Assignment 2

Team number: 17
Team members

Name	**Student Nr.**	**Email**
Adam Hassid	2786418	adamskylt@gmail.com
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!

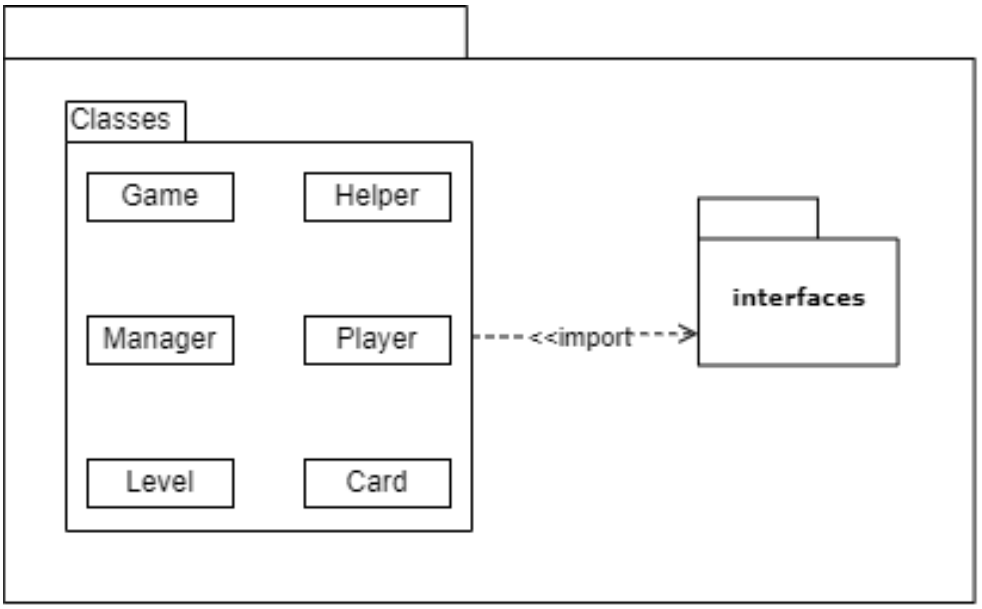
Summary of changes from Assignment 1

Author(s): Sem

- We got feedback on needing to identify the users of the app correctly.
    - We implemented a better description of the user of the app and narrowed it down to individuals looking to expand their vocabulary. Embraced by dedicated language learners, curious minds delving into new words, and those seeking effortless enhancement of linguistic skills, our app caters to a diverse range of users.
- There was some confusion about the CLI feature and how it would include a gamification element.
    - We realised that the CLI feature and the gamification were more of a building block to the app than an actual feature and thus decided to remove it from the features list.

Package diagram

Author(s): Gabriel



Classes

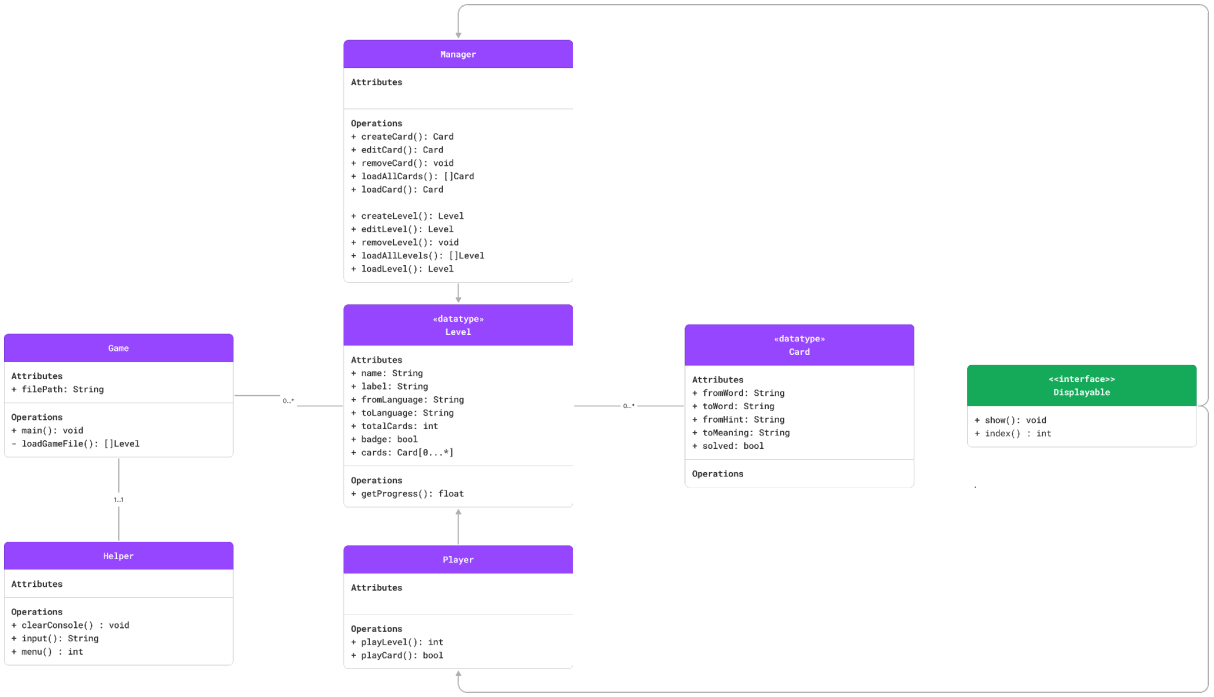
The classes package contains all classes necessary for the execution of the app. The specifics of each class will be all mentioned in the class diagram. Since the project is so simple, the only relation this package has is with the interfaces package.

Interfaces

The interfaces package is for handling how the differences classes are displayed, since it includes the displayable interface.

Class diagram

Author(s): Sem



For our classes, we've chosen to make a difference between datatypes and logic classes. In our flow of the app, there's a very clear distinction between playing mode where you select a level and start playing it by going through all of the cards and solving them. All of this logic will be handled in the Player class where the functions reside to interact with the level or card based on static functions.

In the editing mode, you have a list of levels as well but when selecting one you have the possibility of editing it. When a level has been selected the user has the option to either add or edit a level. When we're in the menu of an existing level, the possibility is given to either edit the properties like labels and languages or we can go to edit the cards belonging to the level. The same goes for the editing of the cards. All of these methods are listed in the Manager class and thus take care of all the editing logic for the application.

When our game class loads we hold some sort of hierarchy where the game is the top level and it has multiple levels and each level has multiple cards. The instances are the classes described above and they serve the purpose of a datatype. We've chosen this approach to make the game scalable if needed. This way the entities of Level and Card can be repurposed for different logic purposes without getting tangled up in existing functionality. We also made all of the attributes for these data types public to avoid writing unnecessary getters.

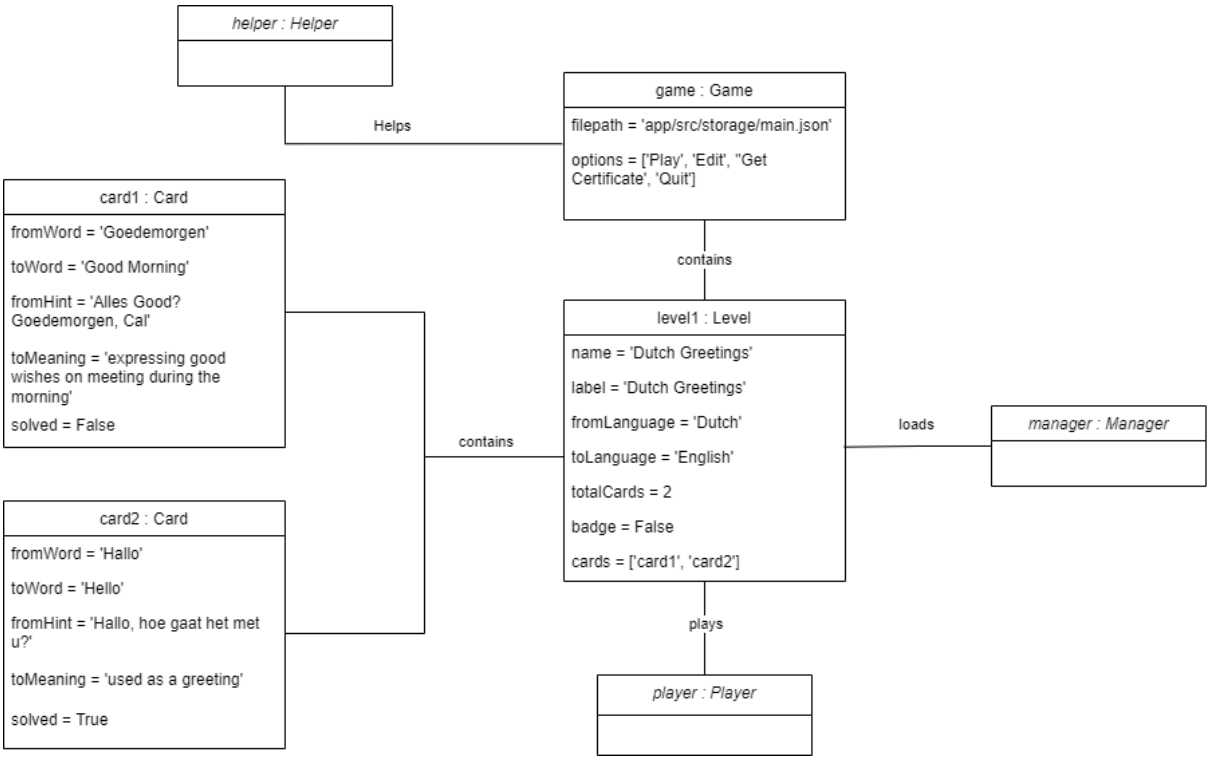
Since the manager/editing and player/play side of the app are relatively similar to each other in ways of displaying data, but still need different approaches, we've made use of the Displayable interface. We hope to utilise this interface to reduce code redundancy in our top-level game logic.

During our process of deliberating about the structure of our app, we came across two approaches on how to handle the game logic in the application. Since most of us have a background in web development, we soon had the idea of setting up a router and passing the logic down the tree of instances. The one problem we discovered during trials with this method was the handling of the functionality to return to the previous page. Since this idea involved a menu which consisted of options each option having a String key and a Callable function it was a scalable solution however it was deemed quite difficult to keep good track of the state of the game.

Our other approach would be to handle all of the game logic at the top level of the tree in the game class. We figured this was less scaleable, but more common practice among GUI games. The function would take in two parameters, one being the state of the game, basically referring to the ‘screen’ of the game and the second one being the choice made on the screen. With this data, we planned to set up two switch statements inside of each other to handle the necessary methods to be called.
Since we’re not entirely set on which design to choose we tried to design our class diagram in a way that would make both approaches compatible. Ideally, we can make use of the best of both worlds. Where we can utilise the ease and common practice of the top-level game logic state function while using the menu function described earlier to maintain code conciseness.
Lastly, we used a Helper class to maintain simple helper functions needed throughout the code. All of these functions are statically called and more methods will probably be added later on in the process. We moved all of these functions out of the Game class for the sake that they don’t only apply to the game, but also to give the project structure more clarity.

Object diagram

Author(s): Gabriel



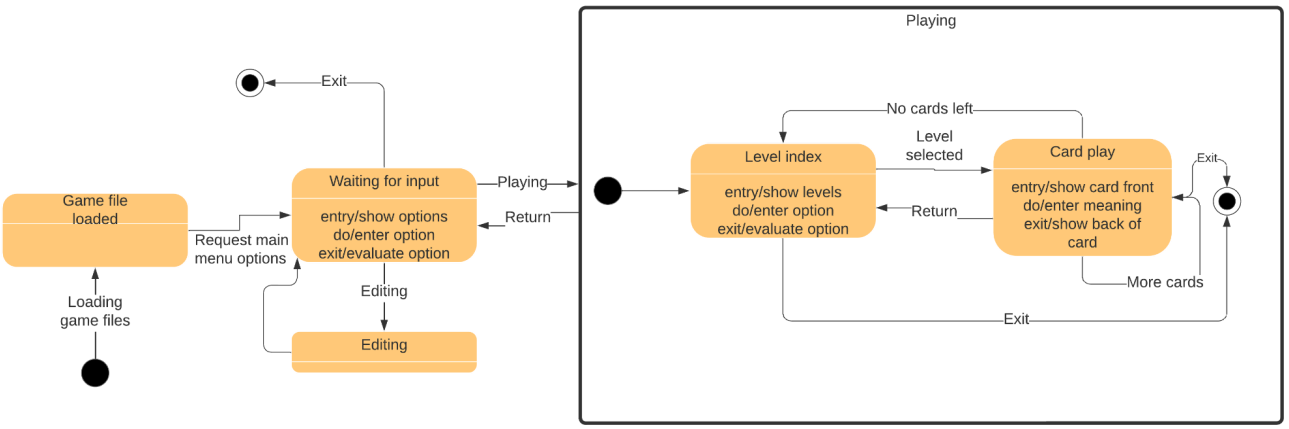
This Object diagram represents an app frame where the player is running the game and is playing the level ‘level1’ which the user knows as the ‘Dutch Greetings’ level. This level only has two flashcards, one for the word ‘Goedemorgen’ and another one for the word ‘Hallo’, both of which include their respective toWord, fromHint and toMeaning which will be displayed to the user as they play the level. The player has not obtained the badge for this level which can be seen by looking at the badge parameter on level1, however it can also be seen by looking at the solved parameter for the cards, where only one of them has been solved so far. The manager is there to perform operations on the levels and cards, in this case loading it into memory from the file in storage. The player handles the logic of actually playing through the level and seeing each of the cards.

Overall this is a basic overview of how the app would look while running, since the user will have the options to play more than one level and the levels will include several times more than just two cards each, but for the sake of demonstration, this frame includes all the basic elements that would be present on a regular use of the app.

State machine diagrams

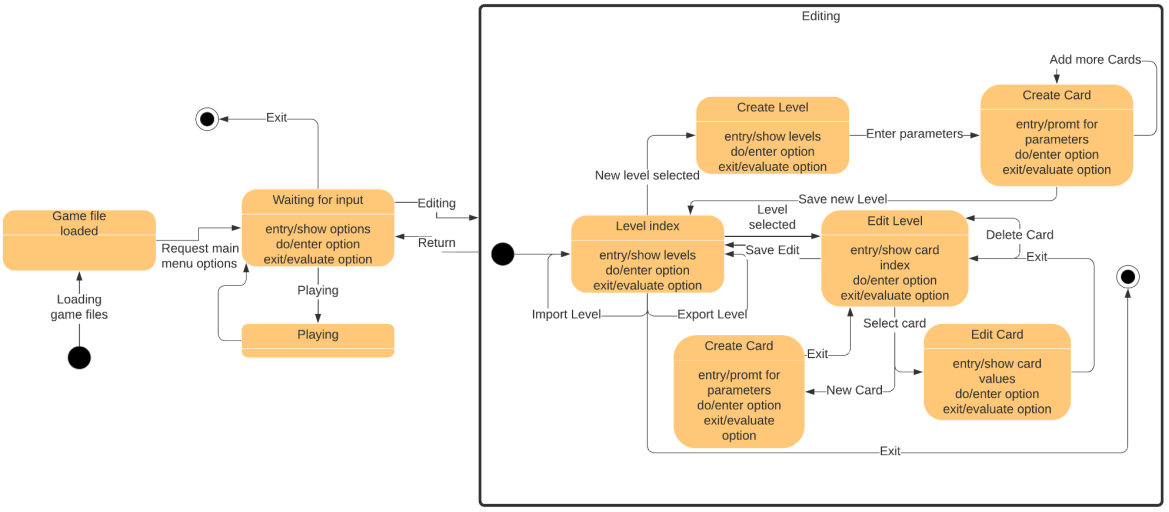
Author(s): Gabriel, Sem

Playing a Game:



When the system is initialised, the game file containing all the levels and progress is loaded into memory and then immediately after, the user is prompted to choose what they would like to do. They have the option to edit levels, play levels or exit. This state machine is focusing on the playing of the levels therefore editing levels has been abstracted. Playing the levels consists of a loop where all levels are indexed and the user can choose to play one of them or to go back to the menu. If the player chooses to play one of them, then another loop will begin where the user is shown a card and they have to enter an answer or they can choose to stop and go back to the menu. Once all the cards have been played, the result of this session is displayed to the user and the game goes back to the level selection state.

Editing Levels:

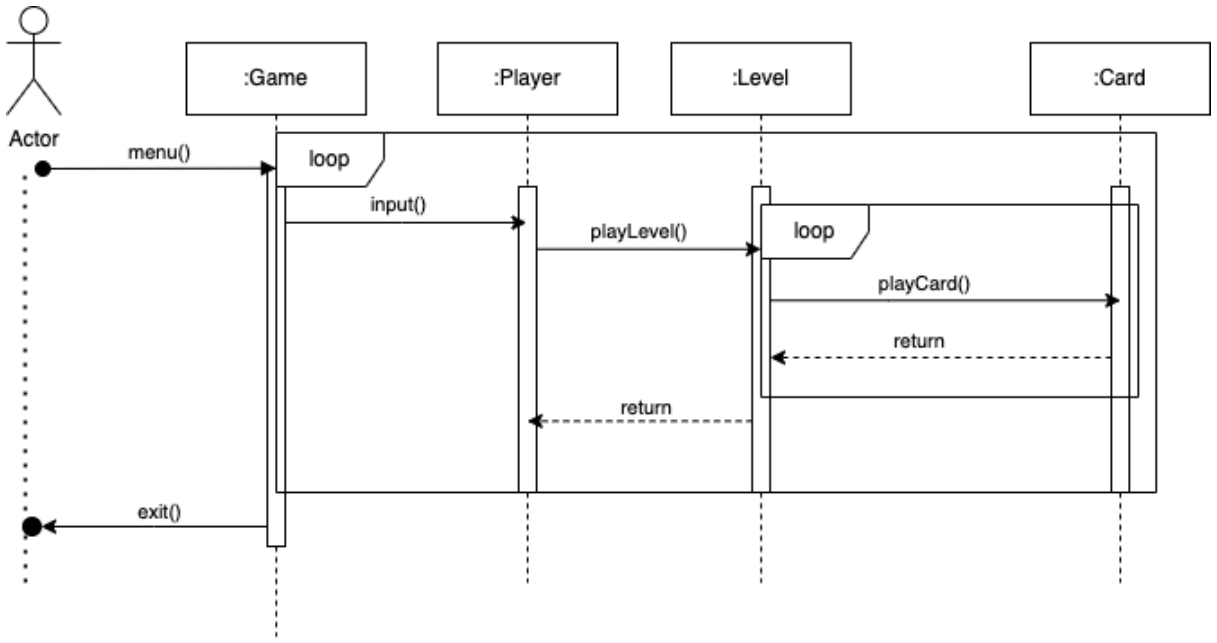


The process of editing levels starts the same as playing a level but on the menu the user chooses to edit levels instead of playing. In the editing mode the user is displayed an index of all the levels and the user can take one of these options: create a new level where they are first prompted to enter the information about the level and then they are prompted to create cards for the level. Levels must contain at least one card so once the first card is created, the user can create more cards or go back to the editing menu. The user can also edit a level, if they select this option they are then prompted to modify one of the cards on the level, create a new card for the level, or delete a card from the level. This is a loop of prompting the user what modification they want to do to the level until they choose to go back to the editing menu and save the new state of the level. The user can also select to either import a level or export a level from the editing menu and the last option they have is to exit the editing menu and go back to the game menu from which they can go back to edit, play a level or exit the game.

Sequence diagrams

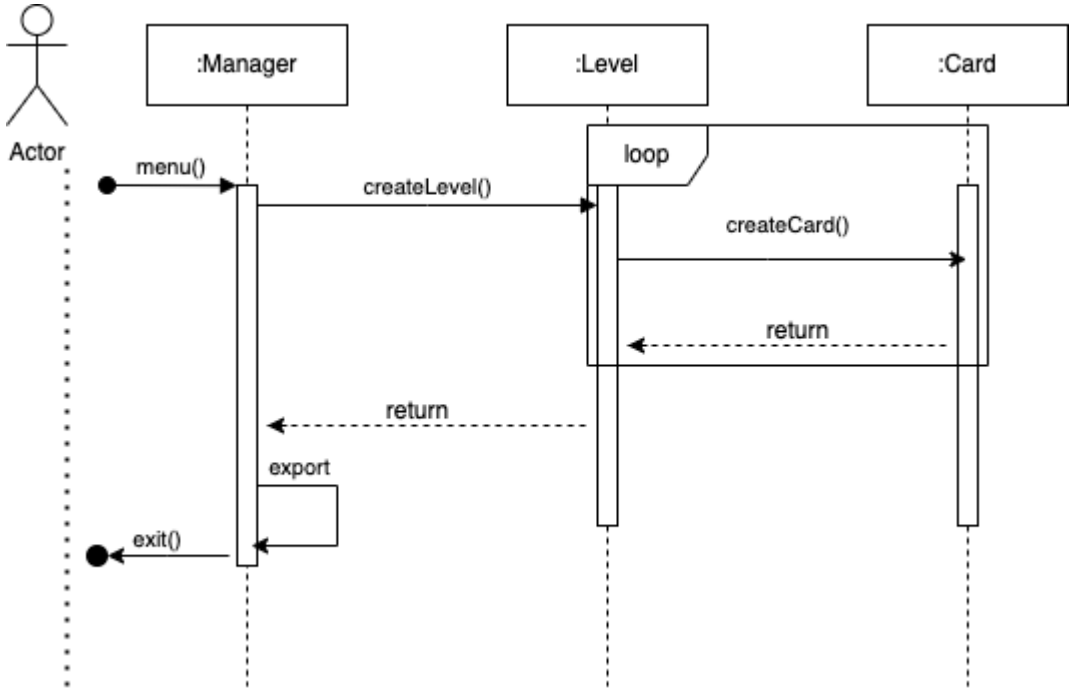
Author(s): Anna

Playing a level



At first the user is displayed a menu from which they can decide what they want to do with the app. If the player chooses to play the game from the menu, they are then prompted to choose a level and once they choose, the player starts the process of playing the level. Playing the level consists of playing all the individual cards from the level and after the user enters an answer, it returns whether the user was correct or not. Once all the cards have been played, the results are shown to the user and the system continues to either repeat the same process with another level or the player can quit the game.

Creating a new level:



At first the user is displayed a menu from which they can decide what they want to do with the app. If the player chooses to edit levels then the Mangaer is called which can do CRUD operations on the levels and cards. In the sequence the player chooses to create a new level. This action initiates a loop to start creating cards for the level. Once they have created all the cards that they want, the manager returns them to the game which prompts the user what action they would like to do next, the user can export the level or the can just exit the editing mode and go back to the menu..

Time logs

	Team number	15			
	Member	**Activity**	**Week Number**	**Hours**	
	All	Assignment 1	1	2h	each
	Sem, Gabriel, Anna	Assignment 2	3	13h	each
	Adam	Assignment 2	3	1h30m	