KENKEN

Delivery version: 2.0

Spring term

Course 2023/24

Group 43.1

Caterina Moll Pons

David Vendrell López

Javier Sánchez Zárate

Marcos Roca Voltá

caterina.moll@estudiantat.upc.edu

david.vendrell.lopez@estudiantat.upc.edu

javier.sanchez.zarate@estudiantat.upc.edu

marcos.roca@estudiantat.upc.edu

Table of contents

Static diagram conceptual model.

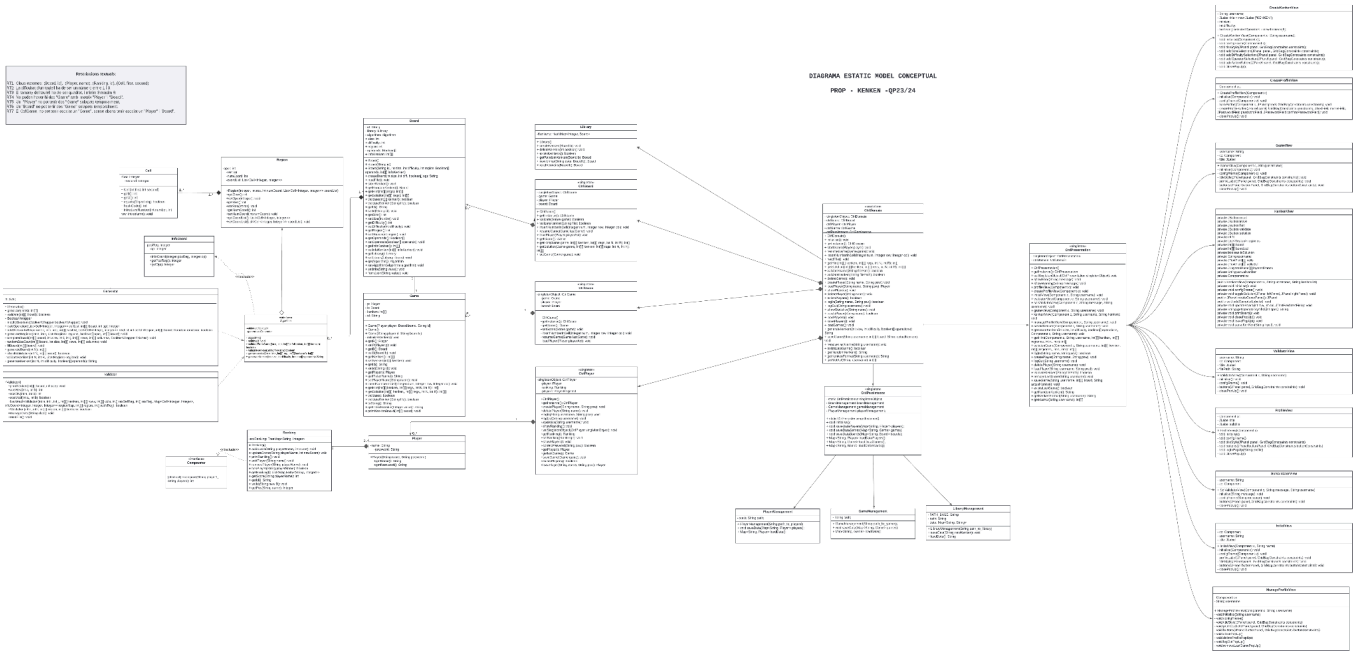

Image 1: Static diagram conceptual model. You can see it in more detail in the DOCS folder.

DOMAIN LAYER

Description of attributes and methods of the classes.

CtrlDomain

This acts as a central controller in charge of coordinating the interactions of the classes of the domain, such as ‘Player’, ‘Board’, ‘Game’ and their controllers. It acts as a high-level interface. It is a singleton class, which means that there can only be one instance of this class in the whole system. This ensures consistency and simplifies the management of control operations.

This is an instance of CtrlPlayer, CtrlGame, CtrlBoard. In other words, these last three controllers are an integral part of CtrlDomain, so their existence is linked to the existence of CtrlDomain. In addition, in this release we also include the CtrlPersistence instance, which is in charge of persistence.

CtrlPlayer

The `CtrlPlayer` class acts as the player controller for the system. It coordinates the operations related to player management, such as creation, deletion, connection and disconnection. This class is a singleton, which means that only one instance of this class can exist in the whole system. It also uses the ‘PlayerRegisted’ class, to simulate that we have persistence and store all the players and their attributes there.

CtrlBoard

The `CtrlBoard` class acts as the controller of the system's tables. It coordinates the operations related to the management of the tables, such as the creation, the reading of files and the obtaining of a random table. This class is a singleton, which means that there can only be one instance of this class in the whole system.

CtrlGame

The CtrlGame class coordinates actions related to the game, such as game administration, inserting names into the cells of the grid and managing the download of player data. It uses the Singleton design pattern to ensure a single active instance at a time, and acts as a bridge between user interfaces and game logic.

Player

The `Player` class represents a player in the system. It contains information such as user name, password, games played and other statistics.

This class allows the management of the player's personal data, such as creation, deletion, login and statistics display. It also keeps a record of the player's last game played and allows new ones to be created.

Board

The `Board` class represents a game board for Kenken.

It encapsulates the information needed to represent a Kenken game board. It provides functions for creating, displaying, and reading game boards, as well as obtaining information about their size and difficulty. It is an essential part of the game system, as it manages the boards with which the players interact.

Game

The `Game` class manages a specific game of the Kenken game, which includes information about the associated player, the used board and the values of the Kenken cells. It provides functions for inserting names in the cells of the box, deleting the game and printing the box with the current values. It is essential for player interaction with the game.

Library

The `Library` class manages the collection of available Kenken boxes and provides functions for their creation, storage and retrieval.

The `Library` class acts as a repository for the available Kenken boxes. It allows the creation, assembly and retrieval of Kenken tables, as well as the reading of data from a file and its conversion into reusable objects by the program. It is an essential part of the management of the available boxes in the Kenken game.

Ranking

The `Ranking` class is responsible for managing the ranking of the players in the system.

It provides functions to manage and display the ranking of the players in the system. It maintains a collection of players and their scores, allowing players' scores to be added, deleted and updated. This ranking is important to show the relative ranking of players according to their activity and successes in the system.

Region

The `Region` class represents a region of the Kenken board, which may contain multiple cells and has an associated operation and an expected result.

The `Region` class defines the properties and behaviors of a region of a Kenken board, such as the associated operation, the expected result and the list of coordinates that make it up. It is used to manage and validate the rules of the Kenken game.

Cell

The `Cell` class represents a cell of a Kenken board, which contains a pair of indices representing the row and column of the cell.

The `Cell` class encapsulates the information of a cell of a Kenken board, with its row and column indices. This class is used to manage the cells of the table and their relationships to other components of the game.

Algorithm, Validator, Checker

Those are the classes where we find most of the code of the algorithms and their main functionality. We will go into more detail about their structure and implementation in the next section.

Here we have chosen to follow a Strategy pattern.

This is a behavioral design pattern that allows us to define a family of algorithms, place each of them in a separate class and make their objects interchangeable.

To do this, the Algorithm class is an interface that calls the different algorithms.

You can also find more detailed documentation about each method of all the classes in the documentation generated by Doxygen. Specifically in the folder /DOCS/doxygen/html/index.html

PERSISTENCE LAYER

Description of the classes.

LibraryManagement

The LibraryManagement class is responsible for managing the KenKen library within the system. Its main function is to enable the storage and retrieval of different KenKen boards that have been created or imported into the system. This class acts as an intermediary between the physical storage of data (in a file) and the application logic that needs access to this data.

Its main functions are the following:

- Save Data: Stores new KenKen boards in the library file, ensuring that existing data is not overwritten but appended. This allows the library to grow over time as more boards are created.
- Load Data: Provides the capability to load KenKen boards from the library randomly. This is useful for providing the user with a random board to play or solve.

JSON Format: Uses JSON to store KenKen data, which is a lightweight and easy-to-handle format for complex data structures like KenKen boards.

Data Structure:

Map<String, String> data: Uses a map to store KenKen boards with a unique identifier for each board. This approach allows for quick and efficient access to the data.

PlayerManagement

The PlayerManagement class is responsible for managing player data within the system. Its main responsibility is to provide a mechanism to save and load player information, including their profiles and game states.

Its main functions are the following:

- Save Players: Stores player information in a file, including their profiles and any data related to their progress in KenKen games.
- Load Players: Retrieves player information from the file, allowing the system to load player profiles when the application starts.

JSON Format: Similar to LibraryManagement, it uses JSON to store player data. This format facilitates the serialization and deserialization of complex objects like player profiles.

Data Structure:

Map<String, Player> players: Uses a map to store player data, where the key is the player's name and the value is a Player object containing all relevant information about the player.

GameManagement

The GameManagement class handles the storage and retrieval of game data using a file-based persistence mechanism. It ensures that the state of the game can be saved and loaded, allowing players to continue their games at a later time.

Its main functions are the following:

- Save Game Data: Writes the current state of all games to a file in JSON format. This includes all necessary information to resume the games exactly where they were left off.
- Load Game Data: Reads the game data from the file and reconstructs the game states, making them available for the application to use.

JSON Format: Similar to LibraryManagement, it uses JSON to store player data. This format facilitates the serialization and deserialization of complex objects like player profiles.

Data Structure:

Map<String, Game> games: Uses a map to store game data, with the key being a unique identifier (such as a player's name or game ID) and the value being a Game object containing the game state.

Justification data structure used

In all classes, the Map data structure is used for the following reasons:

Efficiency: Maps provide a constant average access time O(1) for search, insert and delete operations, which is very efficient for managing large amounts of data.

Uniqueness: They ensure that each key (either a KenKen identifier or a player name) is unique, avoiding duplicates.

Simplicity: The Map API is simple and provides clear methods for the necessary operations, making it easy to implement and maintain the code.

The use of JSON for serialization and deserialization of the data in these maps allows easy integration with text files, which is useful for data persistence. In addition, JSON is a human-readable and widely used format, which facilitates debugging and interoperability with other tools and systems.

PRESENTATION LAYER

Description of the classes.

CtrlPresentation

`CtrlPresentation` acts as the controller for the presentation layer, managing the interactions between the user interface and the domain layer. It coordinates the flow between different views and handles user actions like authentication, profile management, and game operations.

Their functions include:

- Manages transitions between different views.

- Provides methods to show various messages (error, warning).

- Handles user authentication and profile management actions.

- Interfaces with `CtrlDomain` to perform operations like creating players, logging in, and saving games.

CreateProfileView

The `CreateProfileView` class is responsible for rendering the user interface where users can create new profiles. It collects user input for creating a new profile and validates the input before interacting with the backend to create the profile.

Their functions include:

- Displays fields for entering a username and password.

- Provides validation to ensure all fields are filled and passwords match.

- Interacts with `CtrlPresentation` to create a new profile.

CreateKenkenView

`CreateKenkenView` allows users to specify parameters for creating a new KenKen puzzle. It provides options to customize the puzzle, such as selecting the board size, difficulty, and operators.

Their functions include:

- Allows selection of board size, difficulty, and operators.

- Initiates the generation of a new KenKen puzzle based on selected parameters.

- Interacts with `CtrlPresentation` to generate and display the new puzzle.

GameView

`GameView` manages the user interface for playing a KenKen game. It presents the KenKen board and provides various controls for gameplay, such as resetting the game, getting hints, validating the puzzle, and displaying the solution.

Their functions include

- Displays the KenKen board and provides options to reset, leave, get hints, validate, and show solutions.

- Interacts with `CtrlPresentation` to handle game operations and validate the board.

InitialView

`InitialView` is the starting view that users see upon logging in. It serves as a main menu, providing access to different functionalities of the application like starting a game, viewing rankings, and managing profiles.

Their functions include:

- Displays options for starting a game, viewing the ranking, and managing profiles.

- Interfaces with `CtrlPresentation` to transition to the selected view.

ManageProfileView

`ManageProfileView` allows users to manage their profiles, including logging out, deleting profiles, and removing saved games. It provides a user-friendly interface for performing these profile management tasks.

Their functions include:

- Displays options for profile management.

- Interacts with `CtrlPresentation` to perform profile-related operations.

ProfileView

`ProfileView` displays the interface for users to log in or create new profiles. It offers a user-friendly login screen and options to navigate to the profile creation view.

Their functions include

- Provides options to log in or create a new profile.

- Displays a pop-up for logging in, and transitions to the appropriate view upon successful login or profile creation.

SolValidatorView

`SolValidatorView` shows the result of KenKen puzzle validation. It indicates whether the puzzle is valid or invalid and provides options to either validate another puzzle or return to the game menu.

Their functions include:

- Displays whether the KenKen puzzle is valid or invalid.

- Provides options to validate another puzzle or return to the game menu.

ValidatorView

`ValidatorView` provides an interface for users to upload and validate a KenKen puzzle. It handles file selection, performs validation on the uploaded puzzle, and shows the validation result.

Their functions include:

- Allows users to upload a KenKen puzzle file.

- Validates the uploaded puzzle and interacts with `CtrlPresentation` to display the result.