UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

CORSO DI LAUREA IN INFORMATICA

Insegnamento Laboratorio di programmazione ad oggetti

Chess Game

A.A. 2022/2023

Summary

Chapter 1 – Case of Study

1.1 – App Description

Chess Game

Our game application allows you to immerse yourself in the fascinating world of chess. It supports all standard chess rules, enabling you to play against a virtual opponent, challenge a friend, or even watch a game between virtual opponents!

The app includes a series of features designed to enhance the gaming experience. It allows you to display, at each turn, the pieces that can be moved; to show the available moves for each selected piece; to undo moves (up to a maximum of 5 during the game) and redo them; and to detect potential situations of check, draw (if no piece is captured or pawn moved for 50 moves by both players), and checkmate. Finally, it allows a player to resign—or to pause and save a game to resume later.

Additional features are offered when selecting a game to resume from the saved ones. The system enables the user to view saved games in descending order based on the number of moves made, in descending order based on the total number of pieces on the board, and, finally, in ascending order based on the cumulative value of the pieces present.

The Java-based chess game app has been developed with a strong focus on stability and performance, ensuring a quick and smooth response to inputs and an uninterrupted gaming experience.

The app does not allow the use of special moves; therefore, it is recommended for beginner users!

1.2 – Features

The app is intended for two types of players, namely the User and the CPU. Below are the detailed features of the chess game app:

- **User vs CPU Game:** The system gives the user the opportunity to challenge a virtual opponent. The CPU selects pieces and makes random moves using the "Random()" class from the "java.util" library, while still adhering to the established chess rules.

- **User vs User Game:** The system allows the user to challenge another user locally. To select and move a piece, the user must input commands via the terminal. This is enabled by the Scanner() class from the "java.util" library.

- **Watching a CPU Game:** The system provides an option for the user to view a game played between virtual opponents. As mentioned before, these opponents choose and move pieces completely at random.

- **Display of Available Pieces:** During their turn, the user can see which pieces have valid moves. Each player has a list of associated pieces managed through the List() interface provided by "java.util". A deterministic algorithm iterates over the player's list and applies a validation process on each piece to determine its valid moves.

- **Display of Available Moves:** The system displays, in the terminal, the available moves for a previously selected piece. The deterministic algorithm facilitating this employs a "HashBasedTable" from the Guava library and a List as a container for possible moves. The algorithm retrieves the piece's position from the table and, based on the type of the selected piece, computes all valid positions (in accordance with the game rules). These positions are then inserted into the list, which serves as the algorithm's output.

- **Undo Moves:** The system allows users to undo the last move made, up to a maximum of 5 times during the game. All moves made are stored in a list. The undo functionality uses a deterministic method that takes the list as input, returns the penultimate element, and removes the last element from the list.

- **Check Detection:** The system checks whether a check has occurred after a move. This check is executed twice:

    - When the user makes a move.

- At the start of the player's turn.

In this case, the deterministic algorithm takes the King's most recent position (retrieved from the table) and, based on it, calls a method that verifies the valid moves of all opposing pieces present in the table. If the King's position appears among these valid moves, the algorithm returns a boolean value of "true".

- **Draw Detection:** The system offers the possibility to end the game if no pieces are captured or pawns are moved for up to 50 moves. A static counter manages the draw count.

- **Checkmate Detection: The system determines whether, after a check, a checkmate situation exists. It does so by checking the list of valid moves available for the King; if this list is empty, the algorithm returns a boolean value of "true", signifying the end of the game.**

- **Player Resignation**: The system allows users to resign at any turn, thus ending the game. Game termination is handled through an exception called "InterruptedException" provided by Java.

- **Game Saving:** The system allows users to save the game at any turn. The saving feature is implemented using a Container class that holds a list of games. When the user decides to save, the algorithm performs "Serialization" and "Deserialization" operations, facilitated by the "Serializable" interface (implemented by all relevant classes) from the java.io library. Serialization involves saving an object as a string to a file, while Deserialization extracts the string from the file and converts it back into an object for further operations. Hence, the algorithm first deserializes the file to check for existing games. If games are present, the current game is added to them; if not, it is saved as the first game in an empty file.

- **Game Resumption:** The system enables the user to load saved games. The corresponding algorithm deserializes the file and returns a list of all saved games, allowing the user to select the desired game.

- **Displaying Saved Games Based on Specific Order:** The system allows the saved games to be displayed sorted by three criteria: the number of pieces on the board, the number of moves made, and the overall score. The sorting is implemented using classes that implement the "Comparator" interface from the java.util library.

1.3 – Domain Model