Assignment 1

Team number: Exploding Kittens 14
Team members:

Name	Student Nr.	Email
Sergei Agaronian	2661076	s.agaronian@student.vu.nl
Sofia Fraile Zandejas	2659410	s.m.fraile@vu.nl
Zahraa Salman	2659909	z.salman@student.vu.nl
Tenzin Yangdostang	2707824	c.yangdotsang@student.vu.nl
Marco Huijs	2705658	m.c.d.huijs@student.vu.nl

Introduction

Author(s): Sofia

Exploding Kittens is the most funded Kickstarter board game ever, and the reason for this is its simple ruleset that can be learned quickly together with the clever designs and names used for the game. We decided to implement the game by following its basic structure but with a simpler visual design, that allows for the same features the regular game allows.

At the start of the game, each player is dealt 7 cards from the talon pile, each player also gets a single Defuse card. In the talon, there are some Exploding Kittens, the number of which is determined by the number of active players. The game is played by taking turns drawing cards until someone draws an Exploding Kitten; when this happens, the player who has drawn the card explodes and is out of the game. This process continues until there is only one player left, who then wins the game. Another feature added to the main game line is the ability to defuse an Exploding Kitten with a Defuse card, which allows the user to put the Exploding Kitten card back to the talon pile. Just as with the defuse card, there are many other types of cards with different features and applied rules. To provide an insight into the cards included in our implementation the following example turn can be looked at with the following link: https://www.youtube.com/watch?v=4osKpgjvdYo. To have a more detailed description of the game rules refer to the official manual of the game with the following link: https://explodingkittens.com/how-to-play.

The main goal of the featured implementation and the project itself is to allow the users to play the game with extension packs that introduce new types of playing cards, therefore the card instances are separated from the rulesets of the game. This allows the game to be flexible and it

means that the flow of the game is not affected no matter how many extra cards are added to the playing deck.
For the featured implementation, we will create a GUI with the help of Scene Builder and JavaFX libraries that will provide the user with different buttons to make use of the different cards, end turn, and create special card combinations of the game.
The main type of user expected for this game is the same target audience as for the original edition which comprises the population at the age of seven or older.

Features

Author(s): Everyone

Functional features

When choosing between the graphical interface and the command-line interface for the featured implementation, we rationalized that it would be more pleasant for the user to have a concrete visual representation of the game in front of their eyes. The reason for this is that people tend to process visuals faster than words. In a command-line interface, it is easier to overlook the details of the game. For example, with a command-line interface, the user would have to keep track of all the cards that have been played, current cards in their hand, what tricks they can do, and so on. In contrast, the graphical interface has all of the mentioned details displayed on the screen at all times. This way the user does not need to keep any information in their memory and can focus on the game itself.

Naturally, since this is a card game, there has to be a Card class so we can have interactable card objects with their corresponding properties. Furthermore, there should be a Deck class that would keep track of all the cards in the game and their order, and a Rules class to keep track of what can be played and should be allowed in the featured implementation at a certain point in time. The Rules class also allows the Game class to interact with the user; what cards to display, how to initialize the game, etc. The Game class would be the engine that allows all the actions to occur in the game and be displayed on the screen. Rationally, the user can be represented as a Player object, along with the opponents being computer players, also represented as a Player object. Finally, the user should be able to interact with the game environment, for this reason, user actions are added to the implementation. In the following table, our functional features are stated with their corresponding short description.

ID	Short name	Description	Champion
F1	GUI	The players will be able to access the game through a simple graphical user interface. The interface consists of the following components: - Rules button provides the user with practical information about the game and its goals. - Card buttons (only active when it's the user's turn).	Sofia

		- Card buttons appear when the user draws a card and disappear if they play it. - To play a card, the user has to press the corresponding card button. - Every card has a question mark (?) icon to see what it does. - Use Nope button (only active if the Nope card is in hand and can be played at all times). - Discard Pile card is not visible unless a combination of five different cards is played. - Talon is not visible unless See The Future card is played, then a window pops up with three cards on the top of the talon. - When a player draws an exploding kitten, they can use the defuse card button.	
F2	Turn-based system	The players should be able to take turns according to the rules of the game. For this reason, a **Game State class** is added to the implementation. The Game State will keep track of the order in which turns are taken. Besides that, the Game State also keeps track of the effects applied by a certain type of card to determine what is going to happen in the next turn. When a player ends their turn, this class examines the played cards and applies the corresponding effects to the game environment or the other players. When there is only one player left on the playing field, the Game State class determines the winner and ends the game.	Marco
F3	Deck	The featured game utilizes two main types of decks: - **Talon:** includes all the undealt cards that can be drawn during the game. - **Discard pile:** includes played cards and can only be used when the rules allow. The proposed **Deck class** is going to store all the cards from the playing pool. The main goal of this class is to make the interaction between the piles and the players possible. The players would be able to draw the cards from the piles, shuffle the talon, hide an exploding kitten, and check the top cards of the playing pile with See the Future cards. Furthermore, the cards inside the Deck will be of a different class, namely the **Card class**. This class will contain all the information about the card properties and their corresponding effects.	Zahraa
F4	Player	To make competition possible we need players inside our system. For this reason, the **Player class** is going to be implemented. This class treats two types of agents: human and computer players. Before the game starts, the user chooses the number of competitors they want to play against. Based on the chosen parameter a game is started where the user is the human player and all the other competitors are computer players. The computer players simply perform random actions each turn, meanwhile, the human player is allowed to make decisions for themselves.	Tenzin
F5	Rules	Each player has to stick to the rules of the game. There will be a **Rules class** that contains all the possible moves according to the official manual of the game. For example, a player that has two identical cat cards in their hand is allowed to play them as a pair and steal a random card from one of the opponents. When a player declares an action, this class will check its validity. Furthermore, for the user's reference, there will also be a rules button that opens a window containing all the relevant information about the game. This rules class would be different from the game class as it would only contain the possible moves the players can do, while the game class would contain all the information that the GUI would use to show the game, e.g. whose turn it is, how many players are playing, which player has which card, etc.	Zahraa

F6	User Actions	To play the game, the human player has a fixed set of actions they can perform using the game UI. During a game, the player can click on the following interactive elements of the GUI: - By pressing a **card icon** a user chooses to play it during their turn - **End Turn** button draws the top-most card from the playing talon and ends the current turn unless an exploding kitten is drawn - **Use Nope** allows the user to play a special Nope card - **Question mark icon** is available for every card in the user's hand to consult them on what the card does - **Trick/combo** buttons allow the user to play special combinations of cards. For example, a button to play a pair of identical cat cards. - **Defuse** button allows the user to diffuse an exploding kitten While setting up a new game, the player can click on the following interactive elements of the UI: - **Player Count** allows the user to select the amount of players in the game. - **Start Game** allows the user to start a game with the current settings	Sergei

Quality requirements

Author(s): Everyone

ID	Short name	Quality attribute	Description
QR1	Extendable deck	Maintainability	Expansion packs are quite common for Exploding Kittens. They allow for customization of the standard card pool with additional playing cards that have unique features and effects. Likewise, the digital version of the game should be easily extendable by adding new types of playing cards into the deck. Thus, the Deck functional feature (F3) will be implemented according to maintainability principles.
QR2	Instantaneous results	Responsiveness	Once a player declares a move in the game, the effect of the action should take place within 1 second. The F2 feature responsible for the updates within the game environment will be implemented according to the responsiveness principles.
QR3	Move validity checks	Reliability	Exploding Kittens follows a specific set of rules, a minor deviation from the game structure can potentially lead to unexpected results and game crashes. Thus, when a player attempts to perform an action, the system checks if the desired operation is in line with the game rules. If the system detects an illegal move, it will prevent the user from declaring it. The principles of reliability will be applied to the Rules functional feature.
QR4	Reliable GUI	Usability	The GUI should be free of errors and misleading information. Thus, the

			playing card icons will be unclickable when they cannot be used at the current turn. When the cards are drawn from the pile or played, the corresponding buttons will be adjusted accordingly.
QR5	Matching GUI and Game State	Usability	The GUI (F1) should accurately represent the information contained in the Game State (F2) and the Talon/Discard pile (F3). There should be no mismatch between them at any point in time.

Java libraries

Author(s): everyone

Fastjson
We will use it for reading and writing JSON configuration files containing the description of the cards in the deck. We chose it because it is easy to configure and use from Java code and preliminary experiments make us confident about its correct functioning.

JavaFX
We will use JavaFX along with Scene Builder library to construct a simple GUI for the game.

Scene Builder
Used for developing the user interface of the system. We chose it because it streamlines working with JavaFX and because it can be integrated easily into IntelliJ IDEA.

Tinylog
This library is going to be used for logging. We chose it because it is very lightweight and the available documentation is quite extensive and clear.

Time logs

Team number:	14		
Member	**Activity**	**Week number**	**Hours**
Sofia Fraile	Write down set up and rules	1	1
Zahraa Salman	Search Java libraries	1	1
Sofia Fraile	Define functional features	2	1
Zahraa Salman	Define functional features	2	1
Sergei Agaronian	Define functional features	2	1
Tenzin Yangdotsang	Define functional features	2	1
Marco Huijs	Define functional features	2	1
Zahraa Salman	Define quality requirements	2	1
Sergei Agaronian	Define quality requirements	2	1
Marco Huijs	Define quality requirements	2	1

Tenzin Yangdotsang	Define quality requirements	2	1
Sofia Fraile	Assignment 1: introduction	2	2
Marco Huijs	Java libraries	2	0.25
Zahraa Salman	Java libraries	2	0.25
Sergei Agaronian	Java libraries	2	0.25
Tenzin Yangdotsang	Java libraries	2	0.25
		TOTAL:	14