

Technical Report II

Prüfung

Lecturers: Reto Ferri, Johanna Decurtins

Subject: PM3, ZHAW

Team member: Erman Zankov,
Gökhan Bag,
Joël Plambeck,
Lukas Zoss,
Nikita Smailov,
Tobias Ritscher,
Samuel Stalder

Date: 08.12.2020

Abstract

The technical report 2 offers an overview of the elaboration and implementation of the software solution Frantic. Seven team members worked for one semester on this application.

The project can be divided in three milestones. Milestone one includes the idea and the project sketch, milestone two was all about the beginning of the technical report and the code setup, and milestone three is about the technical report 2, the final presentation and the beta-release. The planning is based on iterative-incremental software development process and took place every second week together with the supervisor.

This report documents the developed software architecture and the result of the beta-release. It covers the singleplayer modus against computer opponents, the multiplayer modus in local network and the development of a graphical user interface (GUI) with JavaFX.

The result of the beta-release is a playable software representation of the card game Frantic. The singleplayer as well as the multiplayer modus are working fluently with a subset of all cards. Not all special and event cards are implemented yet, but with the developed architecture, all further parts can be added without big effort in the next release.

Keywords: Java, Frantic, Card Game, Beta-release, Software Development

Content

1	Project idea	1
1.1	Introduction and Idea	1
1.2	Customer values	1
1.3	Market analysis	1
1.4	Main procedure	2
1.5	Feasibility	2
2	Analysis.....	3
2.1	Use-Case Model	3
2.2	Supplementary Specification	6
2.2.1	Functionality	6
2.2.2	Usability	6
2.2.3	Reliability	6
2.2.4	Performance	6
2.2.5	Supportability	6
2.2.6	Implementation Constraints.....	6
2.2.7	Free Open Source Components.....	6
2.2.8	Legal Issues	6
2.2.9	Additional Card Rules	7
2.3	Domain Model	10
3	Design.....	12
3.1	Package diagram	12
3.2	State Machine diagram.....	13
3.3	Interaction Diagram	14
3.4	System Sequence Diagram.....	15
3.5	GUI	16
3.6	DCD	18
3.6.1	AI.....	19
3.6.2	MVC	19
3.6.3	Design patterns	20
3.6.4	GRASP	20
4	Implementation.....	21
4.1	Testing.....	21
4.1.1	JUnitTests	21
4.1.2	Integration test.....	23

4.1.3	System test	24
4.1.4	GUI test.....	25
4.2	Installation guide	26
5	Results	27
5.1	Further development goals:	29
6	Attachment	30
6.1	Project management	30
6.1.1	Team structure	30
6.1.2	Effort estimation.....	30
6.1.3	Risks.....	34
6.2	Glossary	34
7	Appendix	35
7.1	<i>List of figures</i>	35
7.2	<i>List of tables</i>	35
7.3	<i>References</i>	36

1 Project idea

1.1 Introduction and Idea

The Swiss card game “Frantic”[1], developed by four friends in St. Gallen, was released in 2015. The game has a successful history including over 100’000 sales. Successful enough for “Carletto”, a Swiss toy and game distributor, to purchase the selling rights and launch the game in Austria and Japan. [2] Unfortunately, due to the pandemic in the year 2020, social gatherings for card games are not recommended or even prohibited. In order to continue playing Frantic during social distancing times, a digital online version of the game is needed. It is an identical copy of the physical card game with all its cards and rules.

Playing a card game online is not nearly as enjoyable as playing in person with your friends. Although, it is better than not to play at all. This allows you to have all the fun without risking your or your friend’s wellbeing. In addition, it also allows you to play with your soon to be friends from anywhere in the world.

1.2 Customer values

It is the only digital version of Frantic, the mischievous card game, on the market. Frantic will destroy friendships now all over the world. And that’s ok.

- There is a multiplayer with up to eight Players. The matchmaking algorithm allows game sessions with players around the globe.
- It is easy to start game sessions with friends. They can join using the generated QR-code or through the invitation link being sent to the email-address.
- A wide range of devices is supported.
- There is no need for an account on any video game distribution platform. The game is ready to play after being downloaded through the website.
- The game can be easily started at any time and location.

1.3 Market analysis

While there are many different online/digital tabletop games available for computers, frantic isn’t one of them. A direct competitor is therefore as of yet not on-hand. Therefore, possible competition only comes from other popular card and tabletop games. For the market analysis two other online card games were used: UNO and Tabletop Simulator. [3]

UNO is experiencing connectivity issues between the players, which makes it harder and less desirable to play. Tabletop Simulator is a collection of different games such as UNO, but it doesn’t offer single player mode and it requires a Steam account.

1.4 Main procedure

The main procedure of the game is as following:

1. The player decides between single and multiplayer mode.
2. In multiplayer mode they can decide between creating a game and joining one.
3. The length of the game can be set by the players themselves and it regulates the required points to win.
4. As the game begins the cards are being dealt and a card is being placed on the deposit pile.
5. Each playable card on the players hand is highlighted.
6. If the player has no playable cards, they must draw.
7. If the drawn card is playable, they can choose between playing it (or another card on their hand) or ending their turn.
8. If a black card is played an event is being played out.
9. The round finished the moment a player has no more cards.
10. At the end of each round the points are counted.
11. The next round starts by the player with most points from the previous round.
12. The game ends the moment one player reaches the maximum points defined at the start of the game

1.5 Feasibility

The estimated effort is 700 hours for a team of seven developers. For an hourly rate of CHF 120 the total development costs would be $120 * 700 = 84'000$. When pricing the game at CHF 15.00 it would need 5'600 sold beta-release versions to recoup the costs.

Description of earnings	Amount in CHF
Product sales (pre-order)	84'000
Product sales (without pre-order)	100'000
Merchandising	5'000
Sponsors	7'000

Table 1: List of all earnings

Description of expenditures	Amount in CHF
Personnel costs (till beta release)	84'000
Personnel costs (after beta release)	84'000
Marketing	10'000
Events	5'000
Server costs	5'000
Infrastructure	1'000

Table 2: List of all expenditures

Description sum	Amount in CHF
Earnings	196'000
Expenditure	189'000
profit	10'000

Table 3: Calculation of profit

2 Analysis

2.1 Use-Case Model

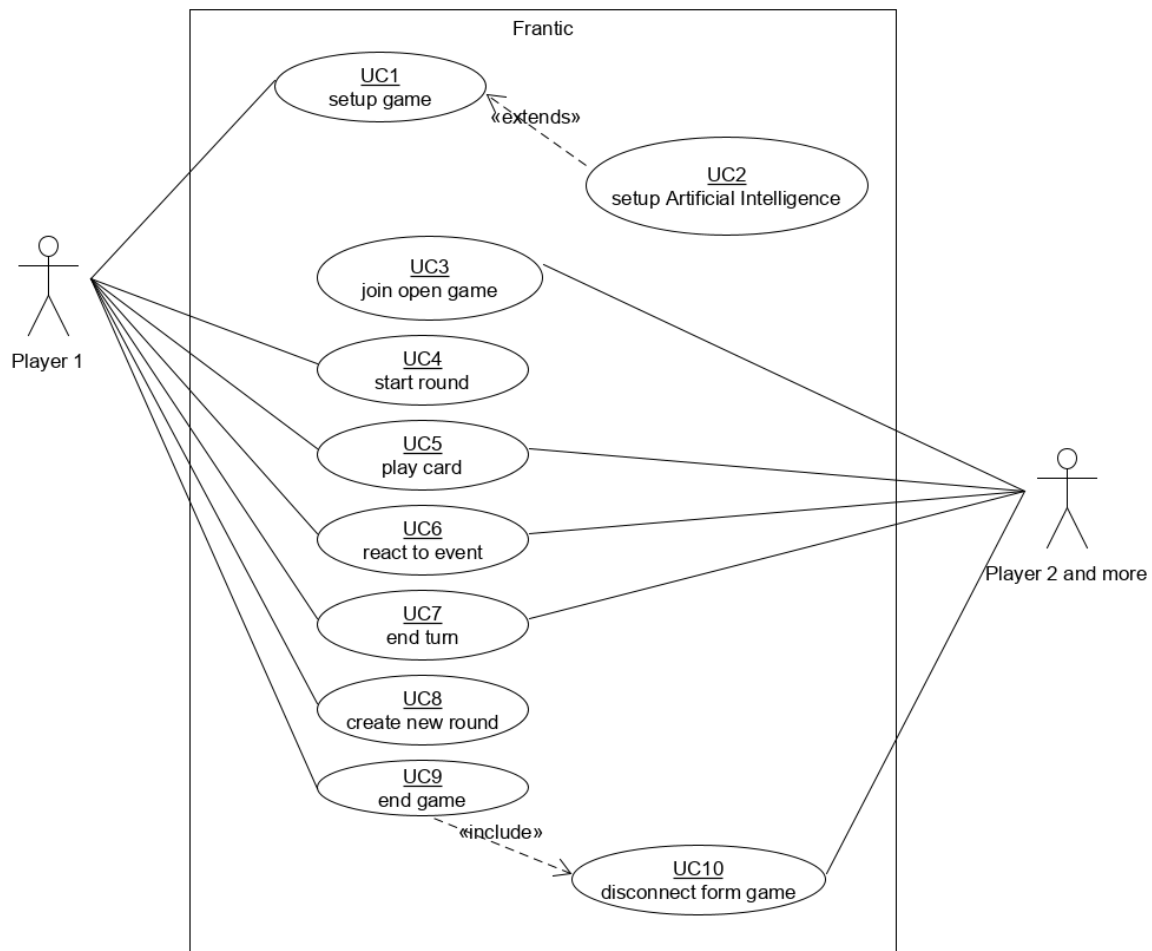


Figure 1: Use-Case-Model of Frantic

UC1	setup game
Scope	This use-case describes the game setup, which the first player undertakes to create a singleplayer game or multiplayer game. In a singleplayer game, the player can choose the amount of Artificial Intelligence Competitors (UC2). The multiplayer mode will create a server connection for other players to connect to and play (UC3).
Level	User-function that initiates the game for a player or many players to play.
Primary Actor	Player 1

Stakeholders and Interests	The Player wants a smooth and easy user-interface to start a game, without any difficulties.
Preconditions	Game executable has to be started in order to be able to set up a game.
Success Guarantee / Postconditions	The Game can be started and played, either alone against AI or with friends. (UC3 – UC10)
Main Success Scenario / Basic Flow	<p>Player starts a new game.</p> <p>System presents the player with three options: Single Player, Multiplayer, Join a Game.</p> <p>Player chooses Single Player.</p> <p>System presents the player with the settings: number of AI Players (UC2), end of game condition.</p> <p>Player chooses a number of AI players between 2 and 6 and the end of game condition as a number of points between 113 and 179.</p> <p>Player confirms the choice.</p> <p>System starts a game according to the player's choice.</p>
Extensions / Alternative Flows	<p>*a. At any time, Player closes the window or System fails: (System logs after each completed move)</p> <ol style="list-style-type: none"> 1. Player restarts the game, requests recovery of prior state. 2. System reconstructs prior state. <p>3a. Player chooses Multiplayer:</p> <ol style="list-style-type: none"> 1. System presents the player with the settings: number of other players that can join, end of game conditions. 2. Player chooses a number of players between 2 and 6 and the end of game condition as a number of points between 113 and 179. 3. Player confirms the choice. 4. System starts a game according to the player's choice and opens a connection for other players to join the game. <p>3 b. Player chooses Join a Game: see UC3</p>
Special Requirements	The setup has to be easy, readable and understandable in basic languages. Depending on success, the game has to be translated into different languages and made available.

Technology and Data Variations List	None
Frequency of Occurrence	Every time a game has to be created; this use-case will be called up.
Open Issues	None

Table 4: Fully dressed Use-Case description for set game

UC2	setup Artificial Intelligence
Description	This use-case describes the setup of Artificial Intelligence for the Singleplayer mode of the game. When the user chooses the number of enemy AIs and starts the game. The number of AIs will be represented in the game itself with AI players.
UC3	join open game
Description	This use-case describes the third option the player encounters when starting the game. A player looking to join a Multiplayer game already created by his or her friend.
UC4	start round
Description	This use-case describes the starting of a round by the host player. The system has to create decks and hands for the player and load it to all the participants simultaneously. In a Singleplayer game, the player and the AI will receive information of hand cards and enemy statuses to work with. The same thing happens with a Multiplayer game.
UC5	play card
Description	This use-case describes the player playing a card from his or her hand. This use-case includes the display of handcards, selection and the playing of a card. It marks an important step in the game.
UC6	react to event
Description	This use-case describes the reaction to an event by either the player or an enemy player. This event could be drawing a card, reacting to a special card, interdicting with a special card or activating a special event card.
UC7	next turn
Description	This use-case describes the end of a player's turn and the transition to the next player's turn.
UC8	create new round
Description	This use-case describes the creation of a new round when the previous round has been ended. If a player has played all his cards unto the discard pile or the draw deck is empty the round

	is completed. As long as the victory conditions are not met, a new round will be started.
UC9	end game
Description	This use-case describes the games end. This occurs when the victory/defeat conditions are met. In this case, a player reached the maximum amount of points.
UC10	disconnect form game
Description	This use-case describes the disconnecting of the participating players from the host player's game.

Table 5: Use-Case description for UC2 to UC10

2.2 Supplementary Specification

2.2.1 Functionality

- All moves and errors are logged.
- Illegal moves should be detected and prevented.
- In multiplayer mode the program should create a virtual room with all participants.
- A multiplayer game must comply with industry security standards to prevent data manipulation and unauthorized access.

2.2.2 Usability

- The program must be executable on the current computer systems (MacOS, Windows).
- The player should be able to enter a game within 30 seconds of starting the application.

2.2.3 Reliability

- It must be possible to interrupt the game at any time without losing any data.
- If the connection to the other players is lost, the program must try to reconnect.

2.2.4 Performance

- Creating a connection to a virtual room may take a maximum of five seconds.
- The move of a computer-controlled opponent must be calculated within 0.5 seconds.
- The transfer of a player's move to the other players may take a maximum of one second and must be successful in 99.9% of cases.

2.2.5 Supportability

- There is a chat function that can be used by all players in the same virtual room.

2.2.6 Implementation Constraints

- The use of Java technology is already stipulated.























2.2.7 Free Open Source Components





















- Free and open source Java components should be used in building the game. That makes JavaFX a strong contender for building the GUI.

2.2.8 Legal Issues


- The game's implementation was greenlighted by the original creators. It must be planned for possible licensing restrictions.


2.2.9 Additional Card Rules


-  **2nd Chance:** The player has to play another card on top of 2nd Chance, either color on color or colorless Special Cards. If a player cannot play a card from his or her hand, he has to draw one from the deck.
-  **Color Swap:** Can be played on two different colors and swaps to the other color.
-  **Exchange:** The player of this card gives another player two cards of his choice from his hand and in exchange has to blindly draw two cards from his chosen opponent.
-  **Gift:** The player of this card gives two cards from his hand to another player.
-  **Skip:** The player of this cards chooses a fellow player, who is suspended for one turn.
-  **Thief:** The player of this card has to look into the cards of another player and steal two of his/her hand cards.
-  **Troublemaker:** Activates a special event card.
-  **Fantastic:** This card can be played on any card. You can choose a number or a color.
-  **Fantastic Four:** The player of this card chooses a person, who in return has to draw four cards from the deck. It is also possible to determine multiple players and divide the four cards between them.
-  **Counterattack:** As soon as a Special Card is played against a player, this card can be thrown in instantly by the victim. The effect is cancelled, and the player of this card can redirect it to another player. The target of the freshly obtained effect is freely choosable.
-  **Equality:** The player of this card chooses a fellow player who holds fewer cards than himself in his hands. This player has to draw as many cards until their number of cards is equal.
-  **Inequality:** The player of this card can choose a victim, which has to draw as many cards from him/her until the victim holds more cards in its hand.
-  **Lucky Bastard:** This card can be thrown in right before a special event is activated. The player of this card is then excluded from the event and its effect is not applied to him/her.
-  **Nice Try:** As soon as a player got rid of all his hand cards and therefore ends the current round, this card can be thrown in immediately. The player who ended the round then needs to draw three cards and the game round continues.
Important: If multiple players finish off their cards from an event and Nice Try is played, every player who finished has to draw three cards.
-  **Special Favours:** The player of this cards can exchange all special cards with another player.
-  **Curse:** This card cannot be played or discarded. But it can be passed on or exchanged at any given opportunity. If a player holds this card at the end of a round, it counts as 13 points.
-  **Fuck You:** You can only dismiss this card, when you have exactly then cards in your hand, including the Fuck You cards itself. The round continues with the card played before Fuck You.
Important: This card can only be blindly obtained by an opponent and not be willingly given.
Exception: During Special Event cards it can be thrown away or passed on.
-  **Mimicry:** When being played, this card can become any existing Special Cards.
Important: If this card imitates a colored special card, it can only be played on said color.
-  **Black Hole:** All Black Cards in the hands of the players go to the player, who activated the Black Hole event.
-  **Capitalism:** He that has plenty of goods shall have more. All players double their hand cards.
-  **Charity:** Every player has to pick one card from the player with the most hand cards.
Important: If two or more player have equal amount of most cards, cards are picked from all of them. These players don't have to draw cards from each other.
-  **Communism:** Everyone has to draw as many cards to equal the player, who holds the most cards in his hands.

-  **Crowdfunding:** Every player has to give one card to the player with the least hand cards.
Important: If two or more players have an equal amount of least cards, cards are given to all of them. There players don't have to give cards to each other.
-  **Distributor:** All players give their hand cards in a small deck face down to the player who played the Black Card. This player can look at each small deck and distribute them between the players, without shuffling them. The player who played the Black Card has to distribute his/ her hand cards as well and keep a deck form another player.
-  **Doomsday:** The game round is immediately over. Every player receives 50 points.
-  **Double Taxation:** The player who holds the least points with the three highest cards has to draw one card, the player with the second least points has to draw 2 and so on. In case of equal score, the affected players have to draw the equal amount of cards.
-  **Earthquake:** Every player gives his cards to the player to his right.
-  **Event Manager:** The next three Event Cards from the deck are flipped over. The player of the Black Card then chooses one of these events which will be executed.
-  **Expansion:** The players have to draw cards from the deck accordingly: The 1st player draws one card, the 2nd draws two, the 3rd three and so on.
-  **Finish Line:** The game round is immediately over and the players count their points according to their hand cards.
-  **Friday The 13th:** Nothing happens.
-  **Gambling Man:** Every player has to place a preferably low Numeral Card of the last played color face down. All cards are simultaneously turned around. The player with the highest digit has to take the other cards in. Players without Numeral Cards of said color have to draw two cards as penalty.
Important: If no color has been played so far, the event is ineffective.
-  **Identity Theft:** The player with the lowest amount of points swaps the score with the player who holds the most points.
If the points are not written down: The player of the Black Card chooses two players (also themselves possible) who swap their place at the table, and with it their cards.
-  **Last Chance:** Every time a player has discarded the last card an event is activated as a last resort.
-  **Market:** As many cards as there are players, the top cards from the deck are turned face up in front of the players. The players then pick in turn one card to take in their hands.
-  **Mating Season:** Every combination that can be achieved with Numeral Cards have to be disposed of. Combinations would be pairs, three of a kind, four of a kind and so on. The color of the cards doesn't matter.
-  **Merry Christmas:** Every player has to give all of their hand cards to other players. They can divide them as they please.
-  **Mexican standoff:** All players dispose of their cards and draw in turn three new cards from the deck.
-  **Plague:** There are immediately two additional event cards executed, one after the other. Even if the first event would end the current round or a player has discarded all cards, the second event will be executed as well as possible.
-  **Plus One:** New rule for the rest of the round: Every time the players have to draw cards from the card deck, they have to draw one more.
-  **Recession:** The players have to dispose of cards from their hands accordingly: The 1st player has to dispose of one card, the 2nd disposes of two, and the 3rd of three and so on.
-  **Repeat:** The player of the Black Card has to execute an event that has already been uncovered.


Important: If no other event has been uncovered so far, the event is ineffective.


 **Robin Hood:** The player with the smallest amount of hand cards swaps his cards with the player who holds the most.


 **Russian Roulette:** All players put one card from their hand together. These cards get shuffled and put on top of the card deck and the game continues.


 **Seppuku:** The players decide in turn if they want to commit Seppuku. If they risk it, they have to draw the top card from the card deck. If it is a numeral card they get credited 21 points, if it's any other card they get an additional 42 points. If they decide not to commit Seppuku then they get 21 additional points.


If the points are not written down: If a numeral card is drawn they can dispose of two cards from their hand, if it's any other card they have to draw four cards from the deck. If they decide not to commit Seppuku, then they have to draw two cards from the deck.


 **Surprise Party:** Every player must give one of their cards to a player of their choosing.


 **The All-Seeing Eye:** Every player has to show their cards. The cards are exposed until every player gives his OK to continue.

 **Third Time Lucky:** Every player has to draw three cards.

 **Time Bomb:** Every player has only three turns left. The round ends when a player would reach his fourth turn. If a player can dispose of all his cards before the fourth turn, he gets credited ten points. The other players get a penalty of ten points. If no one is able to diffuse the bomb, the round is over and the points in this round get doubled.

 **Tornado:** The hand cards of all players are put together, shuffled and one at a time newly distributed by the player of the Black Card.

 **Trust Fall:** Every player has to choose in turn a fellow player who has to give him/her two cards from his hand.

 **Vandalism:** Every player has to dispose of every card (Numeral and Special Cards) of the last played color.

Important: If no color has been played so far, the event is ineffective.

2.3 Domain Model

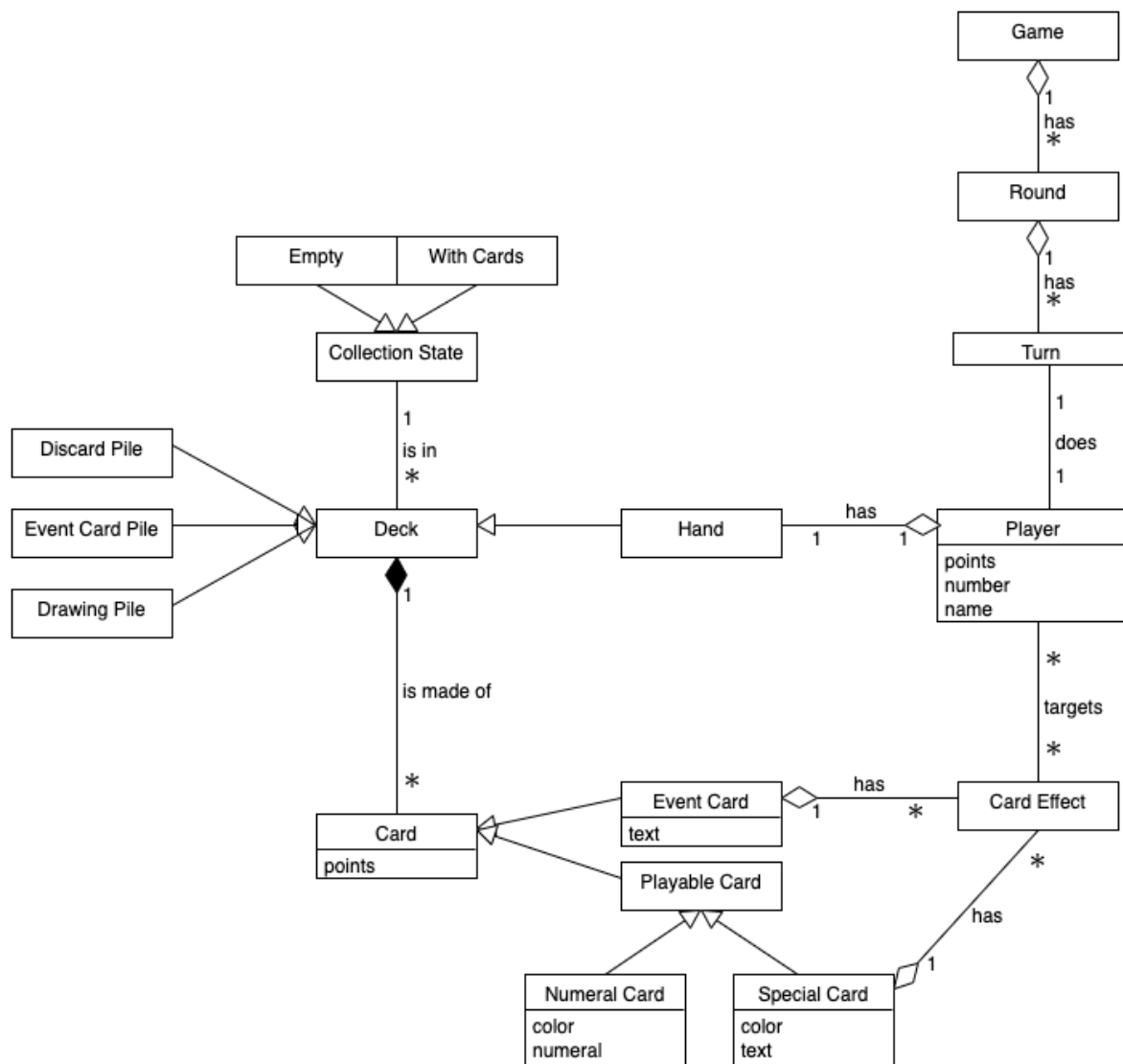


Figure 2: Domain model of the card game Frantic

The Domain Model represents the first step in the planning and designing of game functionalities. This process displays the first thought flow made by the team converting a card game into a program. The Domain Model includes the basic principles found in a card game, players and cards. Obviously, there can't only be players and cards, because those terms cover a broad spectrum.

For example, a card is specified as a playable card and an event card. Then there are two subcategories of playable cards, special cards and numeral cards. Further details of cards would have to be specified in the game rules and further requirements and don't really impact this model. Another important factor in the model is the player.

The player plays a key role in a card game. A player has a hand with cards in it, plays his or her given turn and keeps the game going many rounds until it is finished. Now comes the major factor to connect the player to cards and that is the deck. The player will have to draw cards from the deck to be able to hold hand cards. In addition, the deck would be divided in three categories. A drawing pile, from

which the player can draw his or her cards from, a discard pile unto which the player lays down his or her hand cards and an event cards pile.

The event card pile is mostly called up when a player activates an event through playing special cards. These cards do not mix with the normal game play, but they effect it for the better or worse.

3 Design

3.1 Package diagram

There are three packages contained in the project. UI, Game Logic and low-level infrastructure. The UI packet contains the Game and Menu Interface as well as all the Controller and View classes necessary to build the GUI, which are contained in a subpacket AppFrame. The game logic packet has all classes necessary to run the UI packet such as the interfaces Card and EventCard as well as the classes HandCard, SpecialCard and all the different event cards classes necessary to create all the types of cards which are stored in the Deck and EventDeck class, as well as Player, ComputerPlayer and Game classes positioned in the subpacket Game Rules. The communication between the UI and the game logic is made possible thanks to the other subpacket in Game Logic called Application layer request. The game logic packet also communicates with the low-level infrastructure packet, which contains the logging and networking.

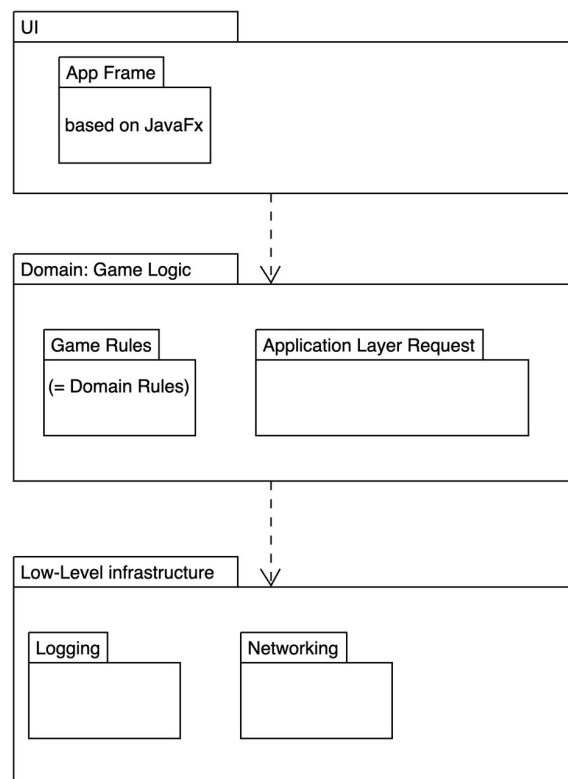


Figure 3: Package diagram of the card game Frantic

3.2 State Machine diagram

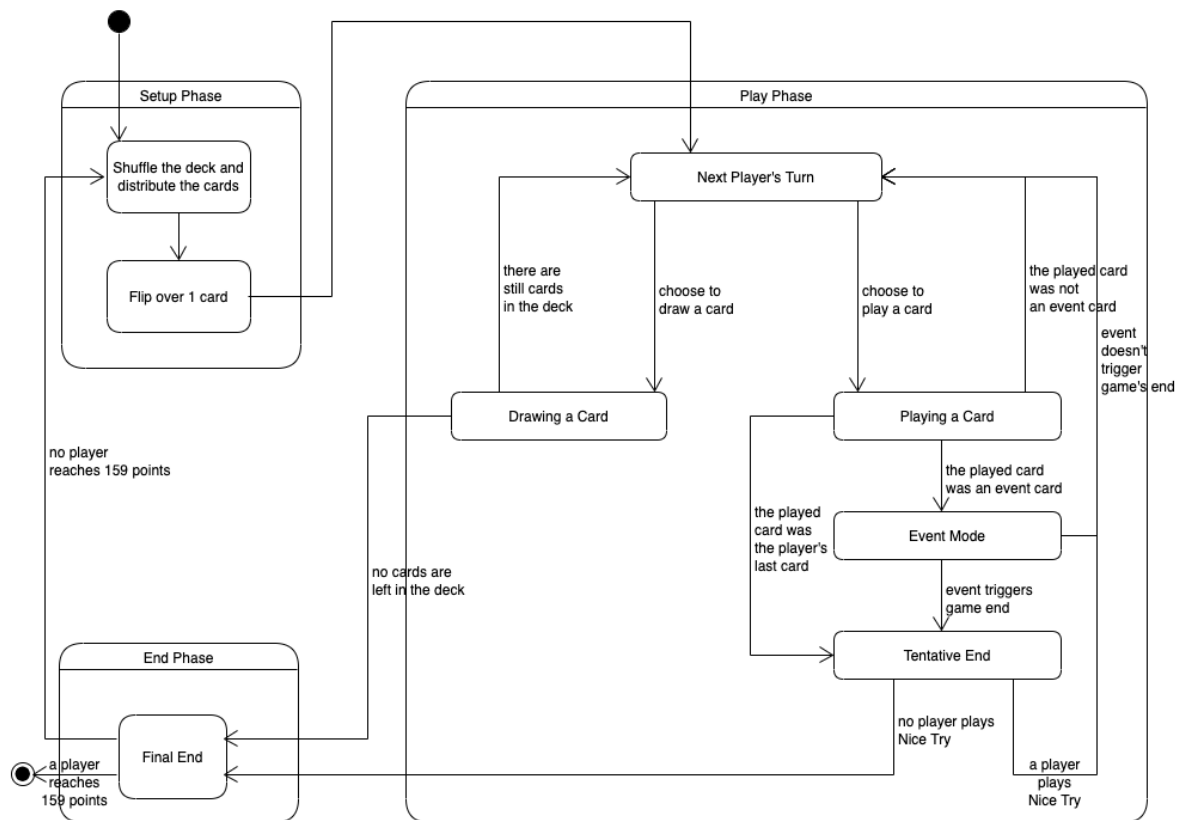


Figure 4: State machine diagram of the card game Frantic

The state diagram represents the general course of the game. It must be said that special functionalities of event and special cards are excluded here. There are three composite state, the first is Setup Phase, the second is Play Phase and the last one is End Phase. The main part is all covered in Play Phase. The inner cycle is based on AI or player decisions. At first weather drawing a card or playing a card. Playing a card may activate an event. After a player finished his turn, the next one is in line. This loop ends if one player has played his last card and nobody plays Nice-Try card. Then a new round is started, and the Setup Phase occurs again. This bigger cycle repeats itself until one player reached over 159 points.

3.3 Interaction Diagram

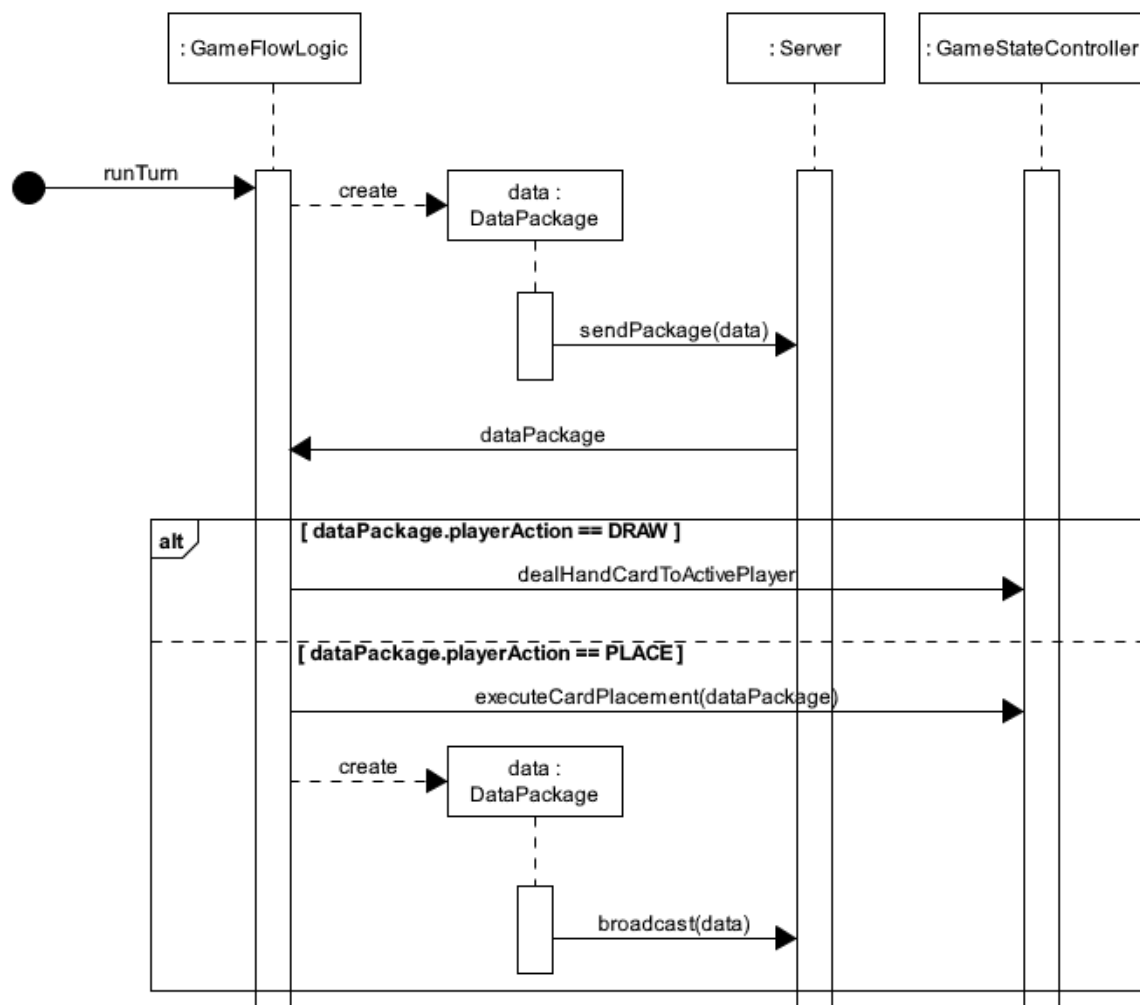


Figure 5: Interaction Diagram of placing a card

What happens when placing a card?

The central element of the design is the **DataPackage** class that encapsulates precisely the current game state. The **GameFlowLogic** class handles the game structure, the **Server** handles sending messages to all player clients, and the **GameStateController** handles the actions in the game.

To reach a state where a player may place a card, the **GameFlowLogic** creates (through a Factory) a new **DataPackage** object. This object is passed as a message to the **Server** and the **Server** transmits it to all clients and waits for an answer, namely for the player whose turn it is to do a game action like drawing a card or placing a card. Once a player makes a decision, the **Server** sends this decision (which is just a **DataPackage** object with the **PlayerAction** set to **PLACE**) to the **GameFlowLogic**. Then, the **GameFlowLogic** sends to the **GameStateController** the corresponding message to trigger a game action: a card draw or a card placement. In the case of a card placement, a new **DataPackage** object is created, which represents the new game state with the placed card, and a message is sent to the **Server** to broadcast this new **DataPackage** object to all clients such that they have an update to see which card was placed on the card pile.

3.4 System Sequence Diagram

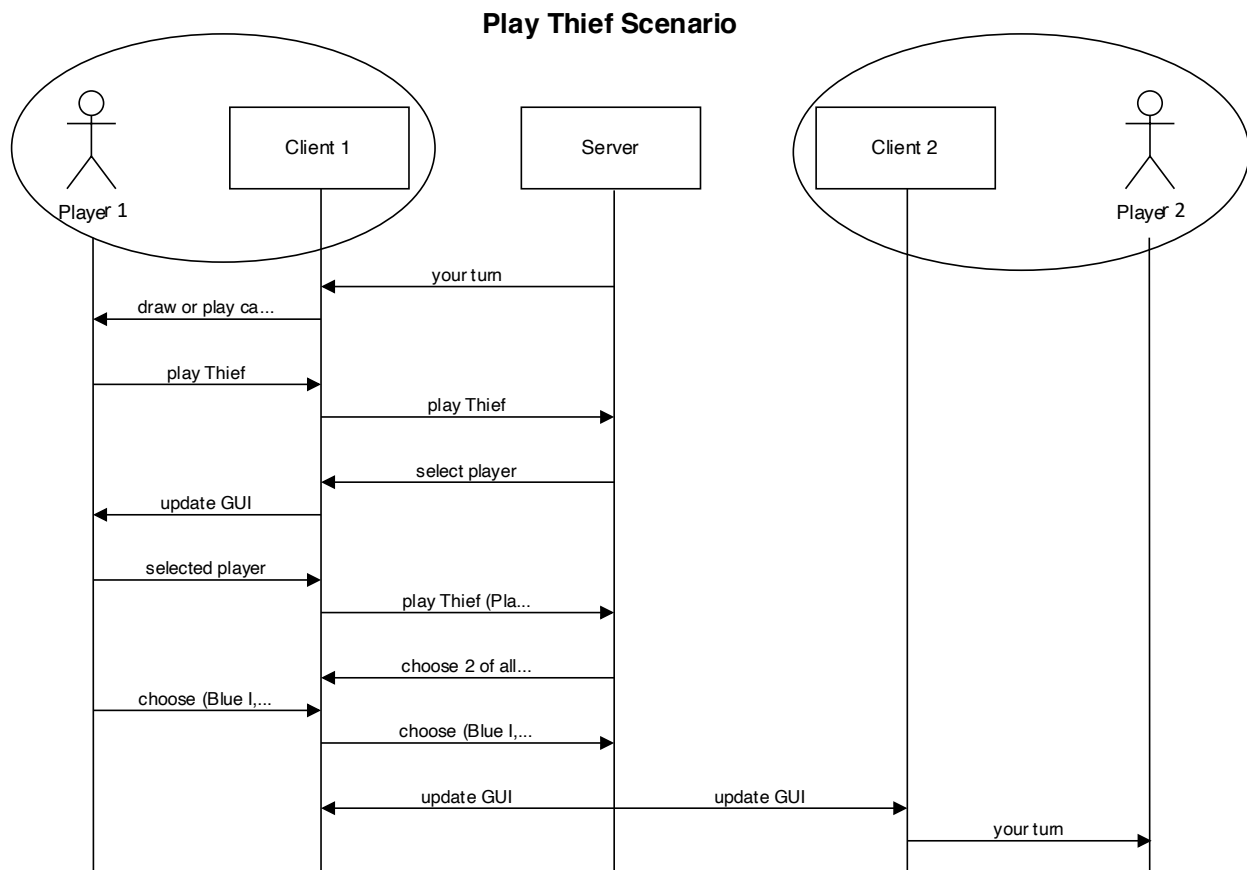


Figure 6: System Sequence Diagram, Play Thief Scenario of the card game Frantic

There are two players (player 1 and 2) who are both connected to the server as a client.

This scenario is simulating Player 1 playing the card Thief against Player 2.

1. Server sets the current Player to Player 1 and sends this information to Client 1 with a DataPackage.
2. Player 1 has to choose between playing a card or drawing one.
3. Player 1 chooses to play Thief.
4. Client 1 tells the Server, that Thief has been played with a DataPackage.
5. Client 1 has to ask the player for the target and the GUI is updated.
6. Player 1 chooses Player 2 as target.
7. Client 1 sends this information to the server.
8. Server sends all handcards of Player 2 to Client 1.
9. Client 1 shows the Cards to Player 1, who has to select two cards.
10. Blue 1 and Red 3 is chosen.
11. This information is passed to the Server.
12. The GUI of all players is updated, they both see the current gameboard.
13. Server sets the current Player to Player 2 and sends this information to Client 2.

3.5 GUI

After starting the Frantic game, you will be presented with the menu view of the game. Here you can choose which type of game you want to play. You can either Play against computers with the "Singleplayer" mode or invite friends to play using the "Multiplayer" mode. If a friend invited you to play, choose "Join game".

In the game creation view, you can specify your own name, choose the party size and the general game duration. Once everyone is ready you can start the game.

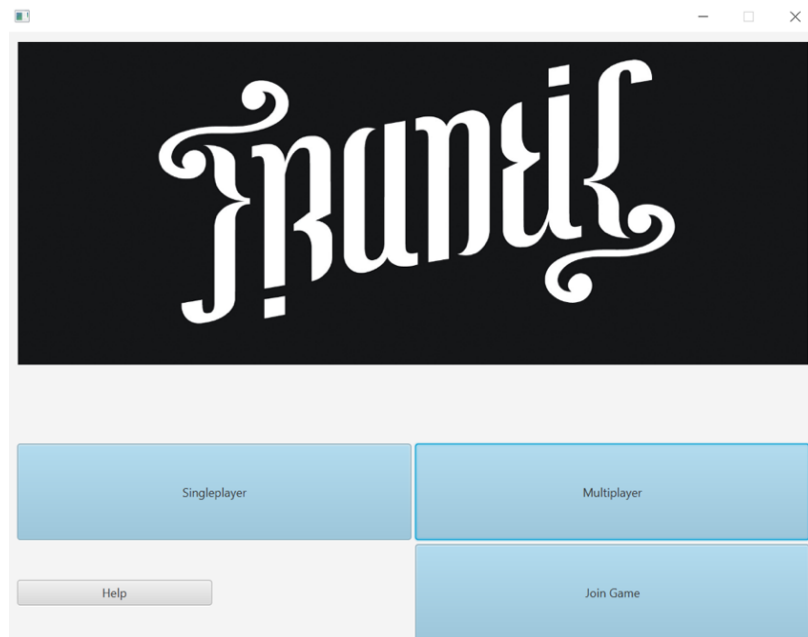


Figure 7: Screenshot of the main menu

In the join game view, you are able to join a friend's game by specifying the secret code and your name.

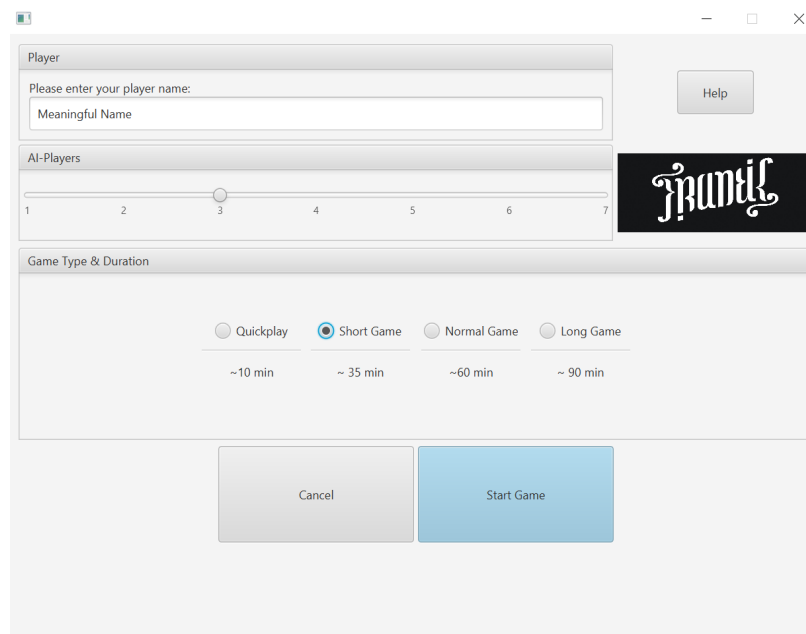


Figure 8: Screenshot of single player setup

Once the game started, the game view is shown with all participating players at the very top. The Player highlighted red is currently playing. Underneath the players are the Event Cards pile and the Discard pile showing only the cards on the very top of the pile. Next to the two piles are your action buttons. Once it is your turn you can choose to draw a card with the "Draw Card" button. When you are done with your turn, do not forget to finish your turn by pressing the "Finish Turn" button. A disabled button signals that the action of the button

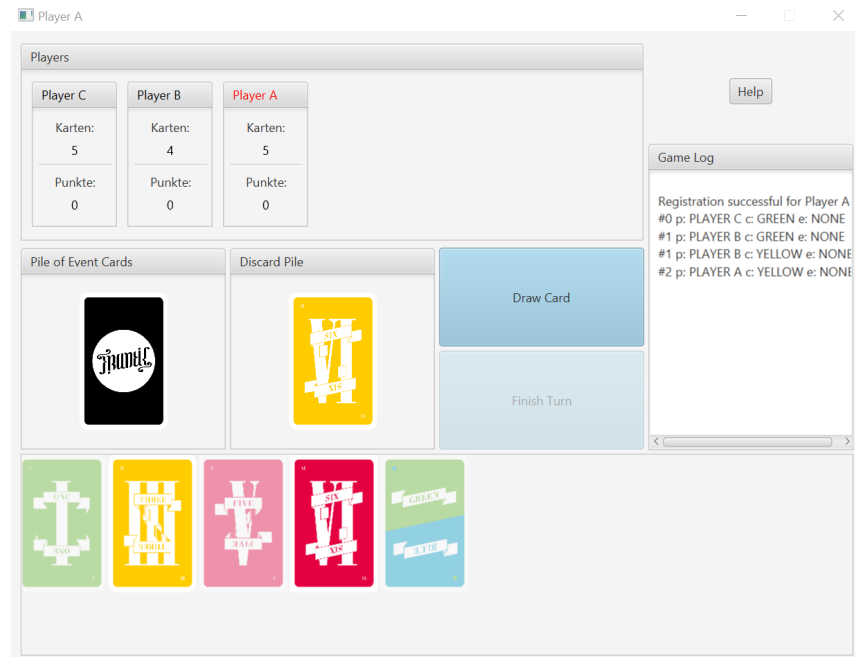


Figure 9: Screenshot of in game board

is currently not allowed. Next to the two buttons you find the game log where you will find information about all the actions of all the players in the game.

At the very bottom you can see your own cards that you are holding in your "hand". Cards that you can't play at the moment are grayed out. Note that all cards are grayed out if it is not your turn. If you want to play a card you can simply double-click the card.

3.6 DCD

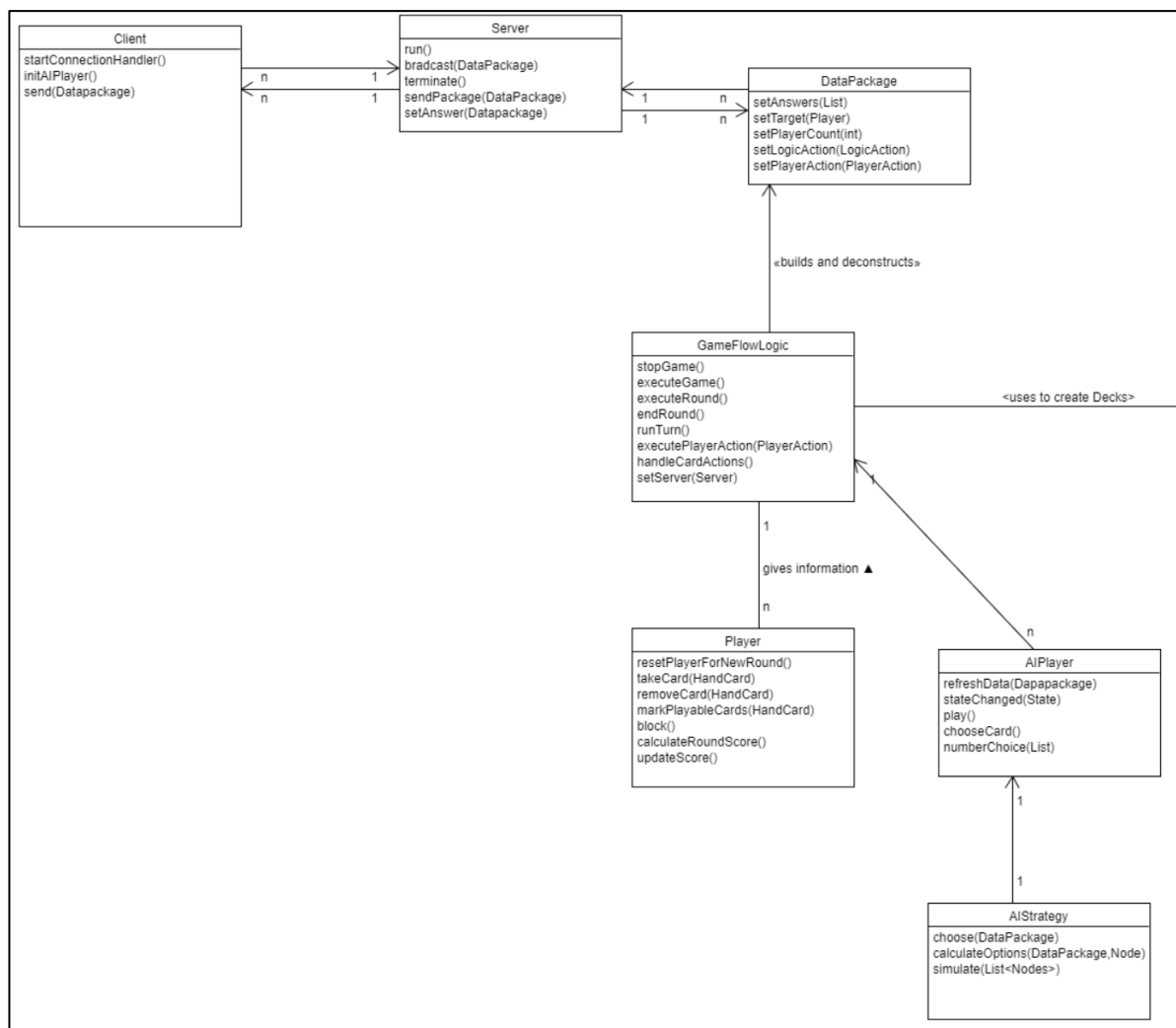


Figure 10: DCD of the card game Frantic, part 1

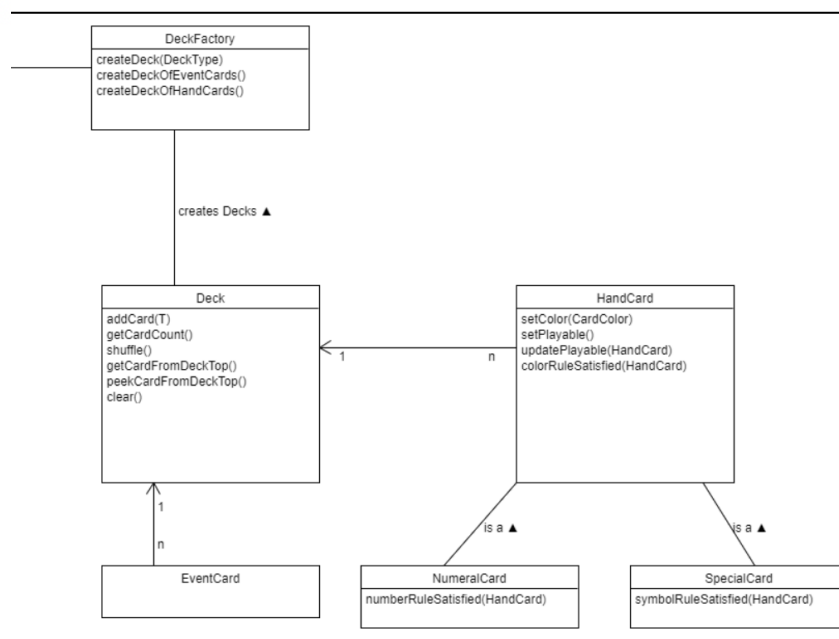


Figure 11: DCD of the card game Frantic, part 2

3.6.1 AI

The Frantic game client has a multiplayer mode and a single player mode. The single player mode requires that the computer player makes the decisions that are expected from a player. This includes:

- Knowing when it is its turn
- Playing a card
- Choosing colors and numbers
- Targeting other players
- Drawing a card
- Ending its turn

For the implementation, there are two strategies. A simple and a sophisticated AI player.

3.6.1.1 Simple AI Player

The simple AI Player reads from the broadcast data package whether it is its turn and which actions it can take. Then it calculates its possible moves. Then it performs a valid random move and sends in return a data package with this move.

3.6.1.2 Sophisticated AI Player

In many games where it is hard to decide on which policy to base the actions of a computer player on, experience has shown that a Monte Carlo tree search performs very well. [4] This is also the general strategy that DeepMind Technologies [5] used when they built their computer program that can beat humans in the game of go and it is also the general strategy that is used in Total War: Rome [6] to steer the AI armies. The strategy is described in Artificial Intelligence: A Modern Approach by Russel and Norvig (2020). [4]

The game tree is calculated from the data package as the data package captures precisely a moment in the game and all possible decisions by all players can be computed from it. From the possible nodes of the game tree the most promising one is chosen. Research has shown that this node can be selected based on an "upper confidence bounds applied to trees" where it can just use the established formula.

This node is expanded by doing one simulation to the game end and it records if its player has won or lost. Then it backpropagates this result and updates all the nodes with the number of wins and number of playouts. It keeps doing this until some time runs out and, in the end, it has the most promising node that has the most promising action for its AI player to win the game. Unfortunately, the implementation couldn't be finished in time for the beta release. So currently only the simple AI is fully working.

3.6.2 MVC

The Project is designed after the MVC Pattern. The MVC Pattern dictates that, a program is divided in 3 different parts: Model, View and Controller. The Model is the part, where all the logical work happens for the game. The View is the display for the player, it shows the UI (User interface). The Controller handles all the functions for the View and connects the View with the Model.

All Views hold the display for the Users and are connected with their own FXML Window. Every Window has a Controller that handles all the buttons and clickable functions. The View starts up a Server/Client which in turn starts the game logic in the model.

3.6.3 Design patterns

The project made use of some design patterns. Here are the most important ones:

3.6.3.1 Simple Factory Pattern

There is a Datapackage used for the Server-Client communication to define the communication. The Simple Factory Pattern is used to build this package with a PackageFactory. This pattern also comes in handy for building the decks and the cards with DeckFactory and CardFactory.

3.6.3.2 Singleton Pattern

The Singleton Pattern is an important part in the project. There can only be one deck for the game and the Players have to be saved in one place for the connection over the server to work. This means there has to be a class to save all this information. This is handled by the GameState and the GameStateController.

3.6.3.3 Dependency Injection

This Pattern is used for the Mock and the Unit Tests.

3.6.3.4 Observer Pattern

The Observer Pattern is needed as Listeners for communication between the GUI and the model. For example, when a button is clicked, the action is called over an eventListener. This is necessary to implement a correct MVC Pattern

3.6.3.5 Strategy Pattern

The Computer has different levels of intelligence. The algorithm for choosing the next card to play (chooseCard) is implemented in the Strategy Pattern with the Interface ComputerStrategy.

3.6.3.6 Visitor Pattern

The GameStateController is implemented as visitable. All classes that implement the Active Interface are Visitors. The Visitors are passed through gameState.getCardOnTopOfDiscardPile. This is an easy way to implement a lot of different cards with different actions and all can be called at the same place.

3.6.4 GRASP

The Information Expert Pattern is used for the Game class to handle the responsibilities. This helps with keeping a low coupling and high cohesion.

The dataPackage led to a problem with building objects. The Creator Pattern defines the responsible class PackageFactory to build these objects.

For an UI based application the Controller pattern is a must and is implemented to comply with the MVC Pattern. This also includes the Indirection Pattern.

Polymorphism is very important in this project. All Cards are implemented with the Polymorphism Pattern to make it easily expandable and keep the coupling low.

4 Implementation

4.1 Testing

4.1.1 JUnitTests

The JUnitTests are to test the game functionalities and logic. JUnitTests are chosen to test the working code and make sure that if changes happen or wrong values are generated, they can be caught.

4.1.1.1 AIPlayerTest

The AI tests cover the AI behavior of different player inputs and game states that currently are in the game.

Tests	
AI Turn	
Not AI Turn	
Color Choice	
Negative Color Choice	
Number Choice	
Negative Number Choice	
Number Or Color Choice	
Negative Number Or Color Choice	
Player Choice	
Negative Player Choice	
Card Choice	

Table 6: List of tested AI player

4.1.1.2 EventCardTest

- Black Hole: Tests the black hole event card, which gives all the black cards on the player's hands to the one that activated the event.
- Capitalism: Tests the capitalism event card, which doubles every player's hand cards.
- Communism: Tests the communism event card, which forces all players to draw cards to equal the one player with the most cards.
- Earthquake: Tests the earthquake event card, which makes all players give their hand to the player on their right.
- FridayThe13th: Tests the Friday the 13th event card, nothing happens with this event.
- Third Time Lucky: Tests the third time lucky event card, which makes all players draw 3 cards.

Tests	
Black Hole	
Capitalism	
Communism	
Earthquake	
FridayThe13th	
Third Time Lucky	

Table 7: List of tested event cards

4.1.1.3 SpecialCardTest

- ColorSwap: Tests the colorswap special card, which allows the player to swap 2 given colors on the card, i.e. blue-yellow, green-blue, green-yellow.
- Skip: Tests the skip special card, which allows a player to skip another player.
- Thief: Tests the thief special card, which allows a player to steal 2 cards from another player.

Tests	
ColorSwap	
Skip	
Thief	

Table 8: List of tested special cards

4.1.1.4 HandCardTest

The hand card tests test the basic functionalities of cards. For example, if a green number card can be played on another green card, a green number card played on a blue number card.

All tests are based on color and number for all different cards.

Tests	
Play "Blue 5" on "Blue 5"	
Play "Blue 5" on "Blue 3"	
Play "Green 3" on "Yellow 3"	
Countertest, play "Green 5" on "Blue 4"	
Play "Blue Skip" on "Blue 4"	
Play "Blue 4" on "Blue Skip"	
Play "Black 2" on "Red 2" & Vice-Versa	
Test if "Skip" skips player	
Test if "Thief" steals from player	
Play "Green-Blue" colourswap on "Green 3"	
Play "Green-Blue" colourswap on "Blue 7"	
Countertest "Yellow-Red" colourswap on "Blue 7"	
Play "Blue 5" on "Green-Blue" colourswap	
Play "Blue Thief" on "Blue 5"	
Countertest "Blue Thief" on "Green 3"	

Table 9: List of tested hand cards

4.1.1.5 NumeralCardTest

The numeral card tests test the card comparison function, to see that the cards are compared properly with the correct return values.

Tests	
Equals Same Class, Different ID	
Equals Same Class, Same ID, Same Number, Same Color	
Equals Same Class, Same ID, Different Color	
Equals Different Types	

Table 10: List of tested numeral cards

4.1.1.6 SpecialCardComparisonTest

The special card tests test special card comparisons, to see if the cards are equal to another card with their values and functions.

Tests	
Symbol Rule Satisfied	
Symbol Rule Satisfied, Same Class	
Symbol Rule Satisfied, Different Class	
Symbol Rule Satisfied, Different Class, Same Color	

Table 11: List of tested special cards

4.1.1.7 GameStateControllerTest

These tests test the state change of the game. The outcome of a new card on top of the discard pile and the switch to the next players are tested.

Tests	
Create Players	
Evaluate Winner and Looser of Round	
Place Card on Top of Discard pile (Numeral)	
Place Card on Top of Discard pile (Black)	
Place Card on Top of Discard pile (Action Card)	
Switch to next Active Player	
Deal Hand card from empty Deck (End Round)	
Execute Card placement, causes Player with no Cards (End Round)	

Table 12: List of tested GameStateController

4.1.2 Integration test

The Integration test is a combination of the JUnitest functionalities. It tests a full game scenario to see if all functions work hand in hand together and assure a smooth gameplay for the users. This test is very difficult to automate which was a planned feature. Yet the complexity of the Integration test makes it very hard for the testing team. At the current state it will be tested by hand.

To test the Integration test, a game has to be created with 3 players with either 3 cards or the usual 7. For testing sake, the players are called p1, p2 and p3.

P1 creates a new multiplayer game with 2 players and a with the quickgame option. P1 waits in the lobby for p2 and p3 to join. As soon the other players joined the game starts. Every player holds 3 cards on the hand. The first uncovered card from the pile is placed, a "Blue 3".

P1 start with laying a "Blue 7" on the current card laying on the pile.

P2 plays a "Green 7" on the last played card.

P3 plays a "Green Skip" card and chooses to skip p2.

P1 draws a card, because p1 does not have a green colored card and ends the turn.

P2 is skipped.

P3 draws a card and plays the drawn card, "Green 2" onto the pile.

P1 plays "Red 2" onto the pile.

P2 plays "Colorswap Red-Yellow" onto the pile.

P3 plays "Red 9".

P1 plays "Black 9"

System says "Earth Quake" occurs

P1 gives hand to P2, P2 gives hand to P3, P3 gives hand to P1
 P2 draws a card, ends turn
 P3 draws a card, ends turn
 P1 draws a card, ends turn
 P2 draws a card, ends turn
 P3 draws a card and plays "Green 9"
 P1 plays "Thief Green" and steals 2 cards of P2's hand
 P2 play last card "Green 8"
 Game Ends with P2 as Winner.

4.1.3 System test

The System test tests the game functionalities from a Use-Case point of View. They represent the game parts for Singleplayer, Multiplayer and Game-Flow. It tests the Singleplayer- and Multiplayer- setups for a game. As well the Game-Flow which test the bigger game loops functions.

Singleplayer	
Create Singleplayer game	
Player is able to play	
AI-player play cards	
Multiplayer	
Create Multiplayer game	
Other players able to join game	
Game can be started	
Player is able to play	
Joined players can play	
Game-Flow	
Round Start	
Card Play	
Event reaction	
End turn	
Next player with the same loop	
Winning condition	
Disconnection to the game	

Table 13: List of system tests

- **Create Singleplayer game:** Tests the creation of a singleplayer game with all its functions and values. The goal is to have a running game with the correct number of players.
- **Player is able to play:** Tests if the player can play his turn throughout the game.
- **AI-player play cards:** Tests if the AI-players in the game play their turn without complications.
- **Create Multiplayer game:** Tests the creation of a multiplayer game with all its functions and values. The goal is to have a code for the joining players and a game with the joined players.
- **Other players able to join game:** Tests the functionality of other players joining the host player's game.
- **Game can be started:** Tests the setup of the game, which means that every player can see hand cards and other player information.
- **Player is able to play:** Tests if the player can play his turn throughout the game.
- **Joined players can play:** Tests if the other players that joined the game can play and see hand cards and other player information.

- **Round Start:** Tests starting of a round.
- **Card Play:** Tests the playing of cards for every player and the check that only one card can be played at a given time.
- **Event reaction:** Tests the informing of player of an event and its function in the game.
- **End turn:** Tests the turn end of every player's turn.
- **Next player with the same loop:** Tests the game loop that every players shares the same functionalities.
- **Winning condition:** Tests the winning condition of the game, if a players empties all hand cards.
- **Disconnection to the game:** Tests the game if a player decides to leave/close the game.

4.1.4 GUI test

The GUI-Tests show if all the GUI functionalities work as planned. It tests all the Interface blocks, buttons, text fields and sliders if they work.

GUI-Tests	Description	Status
Main Menu	Singleplayer Button	
	Multplayer Button	
	Join Game Button	
	Help Button	
Singleplayer	Name Textfield	
	Players Slider	
	Game Duration Radiobuttons	
	Cancel Button	
	Start Game Button	
	Help Button	
Multplayer	Name Textfield	
	Players Slider	
	Game Duration Radiobuttons	
	Code Field	
	Log Field	
	Cancel Button	
	Open Lobby Button	
	Start Game Button	
	Help Button	
Join Game	Name Textfield	
	Code Textfield	
	Cancel Button	
	Join Game Button	
	Help Button	
Game	Help Button	
	Play Card Button	
	Finish Turn Button	
	Cards Button	
	Popup Functions	

Table 14: List of GUI tests

4.2 Installation guide

1. Go to the following link and download and install the latest Version of Java if you haven't yet:
<https://adoptopenjdk.net/>
2. Launch the downloaded Frantic.jar file

For Linux and MAC Users: launch the jar File from the terminal using `java -jar Frantic.jar`

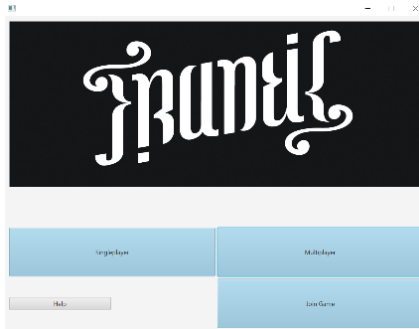


Figure 12: Screenshot of successfully launched game

If you see this window after launching the game congratulations! You have successfully started the game. If you want to play alone against Computer Players choose Single player. From there on the GUI should be self-explanatory. Enjoy!

If you want to play with friends and you are creating the game choose Multiplayer.

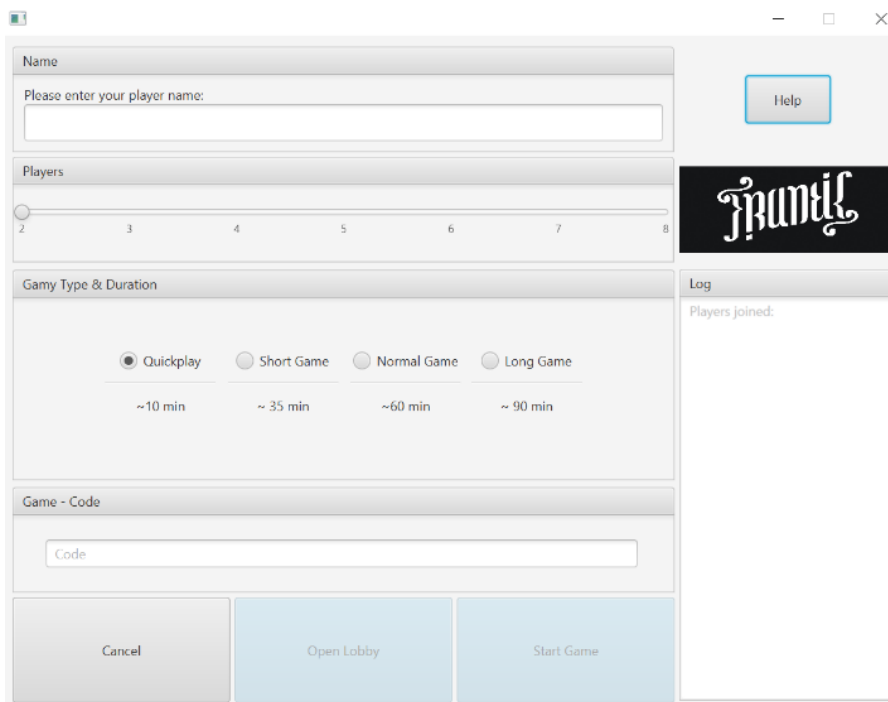


Figure 13: Screenshot of multiplayer lobby to create new game

Only thing to be noted is that the Player count slider includes you as a player, everything else should be self-explanatory. After the settings click on “Open Lobby” to receive the Game Code which the other players need to join your game.

Have fun and rock on!

5 Results

Theme	Description	Status
Functionality		
	All moves and errors are logged.	
	Illegal moves should be detected and prevented.	
	In multiplayer mode the program should create a virtual room with all participants.	
	A multiplayer game must comply with industry security standards to prevent data manipulation and unauthorized access.	
Usability		
	The program must be executable on the current computer systems (MacOS, Windows).	
	The player should be able to enter a game within 30 seconds of starting the application.	
Reliability		
	It must be possible to interrupt the game at any time without losing any data.	
	If the connection to the other players is lost, the program must try to reconnect.	
Performance		
	Creating a connection to a virtual room may take a maximum of five seconds.	
	The move of a computer-controlled opponent must be calculated within 0.5 seconds.	
	The transfer of a player's move to the other players may take a maximum of one second and must be successful in 99.9% of cases.	
Supportability		
	There is a chat function that can be used by all players in the same virtual room.	
Implementation Constraints		
	The use of Java technology is already stipulated.	
Free Open Source Components		
	Free and open source Java components should be used in building the game. That makes JavaFX a strong contender for building the GUI.	
Legal Issues		
	The game's implementation was greenlighted by the original creators. It must be planned for possible licensing restrictions.	
Card Rules		
	numeral yellow cards 1 to 9	
	numeral blue cards 1 to 9	
	numeral green cards 1 to 9	

	numeral red cards 1 to 9	
	numeral black cards 1 to 9	
	2nd Chance	
	Color Swap	
	Exchange	
	Gift	
	Skip	
	Thief	
	Troublemaker	
	Fantastic	
	Fantastic Four	
	Counterattack	
	Equality	
	Inequality	
	Lucky Bastard	
	Nice Try	
	Special Favours	
	Curse	
	Fuck You	
	Mimicry	
	Black Hole	
	Capitalism	
	Communism	
	Crowdfunding	
	Distributor	
	Doomsday	
	Double Taxation	
	Earthquake	
	Event Manager	
	Expansion	
	Finish Line	
	Friday The 13th	
	Gambling Man	
	Identity Theft	
	Last Chance	
	Market	
	Mating Season	
	Merry Christmas	
	Mexican standoff	
	Plague	
	Plus One	
	Recession	
	Repeat	
	Robin Hood	

	Russian Roulette	
	Seppuku	
	Surprise Party	
	The All-Seeing Eye	
	Third Time Lucky	
	Time Bomb	
	Tornado	
	Trust Fall	
	Vandalism	
Achieved	All Achieved points are marked with the following color:	
Not Achieved	All Not Achieved points are marked with the following color: All not achieved points have to be included in the next planning stage for the Pre-Alpha release. Due to unforeseen circumstances, some goals hat to be cut and moved into the next step after the first beta release.	

Table 15: List of project result

The green-colored parts have been successfully implemented and tested. The red-colored parts have not been implemented. The reason for this is explained in the project management chapter.

5.1 Further development goals:

- Upgrade sophisticated AI.
- Dark Theme.
- Train and Update the AI with machine learning.
- DLC (Downloadable Content) / Game-Expansions.
- Game moved from local to open Servers.
- Animations for all User Interfaces.
- Open Game-Lobbies for other players to be able to join.
- Chat function to be able to communicate with other player and community.
- App for mobile devices.

6 Attachment

6.1 Project management

Despite major challenges, the project went mostly according to plan. A big challenge was to create a clear and transparent client-server communication with little prior knowledge. We encountered several problems, which blocked the workflow for the rest of the team. For a long time, the GUI and the game logic could only be tested and further developed with mocks. This process slowed down the implementation of event and special cards. For this reason, only a subset of cards is playable in the beta-release.

6.1.1 Team structure

Small teams were formed for the division of work. During the project, the team's constellations changed. Towards the end the sub teams broke up and issues were solved individually.

Project manager:	Samuel Stalder
Vice project manager:	Lukas Zoss
Team-GUI:	Joël Plambeck (head) & Lukas Zoss
Team-Logic:	Nikita Smailov (head) & Erman Zankov
Team-AI:	Gökhan Bag
Team-Multiplayer:	Tobias Ritscher

6.1.2 Effort estimation

Iteration reviews with the supervisor took place every two weeks. These were used to present the status and to define new tasks. In the meantime, meetings have been arranged to work together on solutions. The rest of the time, all team members worked alone or in small groups.

Tasks	Planned effort per person	Planned effort as team	Effort as team	SW
Introduction, team building	5	35	-	01
Input project management	5	35	-	02
Topic development Frantic	6	42	40	02
Project sketch	7	49	40	03
Presentation project sketch	2	14	10	03
Subtask Iteration-Review #2	10	70	66	05
Team meeting Iteration-Review #2 and debriefing	5	35	35	05
Subtask Iteration-Review #3	10	70	89	07
Team meeting Iteration-Review #3 and debriefing	5	35	35	07
Subtask Iteration-Review #4	10	70	63	09
Team meeting Iteration-Review #4 and debriefing	5	35	35	09

Subtask Iteration-Review #5	10	70		11
Team meeting Iteration-Review #5 and debriefing	5	35	35	11
Technical Report, Presentation, Beta-Release	8	70	77	13
Feedback Beta-Release and technical report II	1	7	-	14

Table 16: Effort estimation of all tasks during the semester

Subtask Iteration #2 (13.10.2020)	Responsibility	Planned Effort	Effort	Result
Draft domain model	Gökhan	10	8	finished
Draft class diagram	Samuel	10	10	finished
Draft Use-Case diagram	Lukas	8	8	finished
Development of the project structure with Gradle JavaFX and test environment	Erman, Nikita	10	10	finished
Draft GUI	Nikita, Joël	6	7	Partly done. The game menu is not planned yet.
Clarification of machine learning	Erman	4	3	finished
Clarification of artificial intelligence	Erman	6	6	finished
Clarification of multiplayer	Tobias	10	9	Partly done. It still has to be clarified which data are sent from the server to the clients and whether the clients should also contain logic.
Setup GitHub	Samuel	6	5	finished

Table 17: List of subtasks of Iteration #2

Subtask Iteration #3 (27.10.2020)	Responsibility	Planned Effort	Effort	Result
Doc: Fully-Dressed-Use-Case	Lukas	4	7	finished
Doc: Definition of further requirements	Lukas	4	5	finished
Doc: Domain model	Gökhan	4	4	finished
Doc: Software architecture	Gökhan, Samuel, Tobias	5	9	finished
Doc: State diagram: game logic	Samuel	4	5	finished
Doc: UML-Package diagram	Gökhan	3	4	finished
Doc: System sequence diagram: Play Thief Scenario	Tobias	5	5	finished
Doc: Design artifacts: UML-Class diagram	Gökhan	5	5	finished
Doc: Documentation of Implementation	Samuel	4	5	finished
Doc: Documentation of project management	Samuel	5	5	finished
Doc: Glossary	Gökhan	3	4	finished

Code: Implementation of the class diagram without logic	Nikita, Erman	7	7	finished
Code: Simple game play in single player mode against a computer-controlled opponent who discards random cards. Special cards and event cards have no effects	Nikita, Erman	7	9	started
GUI: FXML file for displaying the playground	Joël, Lukas	3	4	finished
GUI: FXML file for displaying the menu	Joël, Lukas	2	3	finished
Implementation of MVC-Patterns	Joël, Lukas	2	2	finished
Multiplayer: Definition Protocol	Tobias	2	4	finished
Multiplayer: Definition of the transferred data	Tobias	1	2	started

Table 18: List of subtasks of Iteration #3

Subtask Iteration #4 (10.11.2020)	Responsibility	Planned Effort	Effort	Result
Code: Implementation of the game sequence according to the state diagram	Nikita, Erman	10	9	started
Code: Implementation of the first special card	Nikita, Erman, Joël, Lukas	6	5	finished
Code: Implementation of the first event card	Nikita, Erman, Joël, Lukas	6	5	finished
Code: Unit tests for game sequence	Nikita, Erman	3	3	started
Code: Implementation of Logger	Tobias	4	2	finished
Doc: adjustments	Samuel, Gökhan	10	7	finished
External test project of the server-client communication	Tobias	9	9	finished
Interface definition of the protocol	Tobias	5	7	finished
GUI: display of the cards	Joël, Lukas	5	4	finished
Clear interface definition between logic, GUI and server	Tobias	6	8	finished
AI: Implementation of a simple computer opponent	Erman	6	4	started

Table 19: List of subtasks of Iteration #4

Result: Not all tasks could be finished. The Definition of the interface took more time than estimated and blocked the workflow of the other subtasks. The Implementation of the game sequence and their unit tests are processed in the next Iteration.

Subtask Iteration #5 (24.11.2020)	Responsibility	Planned Effort	Effort	Result
Code: Implementation of the game sequence according to the state diagram	Nikita, Erman	10	9	finished
Code: Unit tests for game sequence	Nikita, Erman	4	4	finished
Code: Implement more special cards	Nikita, Erman	5	5	finished Skip

Code: Implement more event cards	Nikita, Erman	5	6	Finished Black hole
Code: Implementation of a more advanced computer opponent	Gökhan	3	3	Not yet
Code: Unit tests for computer opponent	Gökhan	3	2	Not yet
Doc: Interaction diagram	Gökhan	3	2	Not yet
Doc: adjustments	Samuel	7	7	finished
Integration of the external test project	Tobias	6	7	finished
Definition of Server functions	Tobias	3	2	finished
Modification GUI-Controller	Joël, Lukas	7	6	finished
Definition of dataPackage	Joël, Lukas	4	2	finished
Fully covered game sequence which interacts with Logic, Server, Client and GUI	All	10	9	finished

Table 20: List of subtasks of Iteration #5

Technical Report, Presentation, Beta-Release	Responsibility	Planned Effort	Effort	Result
Doc: Summary of project sketch	Erman	2	2	finished
Doc: Design class diagram (DCD): without UI	Erman	3	2	finished
Doc: Update Play Thief Scenario	Tobias	3	2	finished
Doc: MVC description	Lukas	2	3	finished
Doc: Update GUI images and descriptions	Joël	1	1	finished
Doc: Interaction diagram	Gökhan	1	1	finished
Doc: Documentation of used design pattern and GRASP	Tobias	3	4	finished
Doc: Testing: Unit-, Integrations-, System tests	Lukas	4	4	finished
Doc: Installation guide	Erman	3	3	finished
Push to Master branch and upload code as ZIP	Samuel	1	1	finished
Code: JavaDoc	Joël	2	2	finished
Code: Clean code	Tobias	3	2	finished
Doc: Summary of achieved/not achieved goals	Lukas	3	3	finished
Doc: Management	Samuel	6	7	finished
Formatting of Technical Report 2: List of literature, figures and tables	Samuel	2	2	finished
Doc: Proof reading	Tobias	3	3	finished
Doc: Update glossary	Gökhan	1	1	finished
Slides for presentation	Lukas	2	3	finished
Code: Functional extensions: more special and event cards	Nikita	5	7	finished
Code: Unit test for new special und event cards	Nikita	4	4	finished
Code: Logging game state in GUI	Joël	5	5	finished
Integration test with reported scenario	Nikita	3	4	finished

Code: Illegal moves should be detected and prevented.	Nikita	5	6	finished
Code: Fix Single player modus	Gökhan, Nikita	3	5	finished

Table 21: List of subtasks of final Iteration

6.1.3 Risks

Impact	high	Single player-Mode	Artificial Intelligence	Multiplayer-Modus
	medium	GUI with JavaFX	Game complexity	
	low			Loss of staff (Corona)
		low	medium	high
				Probability of occurrence

Table 22: Risk matrix

6.2 Glossary

Word	Meaning
Deck	Mixture of all drawable hand cards
Hand	Every player has a list of cards on his hand. Game ends if hand is empty
Discard Pile	All played drawable hand cards get collected in the discard Pile
Drawing Pile	Is a subset of deck
Event Card Pile	Set of all event card
Card	Playable object which can be activated. It may activate an event or a special function
Player	Human being who is aware of this card game
Turn	The next Player active if the previous player finished his turn
Round	A round is over if one player has an empty hand
Card Target	The functionality of event or special card can target one or multiple players
Event Card	An event card is a passive card which can be activated during game. See 2.1 for a description of each event card.
Numerical Card	A numerical card is a hand card which can be thrown on the discard pile. It has no special effects.
Special Card	A special card is a hand card which can be thrown on the discard pile. It activates a special effect. See 2.1 for a description of each special card.
JavaFX	The framework used for creating the interactive user interface of Frantic.

Table 23: Glossary of common Frantic words

7 Appendix

7.1 List of figures

Figure 1: Use-Case-Model of Frantic	3
Figure 2: Domain model of the card game Frantic.....	10
Figure 3: Package diagram of the card game Frantic	12
Figure 4: State machine diagram of the card game Frantic	13
Figure 5: Interaction Diagram of placing a card	14
Figure 6: System Sequence Diagram, Play Thief Scenario of the card game Frantic	15
Figure 7: Screenshot of the main menu	16
Figure 8: Screenshot of single player setup	16
Figure 9: Screenshot of in game board	17
Figure 10: DCD of the card game Frantic, part 1.....	18
Figure 11: DCD of the card game Frantic, part 2.....	18
Figure 12: Screenshot of successfully launched game	26
Figure 13: Screenshot of multiplayer lobby to create new game	26

7.2 List of tables

Table 1: List of all earnings	2
Table 2: List of all expenditures.....	2
Table 3: Calculation of profit.....	2
Table 4: Fully dressed Use-Case description for set game	5
Table 5: Use-Case description for UC2 to UC10	6
Table 6: List of tested AI player	21
Table 7: List of tested event cards	21
Table 8: List of tested special cards.....	22
Table 9: List of tested hand cards	22
Table 10: List of tested numeral cards	22
Table 11: List of tested special cards.....	23
Table 12: List of tested GameStateController	23
Table 13: List of system tests	24
Table 14: List of GUI tests.....	25
Table 15: List of project result.....	29
Table 16: Effort estimation of all tasks during the semester	31
Table 17: List of subtasks of Iteration #2	31
Table 18: List of subtasks of Iteration #3	32
Table 19: List of subtasks of Iteration #4	32
Table 20: List of subtasks of Iteration #5	33
Table 21: List of subtasks of final Iteration	34
Table 22: Risk matrix	34
Table 23: Glossary of common Frantic words.....	34

7.3 References

- [1] Rulefactory Frick & Co. KLG. *Rule Factory, Frantic* [Online]. Available: <https://rulefactory.ch/en/frantic/> [Accessed Sept. 27, 2020].
- [2] S. Rohner. (12.09.2020). *Frantic hat auch schon wehen ausgelöst* [Online]. Available: <https://www.fm1today.ch/ostschweiz/stgallen/frantic-hat-auch-schon-wehen-ausgeloesst-139092655> [Accessed Sept. 26, 2020].
- [3] Uno. [CD-ROM/Online]. France: Ubisoft Entertainment, 2016. Available: <https://store.steampowered.com/app/470220/UNO> [Accessed Sept. 27, 2020]
- [4] S. Russell, P. Norvig, "Monte Carlo Tree Search in Artificial Intelligence: A Modern Approach", 4th ed. Hoboken, NJ: Pearson, 05.04.2020.
- [5] DeepMind Technologies Limited. *DeepMind* [Online]. Available: <https://deepmind.com> [Accessed Dec. 08, 2020]
- [6] Rome: Total War. [CD-ROM]. USA: Activision, 2004.