repository. CI is mostly utilised as a safeguard, meaning that, when a build fails on its CI, it is impossible to merge the code to main branch.

5.1.1. Static analysis

The static analysis of this project has been exclusively implemented for Kotlin, as it is the integral element of this project. The static analysis is performed using the framework ktlint [1], and Gradle as its project builder. This is done by using the framework ktlint, and Gradle as its project builder. The plugin utilised to combine these two elements was kotlinter-gradle [2], a gradle plugin that intends to be simple to set up, and to 'behave as you'd expect out of the box' [2]. Whenever code is pushed to the Gitlab repository, the kotlinter-gradle plugin inspects the code on the used format. This can also be checked locally using the Android Studio program utilised by the team to write the Kotlin codebase. If the format does not obey the format rules defined by the team, the analysis will fail and the developer must adjust the code to conform to the expected standard.
Other potential plugins for static analysis included ktlint-gradle [3] and spotless [4]. While ktlint-gradle in particular exists slightly longer than kotlinter-gradle, having its first release in March of 2017 compared to May of 2017, kotlinter-gradle had the personal preference of the team, as it appeared easier to implement and use.

5.2. Unit testing

In verifying the code to work as expected, unit tests are a simple but crucial tool. These tests ensure that written code will perform the intended functionality when invoked. Since Kotlin is an extension of Java, any and all frameworks usable for Java are also available for use in Kotlin. As a result, the codebases utilises JUnit5 [5] as its main framework to perform these tests consistently. Similarly, MockK [6] is utilised as a framework for

[1] https://ktlint.github.io/
[2] https://github.com/jeremymailen/kotlinter-gradle
[3] https://github.com/JLLeitschuh/ktlint-gradle
[4] https://github.com/diffplug/spotless
[5] https://junit.org/junit5/
[6] https://mockk.io/

mocking objects irrelevant to the unit tests. When compared to Mockito [7], the main Java mocking framework, MockK is better equipped to handle the Kotlin language, as it was written specifically for Kotlin, unlike Mockito.

Figure 5.1 shows the code coverage as a result of the unit tests written. As further described in section 7.2, testing has been unconsciously ignored for the sake of progress. Therefore, the code coverage the team intended to have has not yet been met. The fact that various functionality, especially for the front end handling, is dependent on elements that are difficult to mock, does not help this matter.

Element	Class, %	Method, %	Line, %
data	12% (7/56)	16% (26/155)	5% (39/713)
display	0% (0/102)	0% (0/255)	0% (0/1502)
game	0% (0/1)	0% (0/1)	0% (0/1)

Figure 5.1: JUnit coverage

Element	Class, %	Method, %	Line, %
cards	0% (0/4)	0% (0/17)	0% (0/35)
chat	100% (3/3)	87% (7/8)	72% (13/18)
game	8% (3/35)	24% (15/62)	5% (22/435)
map	0% (0/4)	0% (0/14)	0% (0/40)
other	0% (0/5)	0% (0/19)	0% (0/117)
teams	33% (1/3)	13% (4/30)	7% (4/57)
ApplicationData	0% (0/2)	0% (0/5)	0% (0/11)

Figure 5.2: JUnit coverage of the map "data"

This does not mean the game has not been meticulously tested. This is detailed further in section 5.5.

5.3. Scenario testing

On a larger scale, scenario testing is performed to ensure behaviour across modules is as required. When actions in the UI are executed in a specific order, a specific result needs to be derived from that. In fact, these tests perform the exact same actions that a user would do when playing the game. For this reason, not many of these tests were written, as it took less time and was more fun to actually test the scenarios by playing the game with the project team. This does not mean that no effort has been made in testing, however. The scenario tests are performed on certain scenarios that would take more time to reach by playing the game, as the tests can simply skip certain stages.
To this end, the Espresso framework[8] has been selected to perform these tests in combination with the MockK framework. However, the Espresso framework does not work yet in combination with JUnit5. Therefore, JUnit4 is used in the AndroidTest folder instead.
The test coverage is calculated using the JaCoCo framework. This coverage includes the JUnit tests as mentioned in section 5.2. However, as can be seen by comparing figure 5.3 with figure 5.1, the AndroidTests have not tested the data folder. Thus, it can be concluded that these tests only covered the display folder.

debugAndroidTest

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
cityhunter.display		33%		20%	632	833	903	1,432	238	398	59	102
cityhunter.data		11%		6%	457	557	658	741	186	274	42	57
Total	13,385 of 17,953	25%	1,175 of 1,388	15%	1,089	1,390	1,561	2,173	424	672	101	159

Figure 5.3: Espresso coverage

[7] https://site.mockito.org/
[8] https://developer.android.com/training/testing/espresso

debugAndroidTest

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
cityhunter.data.game		3%		0%	280	308	410	424	97	125	30	35
cityhunter.display.trading		0%		0%	105	105	214	214	44	44	13	13
cityhunter.display.dialogs		25%		16%	134	154	149	217	35	50	12	17
cityhunter.display.map		0%		0%	80	80	163	163	40	40	12	12
cityhunter.display.chat		46%		31%	77	116	122	210	28	64	1	15
cityhunter.data.other		0%		0%	51	51	112	112	21	21	5	5
cityhunter.display.cards		72%		47%	92	188	60	320	19	90	1	15
cityhunter.display.welcome		0%		0%	50	50	89	89	27	27	9	9
cityhunter.display.faq		0%		0%	41	41	51	51	24	24	6	6
cityhunter.display.camera		0%		0%	41	41	57	57	16	16	5	5
cityhunter.data.teams		48%		36%	40	73	39	66	19	43	1	3
cityhunter.data.map		0%		0%	18	18	38	38	15	15	4	4
cityhunter.data.chat		24%		17%	28	36	13	19	16	22	1	3
default		23%		5%	15	23	33	41	6	14	0	1
cityhunter.data.cards		60%		16%	18	36	13	36	7	24	0	4
cityhunter.data		18%		25%	7	12	6	10	5	10	1	2
cityhunter.display		79%		0%	7	19	0	38	4	16	0	2
cityhunter.display.rankings		97%		79%	5	39	0	73	1	27	0	8
Total	13,385 of 17,953	25%	1,175 of 1,388	15%	1,089	1,390	1,569	2,178	424	672	101	159

Figure 5.4: Espresso coverage of the map "display"

5.4. Field testing

After the application has been tested on code quality, the application also needs to be tested on performance and user-friendliness. As writing tests cannot assess these qualities, physical tests, or field tests, must be performed instead. Initial field tests have been performed with exclusively the developers, sometimes accompanied by the client. The reason for this is to verify that specific error messages caused by bugs would not appear while using the application, since these error messages can be confusing to regular users. These field tests had the additional purpose of white box testing, to find and consequently remove any bugs not found in earlier testing. These field tests were performed in both Schiedam and Delft.

Afterwards, the client and the developers reached out to friends and family for a first game balancing-oriented playtest. These tests were performed on Friday the 19th and Saturday the 20th of June. There, players were provided an explanation of the game. On Friday, an explanation of the app was also provided. This was purposely omitted during the game explanation on Saturday to test the user-friendliness and intuitiveness of the app.

Once the game has ended, players had the opportunity to fill out an online questionnaire to better help understand how the players experienced the game. A total of fifteen players have filled out this survey. One submission was omitted, as it was duplicated. The remainder of this section discusses the results of the survey.

5.4.1. Diversity of participants

The testing groups for Friday and Saturday both filled out the same questionnaire. On Friday, a group of scouting members of the client played the game. Five of those players have responded to the survey. Saturday, friends and family of the developers tested the game. The following charts (5.5, 5.6 and 5.7) represent the accumulated results for player diversity in these two playtests.

As can be seen in figure 5.5, the age diversity was high. Though the representation of people of middle age

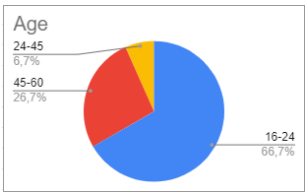

Figure 5.5: Age of participants

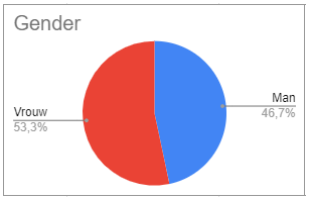

Figure 5.6: Gender of participants

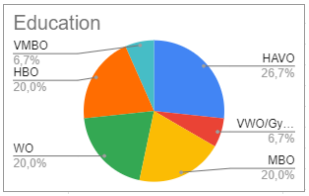

Figure 5.7: Highest education completed by participants

was low, this is compensated by the fact that various players in both of the adjacent age groups trend towards

this middle age rather than the opposite. As a result, this group is likely to represent the customers this game will attract.
A similar trend can be found in figures 5.6 and 5.7. The difference in representation of men and women is marginal, though there were more women than men who have filled out the survey. In terms of education, representation of all levels of education is present in the answers. While this does not make the survey representative, as is to be expected with the little results garnered, information derived from this survey is still valuable.

5.4.2. Test results

From the survey, a number of feedback points were discovered. This subsection will further detail the three major findings.
First, some cards appeared to be more useful to players than others. This is because they are considered the easiest clues to finding the associated location. Figure 5.8 showcases that the "Plattegrond" and "Detail" card are considered to be the most useful for discovering the location of a set. On the other hand, the "Verhaal" and "Foto" were never considered as the most useful card. Unsurprisingly, all people who would often be somewhere near the playing field all preferred the detail card, as these people would likely recognise these locations from the detail card. As each card is intended to be equally useful to individuals, the client may need to look towards re-balancing the usefulness of each card type.

Next, the gameplay explanation provided within the game appeared to have been often overlooked within the game. When asked about whether the FAQ was missing elements they believe should be included, five of the eight given answers stated they were unaware that this screen exists. While this is not relevant for the sake of balance, it was intriguing to see that this was the case.

Lastly, though the focus of the tests was mostly on balancing the game and improving intuitiveness of the app, the gameplay of Saturday unfortunately had two minor but intrusive bugs, which was also reflected in the survey answers of that day. Specifically, various teams lost their clue cards when reconnecting to the game, and others had their games crash upon receiving certain messages from the server. For this reason, the game balance could not be properly assessed during the playtest on Saturday.

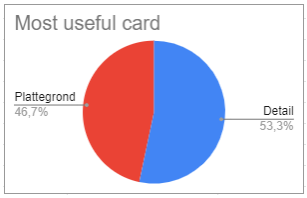

Figure 5.8: Most useful card according to participants

A full list of answers to the questionnaire can be found at `https://bit.ly/HHFEnquete`.

5.5. Error handling and logging

When a tablet is used to play the game, any problem should be handled in such a way that it causes minimal inconvenience to the user, while still providing the developers all the information required to resolve the problem permanently. Towards this goal, the app is created to have strictly defined behaviour in the event of an exception. When an unhandled exception occurs anywhere in the application, it would stop functioning, and return to the login screen (figure 3.1). Upon reconnecting, the server would recognise the tablet based on a unique device ID, assign the tablet to the right team and send all necessary information to the tablet so that the user can continue playing as if nothing happened. During this process, the app would send a full error report to the server. This includes elements such as the device logs and stack trace, as well as the time and build number.
The error log was used to find various major bugs present within the system, including the bugs defined in section 5.4.2. Moreover, the error log was also utilised for testing purposes during implementation. To try to ensure no bugs would be added while implementing new aspects of the game, the developers utilised these error logs to find the source of emerged bugs, and resolve them accordingly. This was especially notable when implementing the trading functionality, where the error lag was utilised to ensure a trade would be properly

conveyed from the app to the server, and back.

6

Product evaluation

This chapter provides various methods of evaluations. That is, the MosCoW requirements set at the start of this project (A.8) are evaluated, after which an overview of the Static code analysis of SIG is delivered. Lastly, the playtesting expected for this product is reviewed.

6.1. Requirements

When comparing the initially laid out MoSCoW requirements as seen in appendix A.8 to the final product, the system has functionality for all but one must-have requirement, all should-have requirements and no could-have requirements. A full review of each individual requirement can be found in appendix B. This section will go into detail about all missing functionality, as well as requested functionality after the requirements had been defined.

6.1.1. Missing requirements

The missing must-have requirement is *The application shall provide a team with a starting location, as chosen by the game master*. This requirement stems from an early design of the system, in which the game would provide an individual location for each team.
When implementing the system, the client made clear that he would prefer one central location for starting the game. All teams would meet there prior to starting the game, and would then start from there. For this reason, this must-have requirement was considered deprecated, and was therefore removed from the final product.

6.1.2. Additional requested functionality

During implementation, the client inquired about the possibility of playing the game in other cities. As the team mostly lives in Delft, this functionality was considered ideal for testing purposes, and this would also allow the game to be playable in other cities. Thus one team member created sets for the game using locations on or near the TU Delft campus, and the product was refactored to enable gameplay in Delft.

6.2. SIG: Static code analysis

As part of the final process, two submissions have been handed in at the Software Improvement Group, or SIG, for static analysis of the project. The first submission was performed after four weeks of the implementation phase, and the second submission was completed after six weeks.
SIG evaluates the codebase on eight aspects, the combination of which can be generalised to 'maintainability'. These aspects are volume, duplication, unit size, unit complexity, unit interfacing, module coupling, component balance and component independence. For this project, the final aspect "component independence"; is ignored, as SIG has not provided a response on this aspect.
An overview of the first submission can be found in figure 6.1.

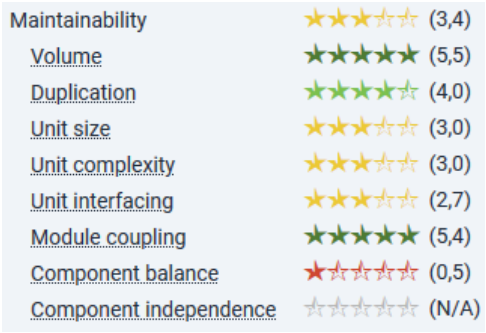

Figure 6.1: SIG submission 1

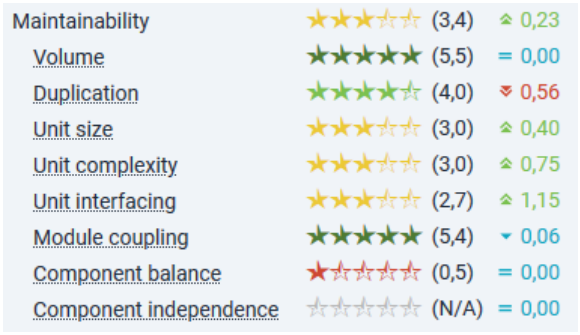

Figure 6.2: The difference of Kotlin

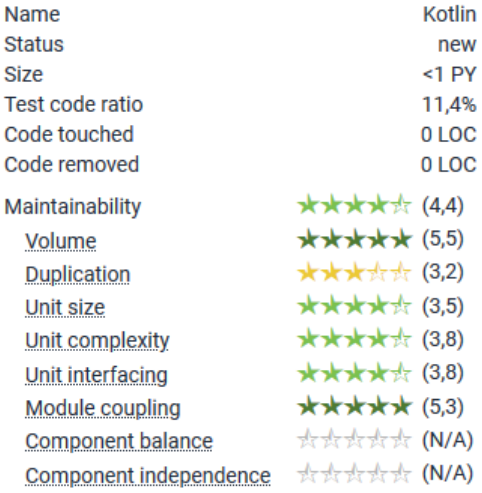

Figure 6.3: The initial results of Kotlin

6.2.1. Kotlin

As mentioned before in section 4.2, the application, written in Kotlin, is the largest component of this project. However, the initial response of SIG did not include evaluation of the Kotlin code. After communicating this to the correspondent, the team retroactively received evaluation on the application. This puts the initial submission in a unique situation where the influence of the Kotlin code on the total codebase is directly visible for the product. The result of which can be seen in figure 6.2, as well as the individual scores of Kotlin in figure 6.3.

As predicted, the influence on the codebase of Kotlin is of a positive nature. This is especially notable in unit interfacing, since the functionality in Kotlin tends to only need one parameter on average, though there are some extreme exceptions. The two decreased elements will be explained further in their respective sections.

Ignored aspects

The clear negative standout in these scores is the component balance. The reason for this is the fact the evaluation considers the entire project to be one component, therefore resulting in a "severely imbalanced" codebase, according to the analysis. For this reason, this aspect is ignored from this point forward.
Similarly, the clear positive standout, the code volume, is also to be ignored from this point forward. The Software Improvement Group is created with codebases of millions of lines of code in mind, and considering this codebase has up to the tens of thousands of lines of codes, the volume of the codebase will always be considered small.

Code duplication

Duplicated code, which tend to clutter a codebase and is a source of increased maintenance, is generally of high quality, only having a few extreme exceptions. These are the CardMenuAdapter, combined with the TradingMenuAdapter, and CardScreenActivity combined with TradingScreenActivity respectively. The reason

for this code to be duplicated is because the functionality for these elements are different in subtle but significant manners. Both are utilised for showcasing a certain screen, or aspect of a specific screen. While these do execute very similarly, the Trading screen requires extra information for functionality that is not part of the Card screen. Furthermore, the activities and their respective adapters are linked to the front end of their respective screens and must therefore remain separated for connection with the backend of the codebase. Thus, these two elements are to remain unchanged.
Code duplication is showcased as the only other aspect to have decreased as a result of the addition of Kotlin code, though it is still of a high quality. Since the main source of code duplication across the entire product stems from the Trading and card screens' adapters, as mentioned prior, this is to be expected.

Unit size
The average unit size should remain relatively small for the simplicity of the codebase. However, the connection handler of the server and web interface comprise the three largest functions, with the HandleClientMessage being almost 250 lines of code. Each of these are comprised of large switch case statements, with the functionality of said switch cases being written in-line rather than as a separate function.
To remedy this issue, the server is to be refactored to use lambda functionality instead of these switch statements. That is, rather than having a large switch statement, the keywords are stored as a list, and, when invoked, the individual functionality associated with that keyword is called. This way, the unit size outliers should be reduced significantly, thus resulting in a codebase that is easier to maintain.

Unit Complexity
The McCabe complexity defines the amount of paths a function can take to finish. As expected, the aforementioned three are again the main functions which have to be changed. The reason is the same as the one described above, and all three of these have been adjusted for the second submission.

Unit interfacing
Ignoring component balance, unit interfacing is the worst aspect of this product. Unlike the previous two aspects, there is no main source for this. While there are three functions with six parameters, all three of these have relatively small influence on the overall score. Rather, there are many functions that use three or four parameters. Most of these issues stem from data classes, which, for example, save the information of a team, a game, a card, and so on. Since defining each of these require multiple parameters, and converting this into methods that require less parameters is more likely to increase confusion rather than reduce it, it is likely that this score is not going to be worked on a lot. The focus will instead be directed to the other two elements, as these can be significantly improved without increasing confusion.

Module Coupling
Module coupling implies that classes must be properly separated, and minimally connect with other classes. The scoring on this is near perfect, and therefore no changes will be performed to improve this. Initially, module coupling had a perfect score. Now, one class requires some attention, this being the BaseActivity class. The reason this class has a relatively large fan-in, despite being only 10 lines of code (LOC), is because it is the general interface used by other activities that all require a specific function. This was done to prevent code duplication, and therefore shall remain unchanged for the sake if the analysis.

6.2.2. HTML and Shell
Although insignificant for the final product, the effect the HTML codebase and the Shell (Defined by SIG as 'batch') codebase have on the overall grade is significant. The Shell element of this codebase, which is comprised of 64 LOC, has a maintainability score of only 2.5. Similarly, the HTML codebase of around 350 LOC has a score of 3.1. The main reason for both of these stems from unit size, as well as unit complexity for the Shell script, and code duplication for the HTML aspect.

Shell
The shell script in this project is a file used for testing at the very early stages of this project. It is irrelevant to the codebase of this project, even though it has been mistakenly added to the submission of SIG. The shell script, however, scored terribly for the static analysis. These results are seen in figure 6.4. Since there is no inherent function call in the shell script, the entire script is seen as one function. As a result, the complexity of said file is also significantly higher than the actual complexity, which is believed to be no more than six. As

Name	Batch
Size	<1 PY (0,00)
Test code ratio	0,0% (0,0)
Code touched	0 LOC
Code removed	0 LOC
Maintainability	(2,5) = 0,00
Volume	(5,5) = 0,00
Duplication	(5,5) = 0,00
Unit size	(0,5) = 0,00
Unit complexity	(0,5) = 0,00
Unit interfacing	(N/A) = 0,00
Module coupling	(N/A) = 0,00
Component balance	(N/A) = 0,00
Component independence	(N/A) = 0,00

Figure 6.4: The Shell script evaluation

Name	HTML
Size	<1 PY (+0,01)
Test code ratio	0,0% (0,0)
Code touched	134 LOC
Code removed	32 LOC
Maintainability	(3,1) ▲ 0,07
Volume	(5,5) = 0,00
Duplication	(2,4) ▲ 0,35
Unit size	(0,5) = 0,00
Unit complexity	(5,5) = 0,00
Unit interfacing	(N/A) = 0,00
Module coupling	(N/A) = 0,00
Component balance	(N/A) = 0,00
Component independence	(N/A) = 0,00

Figure 6.5: The HTML evaluation

a result, this will be disregarded for the final submission.

HMTL
Similarly to the shell script, the HTML codebase also has a very large unit size. Since a HTML class needs to envelop an entire web page, which is also not defined by specific functions, each HTML file is considered to be one function. As some web pages are well above the 'dangerous' level of 60 lines of code, this results in a very poorly scoring code size. In a similar manner, some functionality, which is required twice for slightly different aspects of a web page, is likely to be duplicated. HTML does not easily allow for code to be repeated for different aspects of a web page, nor can a function be invoked to create a certain element twice. For this reason, HTML will not be discussed for the second submission.
A full overview of the HTML results can be found in figured 6.5

6.2.3. Second submission

Three weeks after the initial hand-in, a second hand-in was provided for static code analysis. The result of the new hand-in can be seen for Kotlin in figure 6.6, and for the JavaScript codebase in 6.7. With this new sub-

Name	Kotlin
Size	<1 PY (+0,09)
Test code ratio	32,4% (+21,0)
Code touched	910 LOC
Code removed	141 LOC
Maintainability	(3,9) ▼ 0,42
Volume	(5,5) = 0,00
Duplication	(3,4) ▲ 0,21
Unit size	(2,4) ▼ 1,16
Unit complexity	(3,6) ▼ 0,18
Unit interfacing	(3,2) ▼ 0,53
Module coupling	(4,7) ▼ 0,65
Component balance	(N/A) = 0,00
Component independence	(N/A) = 0,00

Figure 6.6: The second SIG submission: Kotlin

Name	JavaScript
Size	<1 PY (+0,12)
Test code ratio	0,0% (0,0)
Code touched	1.446 LOC
Code removed	407 LOC
Maintainability	(4,0) ▲ 0,23
Volume	(5,5) = 0,00
Duplication	(5,5) ▲ 0,35
Unit size	(1,7) ▼ 0,04
Unit complexity	(1,8) ▲ 0,23
Unit interfacing	(2,9) ▲ 1,50
Module coupling	(5,5) = 0,00
Component balance	(N/A) = 0,00
Component independence	(N/A) = 0,00

Figure 6.7: The second SIG submission: JavaScript

mission, the Kotlin maintainability appears to have decreased, with unit size, unit coupling and unit coupling worsening significantly. In contrast, the JavaScript codebase has improved incredibly in unit interfacing, but has no notable difference otherwise. This subsection considers the changes in the Kotlin and JavaScript code-

base.

Kotlin
As the team focused on the code quality of the JavaScript, the Kotlin codebase has been left at the wayside slightly in terms of tight focus on code quality. As a result, the unit size has decreased from average to weak. However, most of these issues stem from various activity oriented classes. That is, most activity classes have been given a few extra lines of code to better envelop their intended functionality. This results in a small increase in unit size across various functions, including some empty lines within certain functions (which are considered as part of the function method size). However, the team believes this to be somewhat misleading as separating these average to large-sized classes would likely result in more confusion, which in turn also lowers maintainability. Regardless, the score of the maintainability is still of a high enough score to be considered good quality.

JavaScript
Despite a significant refactor to the server tackling the weak unit size and unit complexity, the JavaScript codebase still appears to be very weak in these two aspects. The reason for the relatively marginal difference, as opposed to the expected significant difference in unit size, lies in the proposed solution. That is, the lambda functions created as a means to this refactor (see also: section 4.1.1) are considered as one function of unit size and complexity rivaling the original solution, rather than various small functions.
Each individual lambda function is created at the end of the *server_msg_client.js* and *server_msg_gm.js* files respectively. There, these functions are listed as invoked methods for other classes to utilise. However, since they are not explicitly called a 'function', the SIG evaluation tool considers all to be one singular function instead, which is of sizes and complexities well above expected quality.
As a result, these two metrics have been scored lower, likely to a significant margin.

6.3. Bugs and playtests

Although all of the expected implementation is part of the final product, it is not considered complete. The main reason is that there are still a few bug fixes to be completed. The full playtests, as described in section 5.4, while successful, still had a few of these minor but intrusive bugs. For example, the game had a chance to crash due to fields which were equal to null, where this was assumed to not be the case. Players had to restart their game multiple times during this test, which impeded the process of properly balancing the game. While this issue has since been fixed, and teams no longer have to restart their game to reconnect to the server, this does was impossible to be performed while the game was being tested. As a result, the team has not been able to properly balance the game. While the game is ready for release and the game is created to be customisable, the product can still be improved and tested further after this BEP has been completed.

7

Process evaluation

This chapter evaluates the process of this project. Specifically, the communication with the client, setup and tool choice are commended, after which the critical attitude, and testing will be criticised. Lastly, a few notable problems in communication and language familiarity are discussed, as well as the means to resolving these issues.

7.1. Commendations

Throughout this project, various aspects are considered to have been handled well. First, the team ensured proper cooperation with each other. That is, the team held regularly scheduled online meeting to discuss and prioritise current and future work, and members of the team helped solve problems others might face effectively.

This regular schedule also reflected itself in communication with the client. By holding weekly online meetings with the client, the product was discussed efficiently, and any lingering questions were answered quickly, thus resulting in clarity and structure for both parties.

Lastly, the chosen tools have been deemed proper picks for the final product. Utilising the Kotlin language, the Gitlab repository, and the CI described in section 5.1, initial setup of the project was completed early. Considering this good foundation, starting the first implementation phase became significantly easier.

7.2. Weak points

On the flipside, the critical attitude towards each other's work was lackluster during the project. Oftentimes, the team would merely read through the code with a critical eye, without testing code. This was especially noticeable when combined with the other bad practice of blob merges, which some members of the team tended to create.

This also reflects in the fact that the team could have worked in a more structured manner. For example, Gitlab offers an issue board where programmers can have a clear overview of what should be done. Rather than constantly requiring verbal or chat communication, using the provided Scrum board more effectively would have alleviated some of the issues created as a result of a lack in structure.

A lack in structure can also be seen in the planning of writing the final report. Although the initial planning intended for the report to be written alongside the product's creation, it was instead mostly written as the project was ending. Documentation of elements that would have been useful to keep track of for the final report, such as problems that arose and specific design changes, instead had to be remembered afterwards. for future projects, it is crucial to find a better balance between programming and documenting, to ensure all information can be easily passed on to others.

Lastly, while programming for this project, the focus was too severely put on implementing new features. Rather than writing unit tests and scenario tests, the team often preferred to playtest functionality. Various

tests were then written retroactively, though these could instead be used for white box testing all classes during creation, as this is an integral part of finding bugs that were now often overlooked. In creating these tests, a more stable product would be established at each phase of the project, which would lead to an overall faster process.

7.3. Notable problems and their solutions

7.3.1. Communicating during a pandemic

Because of the COVID-19 pandemic, the project was forced to be done without physical contact. It was near impossible to arrange meetings in real life, taking the regulations regarding the virus in mind. Furthermore, chat conversations could often lead to miscommunication between team members.
This problem was solved by communicating digitally. Every weekday, with some irregular exceptions, the team met via voice calls on Discord to discuss progress made since the last meeting, as well as the schedule for the upcoming day. Similarly, meetings with both the client and the supervisor were completed using the video call platform of Jitsi and Zoom, respectively. Though communication through the internet still leaves some things to be desired, communicating effectively via the internet decreased the severity of the global issues on the project. Thanks to the positive progress made nationally, it was even possible to meet in real life to perform playtests near the end of the project.

7.3.2. Familiarity with Kotlin

Although this problem was not crucial, it is still a good point to mention. At the start of the project, no member of the group programmed in Kotlin before. It took time in the beginning stage of the project to get used to working with this programming language.
This problem was solved by doing courses online to learn Kotlin and to search for solutions to problems encountered online. At the end, Kotlin turned out to be a good choice for a programming language of Hunting Happy Families.

8

Future recommendations

Considering the final result of this project, there are still areas of improvement. This chapter discusses these areas. First, the improvements for the game are defined, after which the improvements for the application are discussed. Lastly, recommendations for the game master web interface are considered.

8.1. Game improvements

The game, in its current state, has not yet been fully tested on balance. As defined in 5.4, the main field test performed could not yet focus on balance, since the game was not entirely ready for this phase. This could be further explored in a later stage, once the game quality is at a level where the team is confident to move on to solely field testing the game. Having said that, with the current implementation of this game, the client is capable of testing this themselves, since the application is fully customisable in terms of handed out cards, point balance, game time, team count and set count, if he so chooses.

8.2. App improvements

Currently, one of the main issues of the game stems from connection loss with the server. Losing connection for a short while may have large consequences. Improvement can be done to increase resistance to connection loss, which in turn leads to a better and more consistent gameplay experience. This would be noticeable when sending chat messages or trading cards with another team, for instance.
Moreover, certain screens of the application take a few seconds to load, since the page has to be rendered entirely. As specific elements are consistent across all pages, this could be reduced to only loading the elements of the page considered unique to that page. This way, loading times may be reduced, resulting in smoother navigation of the menu.

8.3. Web interface improvements

There are still many potential functions that could be added to the web interface for more possibilities regarding game management. These aspects include, but are not limited to:

- Removing card families from a game;
- Removing players from a game;
- Setting the score of a team manually;
- Providing clues or cards to teams.

Similarly, there are some minute details for adjusting the card family database that can be improved in the future. These details are:

- Allowing custom card titles rather than the four standard ones (story, map, photo and detail);
- Placing images next to each other rather than underneath each other to avoid excessive scrolling on the page.

9

Conclusion

This project started as the request to build an application for the game Hunting Happy Families, a game in which people can learn about the history of Schiedam through collecting sets of information and subsequently finding the location. In it, the vision of both the client and the team are created through a Kotlin application for players, a web interface for the game master, and an interconnecting server. As the product had to be created from scratch, including the core principles of the game, the final product is not yet properly balanced for playing the game, and some small bugs might still be undiscovered.

Using the app and web interface sockets of the server, teams are able to communicate with each other and the game master, as well as the ability to utilise the core gameplay mechanics of trading and verifying to gain as many points as possible. The app in particular is created with intuitiveness in mind. After a small instruction on the game mechanics, which the team still intends to provide once the product is properly balanced, people of all ages should be able to navigate the app with ease, and enjoy the scenery the game leads you to see.

As the game is enjoyable for both the team and the playtesters, pursuing the optimal experience is a worthwhile cause after this project has ended. Despite the game not being entirely ready for release to the public, the potential that can be reached with minimal changes appears large and feasible.

With the positive view on the game in mind, and the limitless potential at the team's fingertips, Hunting Happy Families is bound to carve its niche in the market of group events.

A

Research Report

A.1. Introduction

Hunting Happy Families, from here on referred to as HHF, is a card-based variation of the game Happy Families, in which players compete against each other through collecting sets of four cards. In HHF, teams gather and trade cards with the other teams in order to find landmarks in Schiedam with the clues on the cards. By collecting more cards of the set, it becomes easier to hunt down the location. Teams are competing against each other by earning points through completing sets and finding as many locations as possible.
Instead of playing this game with real cards, Cityhunter wants to create a mobile application that is playable on Android. This chapter first delves into the key features of HHF, then delivers the problem statement provided by the context and the background, after which it defines how digitization can be used to solve the problems created and what requirements are decided upon in deliberation with the client. Lastly, the chapter will list the intended approach of the team, followed by potential challenges during this project.

A.1.1. Key features

Hunting Happy Families is a team game that features collecting cards that contain pieces of information that lead to an historic location in Schiedam. The game currently includes 20 unique locations. As players find the information on more cards, they can explore more of these unique locations and also collect more points, therefore creating a fun teambuilding activity. The following components form the key features to do so:

- **Education**
  Schiedam is a beautiful city with great historic value. A team can win the game by finding the correct locations in Schiedam through solving clues and connecting pieces of information, thereby learning about the history of the city.
- **Teamwork and competition**
  Each team member has to work together to find locations as quickly as possible. They are competing against their friends or colleagues in other teams, which adds an extra dynamic to the game.
- **Discovery**
  Players have to walk outside from place to place and from team to team to win the game, which helps create a more physical experience.

A.1.2. Problem statement

The challenge provided by Cityhunter consists of two parts:

- **Developing a mobile application of HHF.**
  The goal of the game is to have a great time together with friends while learning about the city of Schiedam. The game must be fun to play for people of all ages, and therefore be intuitive to use. The key features of the game as it is now should not change.
- **Creating a web interface.**
  Since the client has no experience with programming, they also request a simple web interface in which

they can easily add, remove and adjust cards, as well as monitor an ongoing game. An explanation of what exactly falls under monitoring a game, can be found in Appendix A.7.

A.1.3. Digitization and requirements

Playing HHF with real cards presented various problems, to which digitization allows a solution. These problems and the solutions that can be implemented due to digitization are as follows:

- When asking for a wrong card, teams could not directly receive a new card from a drawing pile. Instead, they would receive a card from the other team of the other team's choosing. In practice, this meant that teams would exchange cards rather than obtaining and collecting them.
  The application solves this by adding a drawing pile.
- Teams had to take a picture with the correct set of cards in front of the location to prove that they had found the right location, and associated it with the right color. After the game ended, the game master would verify these pictures.
  To resolve this, and consequently allow for an in-game ranking, the digitized game verifies a location by using the system's GPS location during gameplay.
- Without any known GPS-locations, teams would rarely encounter each other. Therefore, they had less opportunities to exchange cards and information.
  In the digitized game, a map on the tablet will be provided, with the GPS locations of each team.
- Communication with the game master was only possible through exchanging phone numbers.
  The app will provide a chat service, which allows for quick communication without requiring personal information to be exchanged.
- Information and clues on a card would be lost if said card was exchanged with another team. Teams could easily forget important clues that could lead to the right location.
  The app enables teams to save card information and to treat the ownership of information and cards as two separate entities, therefore providing a concise overview and relieving teams of the need to remember all clues.
- The game master did not have the means to monitor the game. For example, he could not see when a team had stagnated and needed help.
  The digitized game provides the game master with a web interface, which can monitor the locations, points and cards of each item, as well as allowing the game master to chat with teams that may have stagnated.

It can be said that the transition from playing with cards to playing on tablets opens up the way for new features, but also new constraints, to be added. Or in other words, the digitized version of HHF has different requirements than the original game. The list with the requirements for the application game HHF can be found in Appendix A.8.

A.1.4. Approach

To bring the project to a successful end, it requires multiple phases:

- **Orientation phase**: during this phase, the ways of documentation, testing frameworks, programming languages and software development process are chosen. On top of that, the requirements of the product are defined conform the MoSCoW method, meaning that they are divided into "must have", "should have", "could have" and "won't have" requirements. At the end of this stage, a research report will be delivered.
- **Implementation phase**: This project phase encompasses developing the game using the foundation that has been laid in the research report. This includes testing the code and implemented features.
- **Field testing phase**: After four weeks of developing the game, a basic working product should be created. During the field testing phase, the team investigates which variables need to be adjusted in order to create a playable and fun game. This includes elements such as the ranking system, the ratio of teams to the amount of sets, the distance between locations and other small factors that influence the gameplay.

- **Implementation phase II**: This fourth phase is intended for adjusting game settings in accordance with the findings of the field testing phase. Another part of this phase consists of adding possible extra features as mentioned in appendix A.8 to the game.

A.1.5. Challenges

When setting up a project, it is best to determine the challenges beforehand, such that sufficient time can be freed for tackling those difficulties. For this project, the following aspects were identified as potential issues.

New language

first of all, no member of the team has any experience with programming in the chosen language of Kotlin. Therefore, setting up the codebase and adding the first few features might take longer than expected. Nevertheless, as can be read in section A.4.1, Kotlin is the most optimal programming language to use for the application.

Security

The second challenge could be the security of the application. As all information of the game will be sent to the server of the client over the internet, people with malicious intent could hack the game. It will be a challenge to get the project production ready.

Testing

Lastly, the team might encounter trouble with testing the final product. This problem can be split into two components; user friendliness and game balancing. As the game is advertised towards a variety of people, mostly people living in Zuid-Holland, it is necessary to make sure that the application is convenient and even easy to use by people of all ages. Therefore, the performance tests have to be performed with a representative group. Preferably, this group contains elderly people (60+), students and adults, younger children (albeit 12+) with the consent and supervision of their parents or guardian. Finally group of representatives should include both men and women, as well as both people living in and outside of Schiedam.
These conditions for performance testing will likely prove to be a challenge, not even considering the extra difficulty of being in a national quarantine during this project. It is currently unclear whether Dutch citizens will be allowed to go outside for non-essential purposes, such as testing an application, or if the lockdown will remain in place once the field testing phase starts.

Additionally, the game balancing can only be tested through playing the game and adjusting different variables. These variables include elements such as the amount of sets per group, the cooldown time between asking for cards, but also whether the addition of having a draw pile makes the game too easy, how certain mechanics can be misused and how one can cheat in the game. However, one game takes between 2.5 and 3 hours to play and a game must be fully played in order to test the different variables properly. Lastly, the play testing cannot be done by the team alone, as an important part of the balancing is dependent on the time it takes to figure out clues and finding locations. After playing the game twice, the team will be familiar with most of the solutions and will therefore not be able to accurately represent a group playing the game for the first time. This is undesirable for playtesting, as most of the users will only play the game once. Thus testing the final product will likely prove to be a real challenge.

The rest of this research report discusses the background, security, technical details, maintainability and interaction design that accompany the digitization of HHF. Appendix A.7 delves into the gameplay mechanics agreed upon with the project owner, and Appendix A.8 provides a full list of requirements derived from this research report.

A.2. Background

In the last few years, creating a game or other application has become increasingly accessible for any person interested in the topic. This rising activity also results in increased difficulty to produce original ideas. In this sea of content, HHF is created in the image of Happy Families, also known as Quartets or Go Fish, while utilising the concept of location-based games. While there are various games using the concept of Go Fish and games that utilise the concept of location-based games, it is the combination of these two ideas that make HHF unique. This chapter focuses on the two main aspects of HHF, namely the location-based aspects and the game mechanics of Happy Families, after which it will explain how HHF carves its niche within the market.

A.2.1. Location-based games

Although location-based games exist in various forms, at its core, each game requires knowledge of the physical location of all of its current players. HHF utilises the location of its users in the same manner as other location-based games. Therefore, it is integral to find the benefits of other games using this mechanic, to apply these to HHF.

One such game is Geocaching. According to Kenton O'Hara, the game is fun to play, as it motivates its players to go outside and have a walk [19]. Similarly, the game Savannah, which similarly incentivises its players to go outside, also enables forming groups with players and subsequently causing a highly physical experience [3]. Lastly, the famous Pokémon Go is a prime example of how a location-based game can thrive on a larger scale by using the largest gaming franchise in the world, simple gameplay, and the aforementioned location-based gaming mechanic as a promotional tool for the game [20].

A.2.2. 'Kwartetspellen'

The Dutch 'kwartet' game has seen various printed versions. A quick Google search for 'kwartetspellen' gives results ranging anywhere between historical trivia, English proverbs and Disney princesses [24]. Certain versions of the game have been applied to more research-oriented fields. Specifically, utilising the ruleset of Go Fish has shown promise in education, specifically to improve language. Timmer et al [25] studied three different games, including 'kwartet', to determine preferences and usefulness in learning French language within high school students. While Memory and Game of the Goose, the other two games included, did show higher preference depending on the study group, 'kwartet' also had a positive effect on studying, especially to the years where knowledge was still limited.

Similarly, Nurhayati [18] sought to prove if the Maze game and Go Fish could be used to pique interest in, and improve the quality of English in kindergarteners. This strategy appeared to be successful, as the children did show more enthusiasm to learn English after the game was played.

However, while generally useful for people of a lower age group, little knowledge gain is proven for Go Fish or 'kwartet' specifically for adults. Still, it is true that gamification in general tends to increase interest [9, 10] and improve learning [12, 13], productivity [26], physical exercise [11] and more, even in adults. Since HHF could be considered gamification of learning the history of Schiedam, it is likely that the general attributes of gamification also translate to HHF to an extent.

A.2.3. The marriage

By utilising the physical experience and social elements brought by location-based games, as well as the simplicity and educational traits of Go Fish-like games, HHF works to create a fun experience for large groups of people. It focuses on the history of Schiedam through its various sightseeing locations. These locations, provided by the client, are the main goals of education for the game. The general gamification of learning about these historical locations, which is supplemented through the main goal of finding the locations, allow for a unique experience for anyone involved in HHF.

A.2.4. Huntin Happy Families' niche

There have been games with the same premise as HHF, which in Dutch markets itself as playing 'levend kwartet' (which translates to: "Alive Go Fish"). However, no other games were found that employ the same core playstyle as HHF. While various Physical Education and scouting games are explained as being 'levend kwartet' as well, these games, aimed at children between seven and fourteen, have relatively simple game rules and elements. More importantly, they focus on physical exercise. They are either intended as a running game [15, 23], or as a simple memory game [22] with elements of (significant) physical exercise.

This is where the differences between HHF and other 'levend kwartet' games are most notable. HHF has a major puzzle solving element, and is not meant for high intensity exercise. It utilises social interaction and teambuilding as its main goals, as well as learning the history of Schiedam. It, too, intends to remain rather simple. However, although it is considered a physical game, physical exercise is not a focal point of the game. Aside from these examples, no similar game could be found that employs Go Fish as its main ruleset in a similar context as HHF does. In short, while there are other games that use the same namesake in Physical Education or scouting games, this particular combination of discovering history, puzzle solving and social interaction appears to be a one of a kind game.

A.3. Security

From a security perspective, the game contains three possible ways of being insecure, both in terms of the application and the concept of the game. This chapter discusses these insecurities, and how to deal with them.

A.3.1. Player information transport

Over the course of the game, players send information on their tablet in regards to their location and chat messages, to the server over a public internet (TCP) connection. Sending this data in a plain way would be insecure. Not only could this enable easier cheating for players, but malicious people could obtain location and chat data from users that is not supposed to be public.
To prevent such data leaks, transferred data can be encrypted before sending this to the server. This way, no attacker should be able to read it without a proper decryption key. Since the client and server devices are physically kept in the same place, the encryption and decryption key can be stored on the client and server devices as well. This is done by storing them in a project file, or possibly hardcoding them. While this solution is not as elegant as transferring the key using a key exchange algorithm, the security implications would be minimal. This is because, as previously mentioned, the physical devices are kept at the company and the application will only be changed by company staff.

A.3.2. Server security

General server security, such as closing vulnerable ports on the server, is not within the scope of this assignment. However, ensuring the application is secure when it is in contact with clients or possible malicious people, is. This means that the server should have good definitions of possible data sent to the server, and reject any data that is invalid or undefined. Most aspects of the game sent to the various systems can be solidly defined, and therefore should enable a fairly secure system.
The only exception to this rule is the player chat feature. Data sent through the chat system can be of variable length and can have any possible value. No exact measures can be defined yet to combat this issue, but emphasis will be put on the manner in which this data is used. Specifically, the data part of the chat system should be designed such that it cannot be misinterpreted by a computer. A possible solution is by putting the message in an SQL query [1]. This way, any malicious or even unintentional code injections can be prevented.

A.3.3. Anti-cheating systems

As with any well thought-out game, the game should be resistant against users cheating or misusing the game. For HHF in particular, the important data that could be cheated with consists of the card collection and the player location. Since the main focus is collecting unique cards, the collection of each team is easy to handle on the server. This results in simple ways to verify mechanics such as uniqueness of cards across a game, the amount of cards per team, and the methods teams can utilise to obtain cards.
The location of a user is more complex to contain, as a malicious user could try and find ways to forge their location. While little can be done on the client side, the main reason that this should not be an issue is because teams are provided tablets, which should disallow users to close the game without a code.

A.4. Technical details

This section discusses the research results for the technical details. Firstly, the platform on which HHF will be run is considered, followed by discussing the optimal programming language to use. Afterwards, a few design ideas on the map and location tracking are reviewed.

A.4.1. Platform and programming language

The game should be developed for the Android platform, as it will be played on tablets using this software. For the development of the application, Android Studio will be used, which is specifically made for building apps for the Android platform. When researching the language used, two options emerged: Java and Kotlin. These are the first and second official language for Android app development, respectively [2]. While the team is more familiar with Java, Kotlin is made especially with interoperability in mind and therefore has access to all libraries and frameworks that Java has. In practice, this means that both languages can use the frameworks that are mentioned in section A.5.3, which is desirable for testing the application. Moreover, Kotlin also offers concise expressions and abstractions. Thus Kotlin makes a developer's job easier and mitigates the risk of errors [2].

Alongside the game, the client has requested a web interface that can monitor games of HHF being played and adjust settings. For the front-end HTML, CSS and Javascript may be the best to use. For the back-end, a combination of Javascript and PHP may be used. As a platform for the web interface, either PhpStorm or InteliJ will be used, based on personal preference within the team.

A.4.2. GPS system

One of the main features of HHF is a map of Schiedam which displays the locations of the other teams. At the same time, location verification is needed to confirm whether a correct location has been found by a team. In this section, multiple alternatives for implementing those features are discussed.

The map

Multiple alternatives are available for showcasing a map in the application. The following four are mainly considered for this project:

Google Maps API[1]. The Google Maps API can easily be integrated in any Android application, whether it is coded in Java or Kotlin. It automatically handles access to Google Maps servers, map display and responses to map gestures in the same way as using Google Maps on a phone would work. However, the API can not be used in this project as it requires a billing account [8].

Mapbox[2]. Mapbox is an open source software API that can be used to display maps in a same manner as the Google Maps API. It allows for static maps, different modes, navigation and includes various other features. Mapbox is certainly less expensive that the Google Maps API. Mapbox even states that it can be used for free for up to 25,000 users. However, assuming this implies a maximum of 25,000 monthly polls per user, this will still cost the client money if each team has to update their location every ten seconds.

OpenStreetMap[3]. OpenStreetMap has an editing API for fetching and saving raw geodata from the OpenStreetMap database, meaning that it can display a realistic map of Schiedam in the application. It is free to use for any purpose as long as OpenStreetMap is credited. However, the documentation found for OpenStreetMap appears lacking and the API seems less developed in general.

Screenshots. Instead of using an API, it would be possible to create a map of Schiedam based entirely on screenshots instead. This concept has already been implemented in the other application of the client: Hunt the Hunter. In order for this to work in combination with the location tracking of teams, the coordinates of each corner of the map must be known. Then, given a longitude and a latitude, any place on the map could

[1] https://developers.google.com/maps/documentation/
[2] https://www.mapbox.com/
[3] https://www.openstreetmap.org/

be marked with simple calculations. This is the preferred method of implementation. While this method is a little more 'dirty', as it hardcodes a map and is inflexible in its boundaries, a static map is preferred for simplicity, and this solution is less complex in general.

The locations

First, the method to obtain the location of a team at any moment needs to be determined. In other words, the location of their device should be tracked. Android does offer functionality to periodically request the location of the system in use. This can be performed by the Fused Location Provider API [7]. This class contains functions for retrieving the last known location of the device, and may request location updates from a certain device. Using the locations provided, the longitude and latitude can be retrieved, as well as the distance to another specified place. The location of each team can be displayed on the map as described in the section above.

Additionally, the location of a team also has to be compared to the coordinates of the locations associated with the card sets. This does not happen in real time. Instead, the coordinates of a team are only compared to the coordinates of the set locations after the team has pressed a button in the application. When this occurs, there are three possible outcomes: The team verifies a correct location, and collects points, the team verifies a correct location for the wrong set of cards, or the team verifies a completely wrong location. The correct locations are stored within an array, each with coordinates to project onto the map if needed.

A.5. Maintainability

Maintainability is an important factor in the development of an application, which is also the case with HHF. Whilst working on the project, the code should be easy to work with to make sure different tasks can be done in an efficient manner. Additionally, the code needs to be maintainable for other developers that might work for Cityhunter after this project is finished. Cityhunter currently has no employed developers, so the code and the documentation should be as self-explanatory as possible. This section goes into detail on some aspects to achieve a good maintainability [5].

A.5.1. Development methodology

During development of HHF, various development methodologies can be used. The three most common methodologies are Waterfall, Scrum and Kanban, each with their own advantages and disadvantages. This project will use Scrum, as it allows for defined requirements at the beginning of a project. These requirements can still change if needed according to the client or based on the results of the field testing phase. Additionally, Scrum requires the team to meet weekly.
Due to current global circumstances, physical meetings are impossible during most of, if not the entirety of this project. Therefore, the team will meet at the start of the day on a voice chatting program such as Discord to discuss progress and future tasks. Additionally, the team will have weekly meetings with the client and their supervisor using video chatting programs such as Jitsi or Zoom.

A.5.2. Code quality

In order to ensure the quality and maintainability of the code, the following methods will be used and carried out during the project:

Version control. Using version control allows programmers to easily share their code and undo changes. More importantly, version control can be used to find and blame the person that broke the code. The branches will be managed according to the Git-flow method, using a master branch, a development branch, feature branches, hot-fix branches and release branches.

Documentation. Each function will be documented, as documentation allows for other developers to have a better insight of what is happening, making their tasks possibly easier and less time consuming.

Compact code. Code should be kept compact. Code should not contain unnecessary fragments and duplicate code fragments should be avoided. For example, code segments that are used multiple times should be packed into functions instead. Having this specific code in one place simplifies debugging, as only the function needs to be adjusted rather than the same code segment in different areas within the code base.

Code review. By having a pull request reviewed by multiple people, the quality of the code can be significantly improved. This is also a good test whether or not the code is understandable.

Resolving issues. Rather than treating the symptoms, the real issue causing a bug should be resolved.

Static code analysis. Static code analysis is analysis of the code without actually executing it. Several tools will be used to perform this code analysis, such as PMD[4], Checkstyle[5] and Findbugs[6]. Each of these tools is available when coding in Java, and should therefore also be available when coding in Kotlin.

Testing code. Every function should be unit tested to ensure that it works as it should. However, only unit testing is not enough to confirm that the application works without errors. More detailed testing practices are described in the following section.

A.5.3. Software testing methodology

Software testing methodology is defined as strategies and testing types used to certify that the application meets the client expectations and is working properly. In this project, the following methodologies will be

[4] https://pmd.github.io/
[5] https://checkstyle.sourceforge.io/
[6] http://findbugs.sourceforge.net/

utilised for the certification:

Unit testing. Unit testing is a form of white box testing. The purpose of unit testing is to verify that each component of the software code performs as expected, where a component can be an individual function, method, module or object. By writing tests for the smallest testable units first, one can build up comprehensive tests for complex applications. Each test case is tested independently, to help isolate issues that might arise. In combination with JUnit5[7], The Mockito framework[8] will be used.

Scenario testing. In a game of HHF, various scenarios may occur. Recreating all of these might not be possible during playtesting. Additionally, waiting until the field testing phase to assess those various scenarios is inconvenient. Cucumber[9] is a white box testing software tool that supports behaviour driven development and it can be used to create scenarios that will test different edge cases. Therefore, Cucumber will be used in this project to perform scenario testing.

Integration testing. Whereas unit testing is used for testing components individually, integration testing is used to expose defects in the interaction between software modules when they are integrated. However, it is still a form of white box testing. Integration testing will be applied during this project with an incremental approach, making use of stubs and drivers.

Performance and field testing. Performance testing checks the speed, response time, reliability and scalability of a software program under the expected workload. Field testing, on the other hand, is done to validate the behavior or usability of an application as an end user. Both are a form of black box testing, meaning that the tester does not know the implementation of the code. Both will be done during the field testing phase in weeks 7 and 8 of the project.

[7] https://junit.org/junit5/
[8] https://site.mockito.org/
[9] https://cucumber.io/

A.6. Interaction Design

When creating a product, it is of the upmost importance to think of the needs of the users. If the users are not satisfied with the product, they might not want to use it again, which is bad for business. Interaction design focuses on usability and user interaction with the design of the product, with the intent of satisfying players when interacting with the final result. While interaction design also deals with creating a goal-oriented design, this will not be debated, as the game is already established and in use. Instead, this chapter discusses the other key elements of user experience design, as well as the design principles that will be implemented in the application of HHF.

A.6.1. User experience

The core design principles of a good usability can be described with the following words: effectiveness, efficiency, learnability, error tolerance and engagement [17, 21]. The application of HHF must be effective: the application must help the user by making it easier to find locations based on traded cards and their clues. It should not make it harder for the user to trade cards via a tablet than trading cards by hand. Additionally, an application should also be efficient in order to be effective. In other words, an action should not cost too many clicks. Thirdly, as the game will most likely only be played once by a user, the interface should be intuitive and easy to learn. This also means that features should not be hidden away. Next, the application must have a certain error tolerance. Actions performed within the application should always be reversible or at least not be punished. In this game, that principle is applied to verifying locations and trading cards, as those are the only significant and possible irreversible choices a team has to make through the application. Lastly, the application must be engaging. While this is mostly dependent on gameplay, the application can add to it by increasing the feel of friendly competition. Thus having a viewable ranking page and possibly an additional activity log should entertain the user in this regard.

A.6.2. Interaction design for the elderly

HHF is aimed at a variety of groups ranging in age from 8 to 88, meaning that the application has to be usable and enjoyable by different generations. Whereas people within the age groups between young children and adults are increasingly well versed in the world of technology, this is not the case for relatively older generations. Gaming especially requires a different view of interaction design when it comes to the elderly. This is because they often experience cognitive and physical limitations. Another less mentioned problem is the motivation for elderly people to play a (mobile) game [14]. However, this should not be an issue here as being able to play with their grandchildren often is encouraging enough [4]. Nevertheless, the game can only truly be a family game if usability problems for all players are taken care of. Multiple studies have shown that the elderly:

- Prefer games with less controls and options [16].
- Tend to forget instructions and rules easily [16].
- Feel less confident when playing games that require a quick reaction [14, 16].
- Prefer games without animations and visual effects [6, 16].
- Do not like games with small, unreadable text [14].

A.6.3. Design principles

From the results of this study, the following key design principles are to be incorporated into the final design of the game:

- The game shall not include visual effects or animations.
- The application shall have an easy navigation to prevent too many clicks; the home screen will have a fixed menu to navigate to other pages, and each page will lead back to this home screen through a clear, simple button.
- The game shall provide an FAQ page with a simple overview of the instructions and rules.
- The size of the text in the menu, chat, FAQ page, ranking page and activity log can be adjusted.

- The card display will use intuitive colors and contrasts to show which set is selected and which cards are already collected by the team.
- Each card in the card display screen will be colored and titled with the color of the card, to cater to color blind people.
- Each card in the card display screen can be enlarged by clicking on the card, such that the clues are easily readable.
- The map screen will have a zoom function.
- Asking the other team for a card can always be canceled during the process. It can, however, not be reversed once the request is made in order to prevent cheating.
- Verifying a wrong location shall not be punished immediately. This way, accidentally clicking the button will not lead to unwanted consequences. However, to prevent cheating, it will be punished when too many wrong attempts have been made.

These design principles are based on the needs of players, while keeping the overall aspect simple. This way, players will not waste time trying to figure out how to work with the app, and instead have more time to focus on the game itself; Learning more about the history of Schiedam and socializing with the other players in the group.

A.7. Game play

In a game of Hunting Happy Families, teams compete against each other by collecting as many points as possible, by finding locations with the help of clues and by completing sets of four cards. This appendix dives into the general requirements of the game.

Start of the game

Before the start of the game, each team gets to choose a team name (which can be changed by the game master if it is offensive, or has a typo). As soon as the game starts, each team receives the same arbitrary amount of unique playing cards, based on how many teams are playing. These cards are chosen from different sets of four, each with a different color. Aside from the cards distributed over the teams, there will be a draw pile that holds another arbitrary amount of cards. All teams start at the same location. This location is not one of the solutions of a set in the game.

A game takes between 2.5 and 3 hours to play.

Gathering and exchanging cards

Teams are supposed to collect as many points as possible by completing color sets and finding the location that the information of the cards in the set leads to. Locations can be found without having all the available information, but it becomes easier when more cards of a set are collected. Therefore, teams should walk around the city and interact with other teams.
As soon as two teams are within ten metres of each others, the tablets will give them the option to ask the other team for a card. A team can only ask for a card of a color category of which they also own a card. If a team possesses the requested card, the ownership of that card will be transferred to the other team. The information remains available for both teams. If a team does not have the requested card, the other team receives a card from the drawing pile. If there are no cards remaining in the drawing pile, the team will instead receive information, but not the ownership, of a random card in play that they have not yet seen the information of. Lastly, teams can also choose to not ask the other team for a card. They cannot, however, refuse teams to take a card from their hand. After the interaction, both teams cannot interact with each other for five to ten minutes.

Following these mechanics, there are a few edge cases that need to be clarified. First of all, if a set of four is completed, cards of that color can no longer be taken by other teams. This includes the random information obtained when the draw pile is empty. However, teams that collected some of the information on the cards of the completed set should still be able to find the location that belongs to that color and therefore still get those points. Secondly, if a team no longer owns any cards aside from complete sets, if any, they will receive a card from the drawing pile. If there are no cards left in the drawing pile, they will have the option to ask for an "empty card", or a card the other team can never own, to simulate asking for a card the other team does not have.

Considerations

The following list defines alternatives discussed and explains decisions made for design and mechanics choices in card gathering and exchanging.

- In accordance with the demand of the project owner, the game will not allow for requesting cards of a color you do not own. This is also done to more closely mirror the original game of Happy Families.
- Similarly, the choice for a draw pile was made in accordance with the preference of the project owner. By doing so, all teams are less dependent on their initial hand, and the flow of information is larger than when the full set of cards is distributed instead. While this does not mirror classical happy families, the project team and project owner both believe this to be a more fun mechanic than not having a draw pile.
- There have been a few iterations for the mechanics when the draw pile has been emptied.
  - Instead of providing the information of a random card in play, the contents of a card in the opponent's hand would be provided instead. This way, other teams would not be involved in the

interaction. However, the current system is preferred as this does not provide extra info on the hand of the opponent, which could give some unfair advantages to the team that missed their card pick.

- A different approach is to not give anything when the draw pile is empty, but full sets would be shuffled back into the draw pile. If a team completed a set, completing that same set again would not provide any extra points.
  This system allows for an alternative way to keep information flowing. However, this method results in more edge cases, and would likely be more complex to players. For this reason, it was not chosen to combat an empty draw pile. If, however, it turns out the currently used "empty card" mechanics would result in unfairness or less fun scenarios, this solution would be used instead.

- The "empty card" mechanic was chosen to keep the flow of information going, and to give teams that are behind a way back into the game, even if their card pool is depleted and the draw pile is empty.

Locations

Each set has a color as category, with the following four sub cards:

- Historic information about the location;
- A piece of a map that shows the location;
- A picture of a detail of the location;
- A picture of the location.

Points can be collected by finding the location that corresponds to a set of cards. However, a set does not have to be complete in order to find the correct location. If a team finds and verifies a wrong location, nothing will happen initially. However, to prevent potential cheating, teams that make too many wrong guesses will be punished, by means of disabling their option to verify a location for a few minutes.

Considerations

Three other ideas have been explored as punishment for a wrong attempt on guessing a location. It was clear to the team that a sanction was needed as repeatedly guessing a location that could be one of the sets' location would be an easy way to cheat in the game.
The three other solutions were locking the location, locking the set, and deducting points. The first was not decided upon as the location are not marked on the map. The second was decided against as it would go against the principle of learning about the history of the city. Finally, the third option was considered discouraging for teams, and was therefore not chosen as punishment. The current sanction is mild for your final results, but does not allow for huge repeated guessing, as that would rapidly stale gameplay for those that decide to try this tactic.
The choice to allow for multiple guessing before locking down is in accordance with the requests of the project owner. Furthermore, this allows for some leniency in case of hard- and software related inaccuracies.

Application navigation

On the player side of the game, the tablet application will have to following screens:

- A home screen with a little map and a menu leading to the other screens.
- A display of cards and information gathered.
- A chat to communicate with specific teams, all teams and the game master.
- A map of Schiedam that shows the locations of all the teams.
- Rankings and an activity log.
- An FAQ page.
- A camera.

Web interface usage

To allow the game master some flexibility prior to and during the game, as well as to simplify interaction with the game master, they will have access to a web interface. This interface include functionality for the following elements:

- Look at an overview of any ongoing games. That is, for each game:
  - The game master can see the current location of each team.
  - The game master can see the current collection of each team, both in terms of cards and information.
  - The game master can see the current ranking of the teams.
  - The game master has access to the full activity log.
  - The game master can see all pictures that are taken with the tablet.
  - The game master can read messages sent between teams on the tablets. This is needed to monitor potential cheating.
  - The game master can change any team name they consider to be offensive.
- Chatting with certain teams within the chat functionality of the game.
- Adding sets to the card pool of an ongoing game.
- Creating and adding new sets, modifying current sets and deleting sets that are no longer relevant.
- Picking sets prior to creating a game.

Considerations

Each element which requires functionality in the web interface for the game master has been requested by the project owner. The game master has access to private messages sent within the game's chat function to keep a closer eye on cheating.

A.8. Final requirements

This [section of the] appendix includes all MoSCoW requirements that have been decided upon through consultation and discussion with the client. Each requirement is either a "must have", "should have", "could have" or "won't have" requirement. This way, the team knows to focus on the "must have" requirements first, after which it can start implementing "should have" requirements. The "could have" will only be implemented if the project progresses faster than anticipated. No "won't have" features are included in this appendix.

Must have requirements

The game must have the following features:

- The game shall be playable by multiple groups simultaneously.
- The game shall be playable on Android version 5.0 and higher.
- The user shall not be able to leave the application without a code.
- The application shall feature a map of Schiedam with live locations of all teams.
- One game shall be playable with 4 to 10 teams.
- The application shall provide a team with a starting location, as chosen by the game master.
- The game shall provide a pre-selected amount of sets when starting a game.
- The game shall contain a game timer visible by all teams, which can only be started by the game master.
- The application shall feature a chat function.
  - The chat functionality shall feature communication between one team and the game master.
  - The chat functionality shall feature communication between all teams and the game master.
- The application shall include camera functionality.
  - This includes taking a picture and sending that picture to the game master.
- The application shall feature rankings that are visible to the teams.
- The rankings shall not be visible during the last 30 minutes of a game.
- The application shall have a display for seeing your own team's collected cards and clues.
- The application shall enable teams to exchange cards following the rules as mentioned in appendix 3.
- The application shall be able to send images to the game master.
- Upon ending the game, the application shall disable all trading functionality.

The web interface must have the following features:

- The game master shall be able to add and remove new cards before starting the game.
- The game master shall be able to adjust clues and information on the cards.
- The game master shall be able to remove sets of cards.
- The game master shall be able to set the game timer.
- The game master shall be able to monitor all games being played. That is, for each game:
  - The game master shall be able to see the current location of each team.
  - The game master can see the collection of each team, both in terms of cards and information.
  - The game master shall be able to see the current ranking of the teams.

  - The game master shall be able to see all pictures that are taken with the tablet. This feature is still debated for privacy reasons, but is nonetheless written down.
  - The game master shall be able to read messages sent between teams on the tablets. This is needed to monitor potential cheating.
  - The game master shall be able to change any team name, if they find it offensive.
- The game master shall be able to communicate with all teams.
- The game master shall be able to communicate with one team specifically.
- The game master shall be able to add a set of existing cards while a game is being played.
- The game master shall be able to award points to players for correctly identifying set locations.

Should have requirements

The game should have the following features:

- Each team should be able to enter a team name.
- The application should have an FAQ page.
- The application should have an activity log displaying at least all interactions of a player's own team.
- The game should be able to award points for finding and verifying a correct set location with the accompanying set, in case the game master is not able to monitor a full game.
- The chat should display notifications for new, unread messages.
- the chat functionality should feature communication directly between two teams.
- The game should have a cool down time for exchanging cards between two teams.
- The game should include a cool down time as penalty for too many wrong attempts at verifying a set location.
- The game master should be able to adjust the amount of sets in a game.

Could have requirements

- The game could include sounds.
- The game could feature bonus exercises that allow teams to collect more clues.
- The game could have an English translation.
- The application could include a QR code for every card in the game, that can be scanned by other systems not part of the game.
- The map could show set locations on the map after they are found.
- The game could censor bad words contained within chat messages. The definition of 'bad word' is to be defined by the client.
- The game could disallow team names that contain a bad word. The definition of 'bad word' is to be defined by the client.
- The game could include an animation showcasing a team's movement throughout the game, after the game has ended, for that team to take home.
- The game could include a story showcasing a team's progress throughout the game, after the game has ended, for that team to take home.
- The game could include image recognition software for identifying the set locations compared to the sent images.
- The game could include a layout for both horizontal and vertical landscape.

B

Requirements review

This appendix evaluates all requirements as laid out in appendix A.8, based on if it has been met, and also provides an explanation of each function's implementation.

Requirement	Met?	Explanation
Must-have requirements		
The game must have the following features.		
The game shall be playable by multiple groups simultaneously.	Yes	Teams not part of the same game are incapable of seeing one another within the game. Cards could be duplicate between two different games. The game master can swap between games and will see the respective overview within the web interface.
The game shall be playable on Android version 5.0 or higher.	Yes	The game was implemented with Android version 5.0 in mind, though the tablets used for the game are of version 6.1 instead.
The user shall not be able to leave the application without a code.	Yes	This functionality was provided by the client, as this was also used for a previous game .
The application shall feature a map of Schiedam with live locations of all teams.	Yes	The game has been tested in Delft, which also includes a map of Delft. This also means the game could be created in different locations as well, as long as a proper map and proper sets are provided for said location.
One game shall be playable with 4 to 10 teams.	Yes	This has been expanded to allow up to 14 teams, though this is not recommended.
The application shall provide a team with a starting location, as chosen by the game master.	No	Rather than giving each team an individual location, a central location is decided by the game master.
The game shall provide a pre-selected amount of sets when starting a game.	Yes	While a recommended amount is provided, this is adjustable by the game master.
The game shall contain a game timer visible by all teams, which can only be started by the game master.	Yes	The timer will become visible once the game master has started the game. Before that, the game is indicated to not have started for teams.
The application shall feature a chat function:	Yes	This is accessible from the main menu.
The chat functionality shall feature communication between one team and the game master.	Yes	
The chat functionality shall feature communication between all teams and the game master.	Yes	
-		
The application shall include camera functionality:	Yes	This functionality is exclusively used for sending images to the game master.
This includes taking a picture and sending that picture to the game master.	Yes	
-		
The application shall features rankings that are visible to the teams.	Yes	This is accessible from the main menu.
The rankings shall not be visible during the last 30 minutes of a game.	Yes	The ranking is set to 0 for the last 30 minutes.
The application shall have a display for seeing your own team's collected cards and clues.	Yes	This is used as the main screen.
The application shall enable teams to exchange cards.	Yes	If teams are: - Within 20m of each other, - Are not already in another trade, - Have not traded recently. The teams can trade with each other.
Upon ending the game, the application shall disable all trading functionality.	Yes	This is also disabled before a game has started.

Requirement	Met?	Explanation
The game master web interface must have the following features:		
The game master shall be able to add and remove new cards before starting the game.	Yes	Incomplete sets created by the client cannot be part of a game until they are completed.
The game master shall be able to adjust clues and information on the cards.	Yes	This also include the name of the set, the answer of the set, the hex code, the location, and the type of clue the card is.
The game master shall be able to remove sets of cards.	Yes	
The game master shall be able to set the game timer.	Yes	When creating a new game, this is part of the procedure.
The game master shall be able to monitor all games being played. That is, for each game:	Yes	
The game master shall be able to see the current location of each team.	Yes	
The game master can see the collection of each team, both in terms of cards and information.	Yes	
The game master shall be able to see the current ranking of the teams.	Yes	The amount of points of a team is also showcased at their respective overview.
The game master shall be able to see all pictures that are taken with the tablet.	Yes	Pictures can exclusively be made for verifying locations.
The game master shall be able to read messages sent between teams on the tablets. This is needed to monitor potential cheating.	Yes	
The game master shall be able to change any team name, if they find it offensive.	Yes	While most likely used for correcting typos, this is included.
-		
The game master shall be able to communicate with all teams.	Yes	This is done by using the public chat, which every team is a part of. A pop-up is sent to the teams when the game master sends a message.
The game master shall be able to communicate with one team specifically.	Yes	
The game master shall be able to add a set of existing cards while a game is being played.	Yes	
The game master shall be able to award points to players for correctly identifying set locations.	Yes	A team sends an image, the game master approves or disapproves the image. The points are then updated on the server, consequently at the game master and the players (if it is not the last half hour).

Requirement	Met?	Explanation
Should-have requirements		
Each team should be able to enter a team name.	Yes	Creating a team name is used as the method to join an open game.
The application should have an FAQ page.	Yes	This can be accessed from the main menu.
The game master should be able to adjust the FAQ page.	Yes	
The application should have an activity log displaying at least all interactions of a player's own team.	Yes	This can be seen on the same page as the ranking functionality.
The chat should display notifications for new, unread messages.	*Yes*	This is only the case when the game master sends a message. When another team sends a message, no notification is sent. When the team is already on the chat screen, no notification is sent for the game master either.
The chat functionality should feature communication directly between two teams.	Yes	
The game should have a cool down time for exchanging cards between two teams.	Yes	
The game should include a cool down time as penalty for too many wrong attempts at verifying a set location.	Yes	Images sent by a team must first be approved before another can be sent for the same set. If a team has already correctly identified a set, they cannot
The game master should be able to adjust the amount of sets in a game.	Yes	

Requirement	Met?	Explanation
Could-have requirements		
The game could include sounds.	No	*All of these functionalities were scrapped to focus on more important issues.*
The game could feature bonus exercises that allow teams to collect more clues.	No	
The game could have an English translation.	No	
The application could include a QR code for every card in the game, that can be scanned by other systems not part of the game.	No	
The map could show set locations on the map, after they are found.	No	Effort has been put into this requirement, and showing set locations is implemented for the game master, but no such functionality was included for the teams.
The game could censor bad words contained within chat messages. The definition of 'bad word' is to be defined by the client.	No	
The game could disallow team names that contain a bad word. The definition of 'bad word' is to be defined by the client.	No	
The game could include an animation showcasing a team's movement throughout the game, after the game has ended, for that team to take home.	No	
The game could include a story showcasing a team's progress throughout the game, after the game has ended, for that team to take home.	No	
The game could include image recognition software. for identifying the set locations compared to the sent images.	No	
The game could include a layout for both horizontal and vertical landscape.	No	This requirement has been looked into, but was ultimately ignored to focus on more important requirements.

C

Project Description

Hunting Happy Families – a new group-based mobile game to explore the city Schiedam

Cityhunter organises group activities in which the history of the city Schiedam is explored through the use of active competitive games.

One of the games already offered is the game 'Hunt the Hunter' (http://www.huntthehunter.nl/). In this game every group has to hunt and catch a member of another group. To get knowledge of the person to catch, the group has to answer questions of which their answers can be found around the city. This whole game is played through the use of tablets with an internet/GPS connection.

Cityhunter wants to deploy a new game called "Hunting Happy Families" (op jacht naar een kwartet) that can be played on tablets, similarly to Hunt he Hunter is played. The groups have the goal to perform as much tasks around the city to earn points. Hints for these tasks are contained in (virtual) cards (like a picture, location, detailphoto, etc.), which the groups can exchange with each other while playing the game. At the moment the game is played with physical cards, but the aim is to integrate this in a mobile game which can be managed by the organisation via a separate view.

D

Info Sheet

Project title: Hunting Happy Families
Client organization: Cityhunter
Final presentation date: 02-07-2020

Project description
To expand on their current list of games, Cityhunter wants to create the app Hunting Happy Families. The team is expected to create a fully functioning game from the ground up, including creating a full design of the game.
The product would first undergo a research phase, followed by an implementation phase, a field testing phase, and a second implementation phase. Over the course of the project, the two implementation phases melded into one field testing phase.
During the research phase, the team explored various facets of the game, the most notable of which is the choice to utilise the programming language of Kotlin, as it is the biggest language in Android app development.
The final product was a system with a server, a web interface and an android app. These were eventually playtested using volunteers sought by the team, as well as between the team members.
The current product is ready for use, but still lacks some finer details the team would like to include. The client is fully capable of finding the balance of the game with the customisation provided by the system and he intends to use this product for their future games of Hunting Happy Families.

Team members and their main contributions

- Ruben Band: Developer. Responsible for the chat system, game master functionality and JUnit testing.
- Koen du Buf: Developer. Responsible for connection between app, server and game master, as well as trading and verification functionality.
- Bob Dorland: Developer. Responsible for the map functionality, and most functionality of the Game Master interface.
- Quintin van Leersum: Product owner, developer. Responsible for communication, writing and documentation. Main writer of all reports.
- Emma Sala: Developer. Responsible for the lay-out and functionality of the application's front-end including connection with the server, scenario testing and field testing.

Contact information

Marcel de Bruin,	Owner of Cityhunter,	e-mail: Info@cityhunter.nl
Gosia Migut,	ISPR Laboratory at TU Delft,	e-mail: m.a.migut@tudelft.nl
Quintin van Leersum,		e-mail: quintin@quicknet.nl

The final report for this project can be found at: `http://repository.tudelft.nl`

Bibliography

[1] Aljawarneh, S., Alkhateeb, F., and Al Maghayreh, E. (2010). A semantic data validation service for web applications. *Journal of Theoretical and Applied Electronic Commerce Research, ISSN 0718-1876, Vol. 5, Nº. 1, 2009, pags. 39-55*, 5.

[2] Banerjee, M., Bose, S., Kundu, A., and Mukherjee, M. (2018). A comparative study: Java vs kotlin programming in android application development. *International Journal of Advanced Research in Computer Science*, 9(3):41.

[3] Benford, S., Rowland, D., Flintham, M., Drozd, A., Hull, R., Reid, J., Morrison, J., and Facer, K. (2005). Life on the edge: supporting collaboration in location-based experiences. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 721–730.

[4] De Schutter, B. and Vanden Abeele, V. (2008). Meaningful play in elderly life. *Annual Meeting of the International Communication Association*.

[5] Gabriela Motroc (2016). 6 tips for writing more maintainable code. Available at https://jaxenter.com/6-tips-writing-maintainable-code-124700.html.

[6] Gerling, K. M., Schulte, F. P., and Masuch, M. (2011). Designing and evaluating digital games for frail elderly persons. In *Proceedings of the 8th international conference on advances in computer entertainment technology*, pages 1–8.

[7] Google (2020a). Apis for android. Availble at https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient.html.

[8] Google (2020b). Maps sdk for android. Availble at https://developers.google.com/maps/documentation/android-sdk/intro.

[9] Hamari, J. (2013). Transforming homo economicus into homo ludens: A field experiment on gamification in a utilitarian peer-to-peer trading service. *Electronic commerce research and applications*, 12(4):236–245.

[10] Hamari, J. (2017). Do badges increase user activity? a field experiment on the effects of gamification. *Computers in human behavior*, 71:469–478.

[11] Hamari, J. and Koivisto, J. (2015). "working out for likes": An empirical study on social influence in exercise gamification. *Computers in Human Behavior*, 50:333–347.

[12] Hamari, J., Shernoff, D. J., Rowe, E., Coller, B., Asbell-Clarke, J., and Edwards, T. (2016). Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning. *Computers in human behavior*, 54:170–179.

[13] Herger, M. (2012). Gamification facts & figures. *Enterprise-Gamification. com*.

[14] Ijsselsteijn, W., Nap, H. H., de Kort, Y., and Poels, K. (2007). Digital game design for elderly users. In *Proceedings of the 2007 conference on Future Play*, pages 17–22.

[15] Mediaraven vzw (2020). Levend kwartet. Available at https://www.spelensite.be/spel/levend-kwartet.

[16] Mubin, O., Shahid, C., and Al Mahmud, A. (2008). Walk 2 win : towards designing a mobile game for elderly's social engagement. In *BCS-HCI '08 Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction - Liverpool, United Kingdom, September 01 - 05, 2008*, volume 2, pages 11–14. British Computer Society (BCS).

[17] Nielsen, J. (1994). *Usability engineering*. Morgan Kaufmann.

[18] Nurhayati, D. A. W. (2015). Improving students' english pronunciation ability through go fish game and maze game. *Dinamika ilmu*, 15(2):215–233.

[19] O'Hara, K. (2008). Understanding geocaching practices and motivations. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1177–1186.

[20] Paavilainen, J., Korhonen, H., Alha, K., Stenros, J., Koskinen, E., and Mayra, F. (2017). The pokémon go experience: A location-based augmented reality mobile game goes mainstream. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 2493–2498.

[21] Quesenbery, W. and Design, W. I. (2003). Dimensions of usability: Defining the conversation, driving the process. In *UPA 2003 Conference*, pages 23–27.

[22] Scoutpedia (2020a). Levend kwartet geheugenspel. Available at https://nl.scoutwiki.org/Levend_Kwartet_(geheugenspel).

[23] Scoutpedia (2020b). Levend kwartet renspel. Available at https://nl.scoutwiki.org/Levend_Kwartet_(renspel).

[24] Speelkaartenwinkel (2020). Kwartet. Available at https://www.speelkaartenwinkel.nl/kaartspellen/kwartet.html.

[25] Timmer, J. J. (2008). Een potje kwartet met een 30 koppig monster. Master's thesis.

[26] Zichermann, G. and Cunningham, C. (2011). *Gamification by design: Implementing game mechanics in web and mobile apps*. O'Reilly Media, Inc.