Assignment 3

Team number: 17
Team members

Name	**Student Nr.**	**Email**
Anna Lehetska	2726397	rofingase@gmail.com
Gabriel Gil Pichelbauer	2726247	gabgilp@gmail.com
Sem Tadema	2663547	s.tadema@student.vu.nl

Format: establish formatting conventions when describing your models in this document. For example, you style the name of each class in bold, whereas the attributes, operations, and associations as underlined text, objects are in italics, etc. Consistency is important!

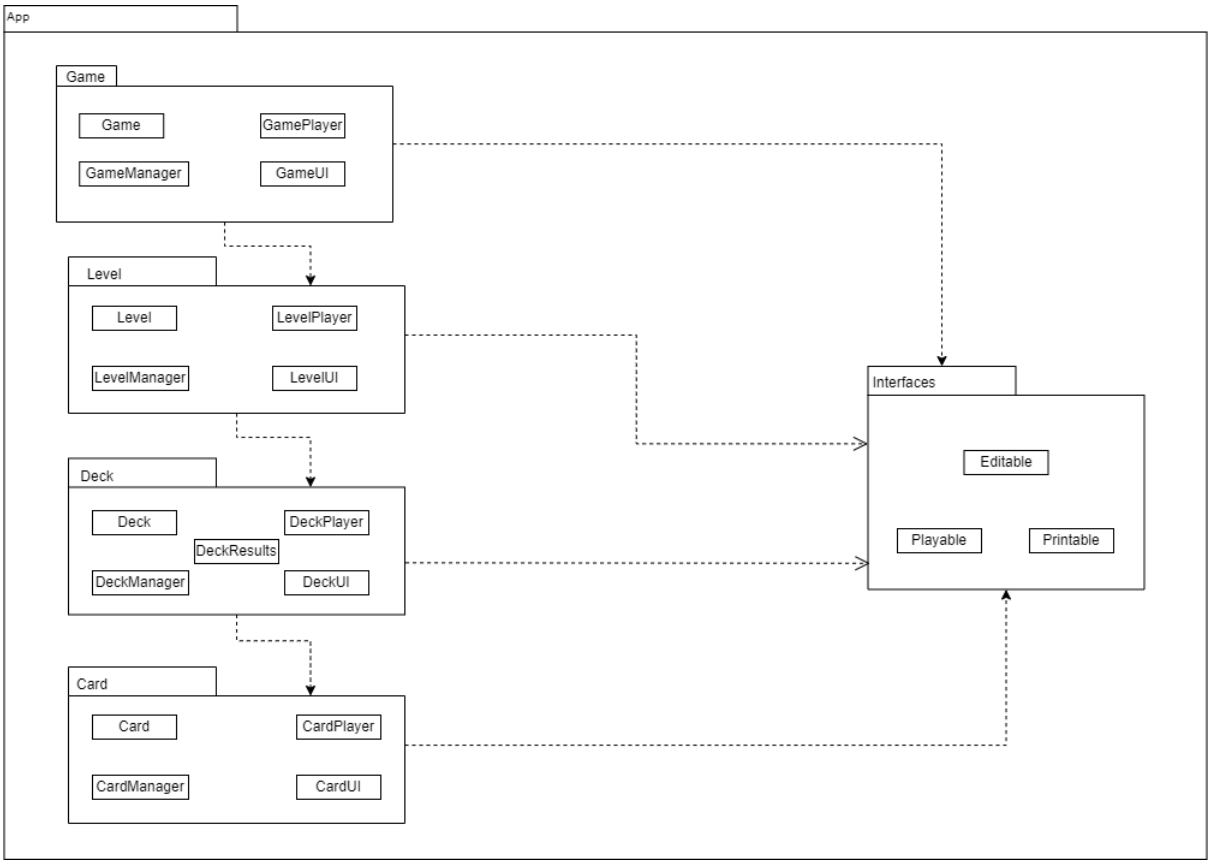
Summary of changes from Assignment 2

Author(s): Gabriel

- We received feedback about our package diagram
    - The package diagram made for Assignment 2 was very flawed therefore it was completely reworked in order to better follow the given paradigm and to resemble the structure of our code.
- We updated our class diagram in order to make better use of design patterns
    - We restructured our project in order to better follow the design patterns given in class, specifically we are following the singleton design pattern and we also added more classes to ensure the single responsibility principle.

Revised package diagram

Author(s): Gabriel

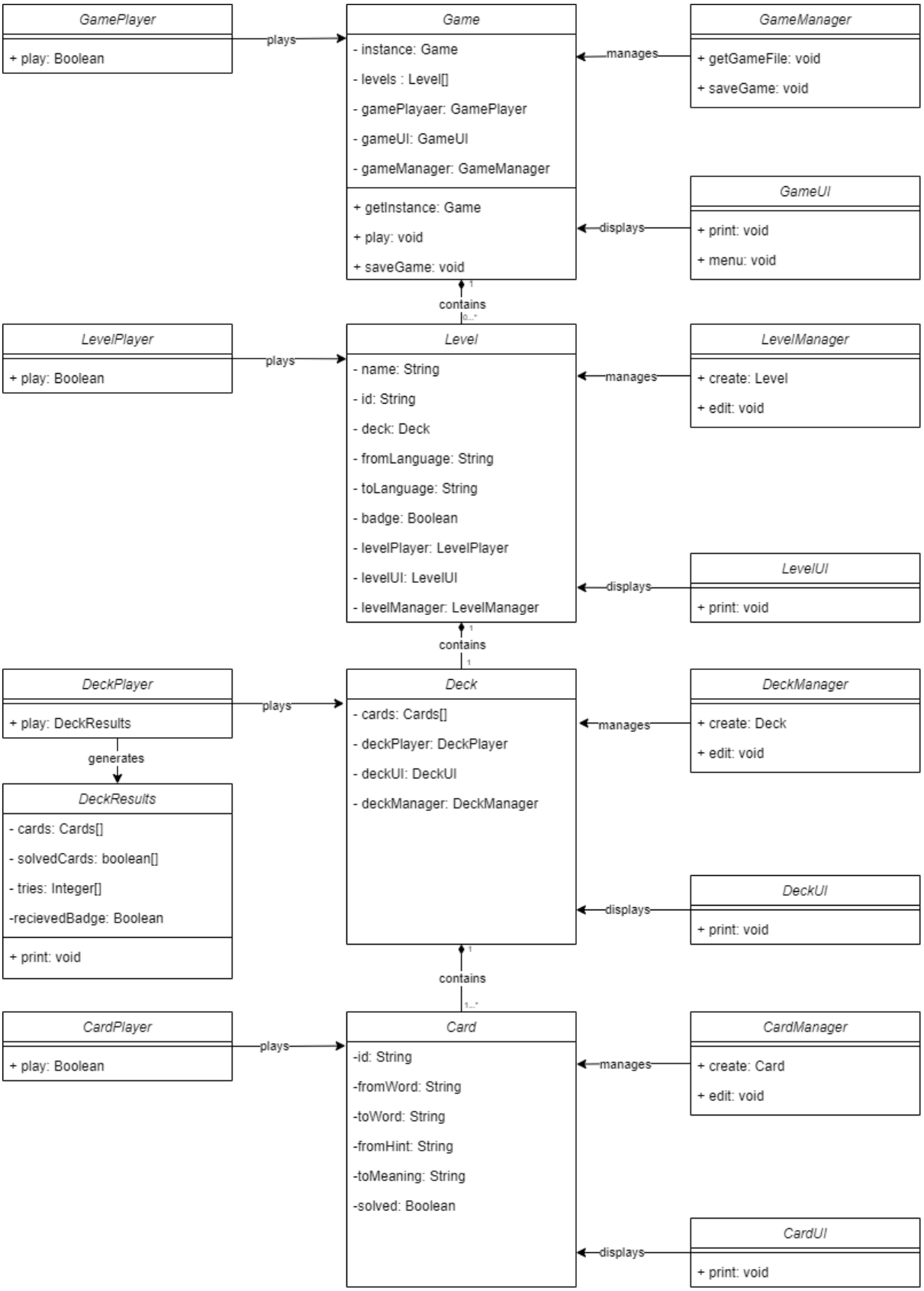


The new package diagram properly reflects the structure and design of the project. Since we decided to divide our classes in order to make sure that every class serves a single purpose, we ensured that all related classes are kept in their own package and unlike the diagram used in assignment 2, this one implements more than just one central package that includes all the classes.

For our design we decided to follow the package by feature structure since we prefer to organise the classes in respect to their domain-related functionality. In all the packages there is the core class and a Player, Manager and UI class, which implement the functions present in the Playable, Editable and Printable interfaces respectively. This way all of the functionality for the components isn't kept in a single class.

Revised class diagram

Author(s): Anna



As stated before in order to preserve the single responsibility principle we added more classes that were not present in the previous assignment in order to handle the distinct tasks for each component of the project.

- Game: is the main class in the project and contains the initial menu where the user decides what to do. GameUI takes care of general UI used throughout the whole project, GamePlayer takes care of the playing process of the game and GameManager handles loading and writing the game file to make the save data persistent throughout playthroughs.
- Level: this contains metadata on each level as well as it contains the deck of cards to be played in each level. LevelUI takes care of displaying the data about the level, LevelPlayer takes care of playing the level and LevelManager is used to create a level or edit an existing level.
- Deck: this contains all the cards present in a level. DeckUI takes care of displaying all the cards in a level, DeckPlayer takes care of playing that deck of cards while it returns DeckResults which contains information about that playing attempt like what cards were solved and how many attempts they took, DeckManager takes care of creating the decks and editing existing decks.
- Card: this contains all the data for the cards themselves. CardUI takes care of displaying an individual card, CardPlayer takes care of playing through an individual card and CardManager takes care of creating and editing existing cards.

Application of design patterns

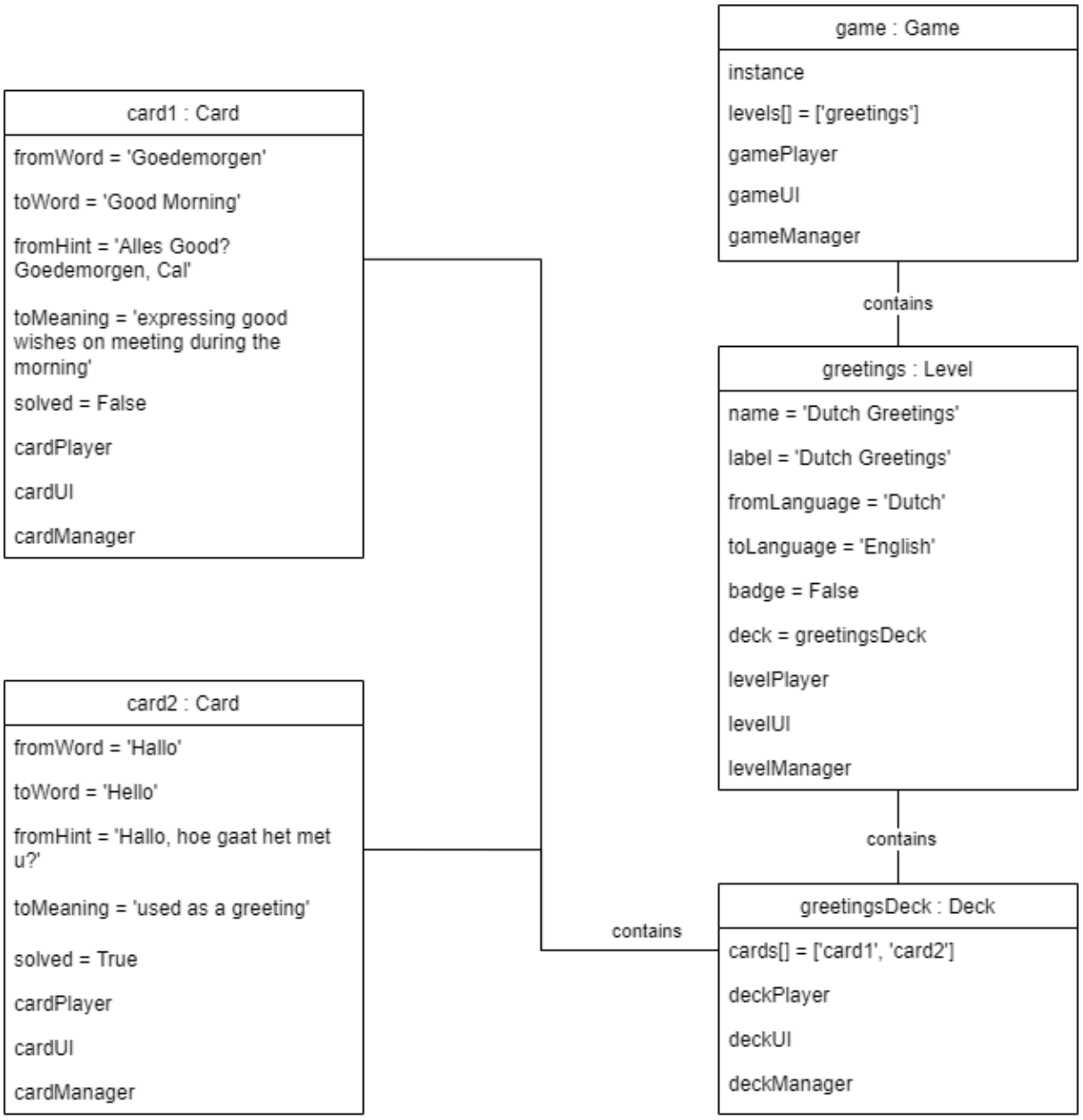
Author(s): Sem

For each application of design patterns, you have to provide a table conforming to the template below.

	DP1
Design pattern	Singleton
Problem	Only one game instance should be running at a time.
Solution	We implemented a game singleton where if an instance of the game has already been initialised then no new instance can be initialised.
Intended use	As soon as the application is started an instance of the game is initialised which loads in all the information from the game file so that the game can display all the levels and the cards.
Constraints	In the implementation checks were placed to ensure that if an attempt is made to initialise a different game object it would refer to the one already created instead.

[optional] Revised object diagram

Author(s): Gabriel

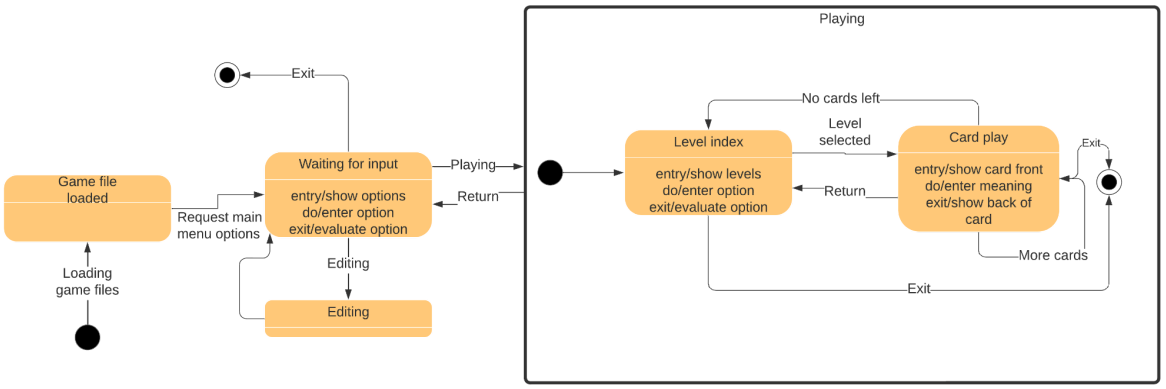


The object diagram has no design changes compared to assignment 2 however it has been updated to represent the new class diagram.

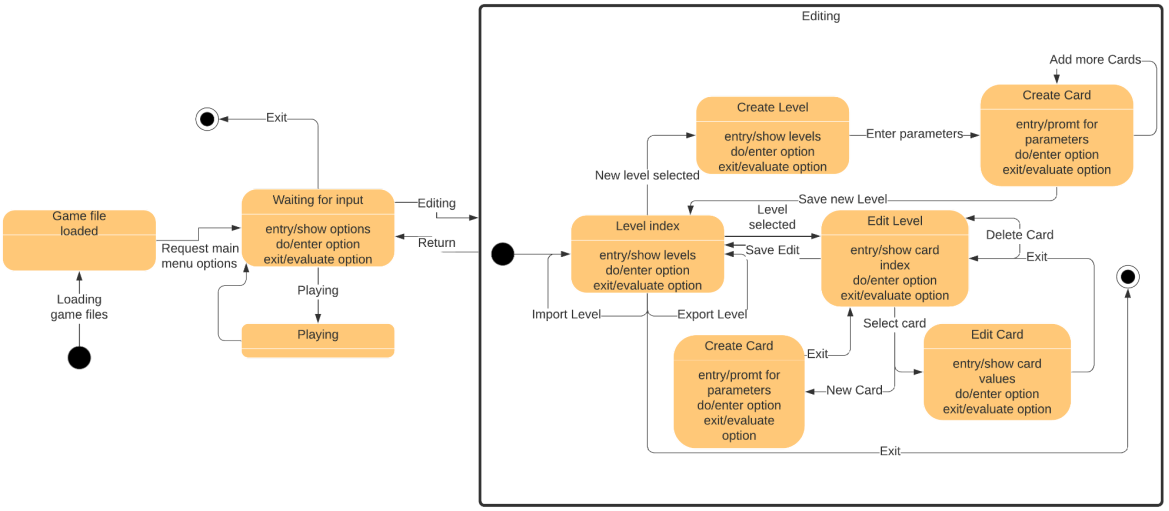
Revised state machine diagrams

Author(s): Gabriel and Sem

Playing a level:



Editing a level

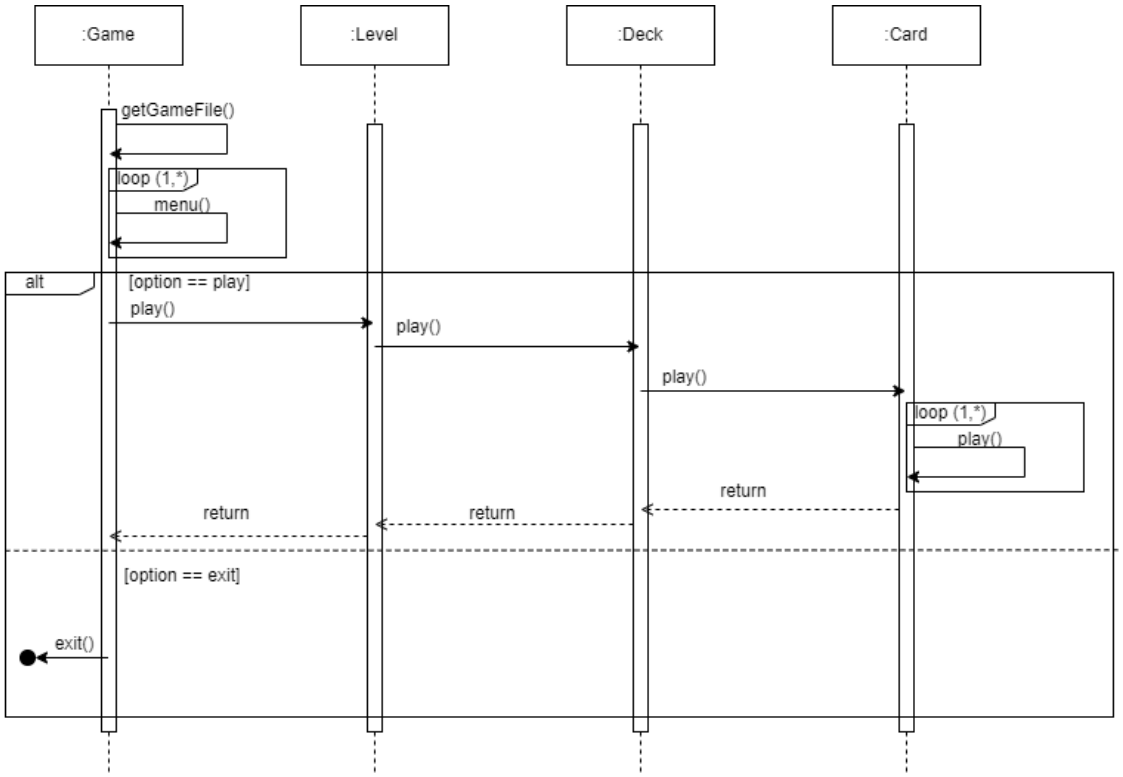


For the state machine Diagrams we were satisfied with the results from assignment 2 therefore we decided to keep them as is since they properly represent the flow of the application.

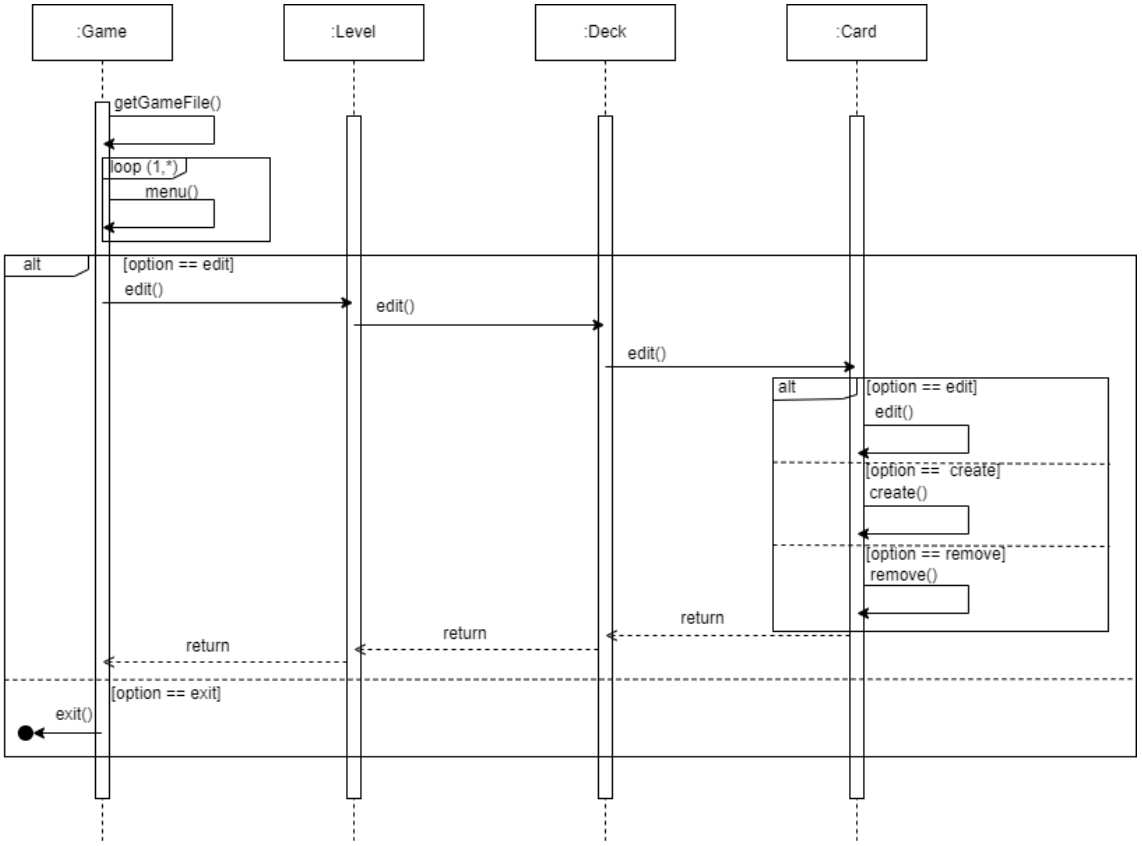
Revised sequence diagrams

Author(s): Anna

Playing a level:



Editing a deck:



For the sequence diagrams we implemented the changes mentioned in the feedback, using the correct types of lines, adding the correct parameters for the alternative sequences and adding the correct parameters for the loops. We also added the deck class which was not present on the previous model.
Another change we made was that for the second diagram rather than presenting the process of creating a card, we presented the process of editing a deck of cards which includes not only creating cards but also removing and editing existing cards.

Implementation

Author(s): Sem, Gabriel, Anna

- For converting our UML diagrams we used an iterative approach were we came up first with a basic conceptual model and attempted to implement it in code, as we saw that the code grew in complexity and we needed to expand its components we came back to the UML and conceptualised these components in the diagrams to then be implemented, this way we ensured that the code and the diagrams remained consistent while add the same time not adding concepts to the diagrams that were not realistically applicable in the timeframe we had.
- Organising our code in a way that we code applied the interfaces that we designed across all the classes provided a challenge but after a while we were able to make a vertically deep interface where the classes would have very few functions that packed a lot of functionality.
- The main java class needed for execution is found at: software-design-2024/src/main/java/com/lexiflash/app/game/Game.java
- The application is purely run on the command line so for execution double clicking will not work, it needs to be run through a CLI.
- The location of the JAR file for directly executing the system is found at: software-design-2024/software-design-2024.jar
- The link to our video showing execution of the application: https://youtu.be/pdMbvDKjHnA?si=3h8Y1w33wjj9nWs7

Time logs

	Team number	17			
	Member	**Activity**	**Week Number**	**Hours**	
	All	Assignment 1	1	2h	each
	All	Assignment 2	3	13h	each
	All	Assignment 3	4	4h	each
	All	Assignment 3	5	5h	each
	All	Assignment 3	6	5h	each