Design Descriptions

Team NASA

1 Main classes

Below we have a short description of all the main classes in the project, separated into subsections.

1.1 Board

The implementation of the chess-board. It consists of a number of squares, saved in an ArrayList, one square for each position on the board. The board keeps track of the move-history (which moves was made previously on the board), whose turn it is - color of the pieces that are moving, and which color is on the bottom of the board.

1.2 Square

A square is an implementation of a chessboard-position. It contains information about where on the board it is, and whether or not it contains a piece. You can get a square from the board, using it's position. Contains a toString() algebraic representation of the square.

1.3 Move

Immutable class that stores a move. Contains information about the square you're moving from and to. Which piece you're moving, and alternatively, a captured piece. Contains a toString() algebraic representation of the move.

1.4 ChessGame

The ChessGame class ties together and keeps track of logic surrounding the current game of chess. This includes an implementation of the game clock, deciding when the game is over and updating player statistics after a game.

1.5 GameInfo

Holds information about the game; such as player info, opponent info, which game type (singleplayer/multiplayer) and which AI to use if any.

1.6 Checkerboard

Draws the board to the game scene and handlers user input/output.

1.7 Chess

The chess class gets called from the libgdx desktop launcher and initializes the GUI/Chessgame. It also connects to the playerfile

1.8 AbstractPiece

Abstract class that implements methods for all the common characteristics for the pieces in the game. The class is extended by all the piece classes which have their own methods for different specific rules.

1.9 AIEasy/AIMedium/AIHard

The AIEasy and AIMedium class finds and returns suitable moves for a player. The AIEasy just return a random move, while AIMedium does some calculations and choses the best move it can find. AIHard performs some harder calculations than AIMedium and does outperform it in every case.

1.10 Database

The Database class deals with reading and writing to the external database we have hosted at AWS. The class implements IDatabase methods, such as listPlayers, registerPlayer, getPlayer, isPlayerRegistered and updatePlayer.

1.11 Queries

Small helper class for Database with all the SQL queries.

1.12 AbstractScene

An abstract class responsible for the graphical user interface. It allows the screen classes to draw upon the same Stage and implement all the methods that a screen should implement. The Stage is responsible for drawing the graphical elements (also known as actors) to the screen. This class is currently inherited by the three classes MainMenuScene, GameScene and VictoryScene. The class “Stage” and the interface “Screen” are imported from the LibGDX library.

1.13 SceneManager

Manager for all the scenes to make navigating scenes easy.

1.14 AbstractSetup

A setup for a chess-board. Used when creating a new ChessGame, to determine how the board will be set up initially. Contains some helper-methods shared across all the initial board setups.

1.15 DefaultSetup

Initial board setup for standard chess rules.

1.16 Chess960Setup

Initial board setup for Fischer Random Chess rules.

1.17 AllTests

Runs all the test for the entire project. This includes tests for board structure,pieces, both of the AIs and the menu options.

1.18 AudioManager

Small utility class allowing the user to listen for sound-effect, such as move, hint and undo-sound.

1.19 AnimatedImage

This is a class that inherits from the class Image (from the LibGdx library) and therefore indirectly from Actor. It can therefore be added to our UI stage as a graphical element (more specifically an animation).

1.20 CreateAnimation

This creates an Animation from a spritesheet and can be adapted to different desirable specifications. This class is used internally in combination with the class AnimatedImage in order to create and add an animation to our application.

1.21 Models

The "models" package contains three models needed for serializing and deserializing JSON responses from the server. The three models we have so far are:

- ApiResponse
- GameState
- MultiplayerGame

1.22 Multiplayer

Handles all requests/responses from Heroku API. Implementation of IMultiplayer interface.

1.23 SocketHandler

Implementation of ISocketHandler. Used by ChessGame to connect to Heroku API's Socket.IO websockets. Acts as a middleware class.

1.24 SocketEvent

Small helperclass to define Socket.IO message types.

2 Interfaces

Below we have a short description of all the main interfaces in the project, separated into subsections.

2.1 IBoard

An interface implemented by Board. Many of the methods are getters for attributes such as the board's dimensions, squares and their pieces, move history and player turn. IBoard also includes the methods responsible for moving the chess pieces correctly, as well as storing the move history.

2.2 IChessGame

An interface implemented by ChessGame. The methods are used for setting up a game, finding out if a game is finished, calculate and update player ratings, as well as finishing a game.

2.3 CheckerboardListener

Communicates between the GameScene and the checkerboard.

2.4 ChessGameListener

Communicates between the GameScene and ChessGame.

2.5 IPiece

An interface implemented by all the piece classes (AbstractPiece, King, Queen, Bishop, Knight, Rook, and Pawn). It includes getters for the chess pieces' attributes, such as their color and a list of their legal moves. Other methods are responsible for identifying and capturing each piece's enemies, moving logic and putting each piece in or out of play.

2.6 AI

Interface used by all AI. It contains three functions, returning the AI's move, rating and color.

2.7 Playable

Interface used by all AI. It contains one function that returns the move the AI will preform.

2.8 IDatabase

Interface used by the Database class. Specifices all the methods needed for highscore listing, etc.

2.9 HerokuService

Specifies and is required for all the endpoints to our Heroku API.

2.10 IMultiplayer

Interface used by the Multiplayer class. Specifices all the methods needed for listing, creating and joining games.

2.11 ISocketHandler

Interface used by the SocketHandler class. Specifices all the methods needed for websocket joining, emitting data, disconnections and such.

Relationships

The relations in the design model initially starts with the Chess class. The Chess class shows the first Screen, MainMenuScene, using the SceneManager and initialises the Database. MainMenuScene gets input from user and shows highscores/starts game accordingly.

When a game is started, SceneManager shows the GameScene given a GameInfo by the MainMenuScene. In the GameScene the ChessGame and Checkerboard gets initialized. The ChessGame creates a new Board given a Setup. The Setup creates the actual Board instance and creates all the Squares, containing x, y and chesspiece (which extend AbstractPiece). The Checkerboard gets input from user and requests move to the ChessGame class which responds using the CheckerboardListener. If the move was valid, it gets executed and the checkerboard gets updated accordingly.

The AI's also gets initialized in the ChessGame class and is used if needed.

Multiplayer gets initialized in the MainMenuScene and is used to list, create and join games through network.

SocketHandler is used in the ChessGame class and acts as a middleware between the server and the client to smoothen things out. It is used to communicate between clients during games.