SEG2105 Assign 5 – Group 53 – Mark Kasun (3806554), Patrick Shortt (6036229)

Problem Statement

The system will be responsible for handling a chess game played between two players.

Requirements

The system will include a chess board and all the pieces on said board. Each piece type will have certain valid move combinations. The player will have the ability to move pieces on the board when it is their turn. The system will be responsible for generating a graphical representation of the board for each player to see. In early versions, the program will generate a textual display on a command line interface, with options to expand to html or android interfaces in the future. The system will programmed in such a way as to make it easily modifiable to other variants of chess and even non-chess board games.

User Stories

Get Help: Upon starting either the server or client consoles, the users should be made aware of the "#help" command. This displays the list of valid commands and what they do to the user. Note that that the \docs directory containing the help documents may need to be moved to either the root or bin directory depending on compilation.

Start New Game: Either player must be able to start a new game and let another player join it. Upon starting the game, the user must decide who goes first or to randomly decide who goes first. The system will then generate the starting board and all necessary components.

Join New Game: A user must be able to join a game set up by another user. The system will check that no one else has already joined that game and connect the two users.

Move a Piece: A player may send a message of the form "#move <piece to be moved coordinate> <destination coordinate>" to the server, and immediately see the result on the newly updated board state. For this iteration of the game, a player may move any piece on the board to any square on the board. If the destination square contains a piece, it will be captured. If the coordinates sent are invalid, the user should be made aware. Note that there must be a space between the two coordinates.

Reset the Game: The game should be able to be reset by a player at any point. This will re-initialize the board state, and display it to all players.

Planned Expansion User Stories

Taking a Turn: Moves will only be allowed when it is the given player's turn.

Rules Enforcement: Moves made by a player that are determined by the system to be against the rules of chess will be rejected. The user will be given a chance to make a valid move on their turn. Checks, checkmates, etc. will be detected by the system.

Ending the Game: The user may end the game prematurely by resigning or offering a draw to the other player. In the event of a draw offer, the other player will be prompted to accept or deny the draw. The game will also naturally end when one player's King is placed in checkmate.

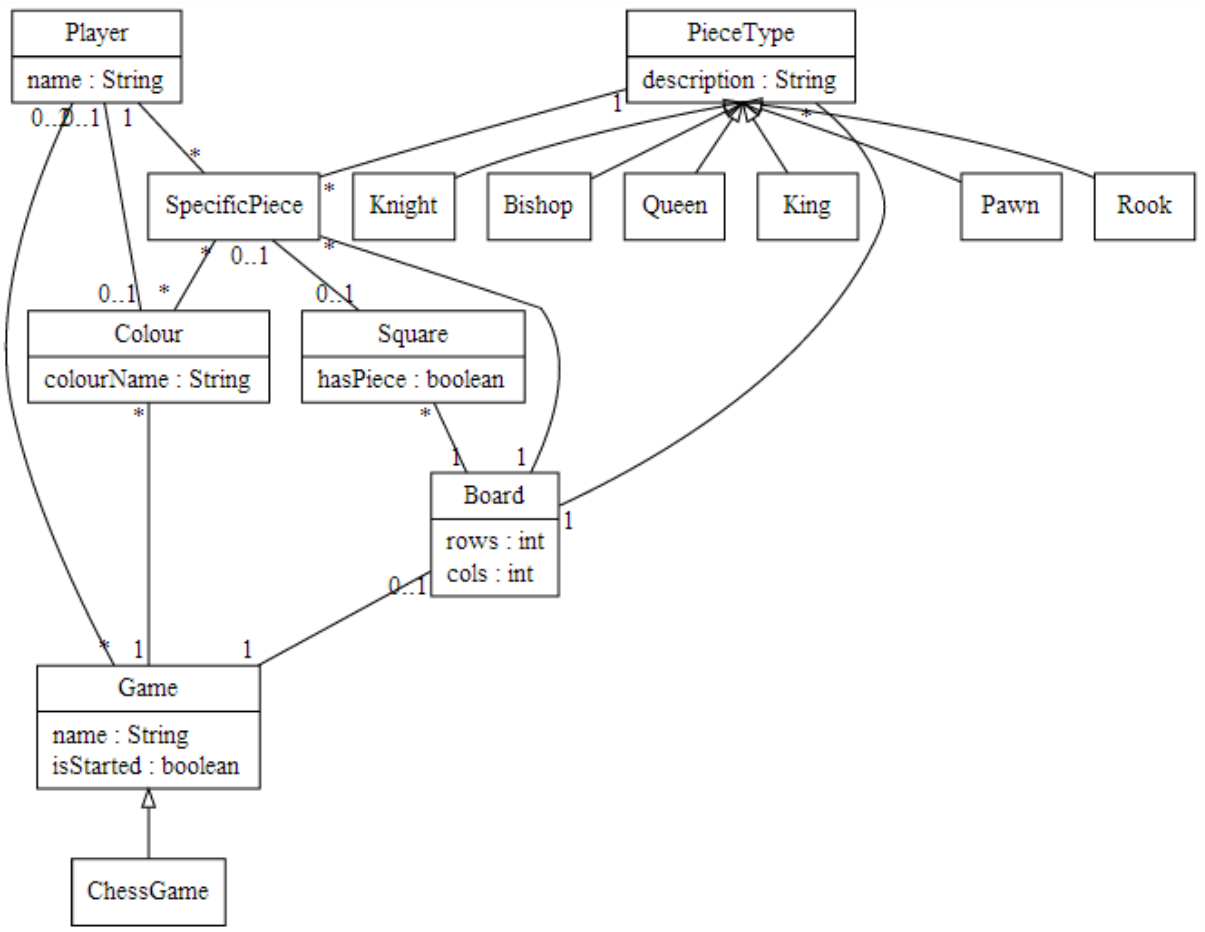
Architecture

For this chess application we will be designing it using Java. In early versions the game will be rendered using command line and text based representation. By using Java, we have the option to add an Android UI in future releases assuming time is available.

The server will be responsible for the bulk of the system. This is a thin-client system. The server will keep data on the current game state, determine validity of moves, and ensure users are unable to cheat while playing the game.

The client will be responsible for receiving commands from the user and transmitting them to the server. Such commands will be starting a new game, taking a turn, resigning, offering a draw, etc. The client will also be responsible for displaying the board state to the users.

UML Diagram



Example Messages between Server and Client

Client1: #help
Help documents are displayed to user.
Client1: #help chess
Chess specific help document is displayed to user.

Client1: #newchess <name>
Server: (to Client1 only) Game Created successfully; Waiting for other players.
Client2: #join <name>
Server: Client2 has joined <name>.

Client<1/2>: #start
Server: Generate board and send to clients.
Client1: #move 14 34
Server: Generate board and send to clients.
Client2: #move 69 40
Server: (to Client2 only) Move is outside board.
Client2: #move 60 40
Server: Generate board and send to clients.
Client1: #reset
Server: Generate board and send to clients.

NOTE: Several classes like MOVE/COLOUR/SQUARE are purposely designed to allow for adaptability and improvements to be made to the system. They are currently not used to their full potential in this release. Similarly, there are unused associations (eg., Player – SpecificPiece) that will be useful in further implementations but are not currently used.

STARTING THE SYSTEM:

Make sure you're in the /bin directory
Start server first (java server.GameServer)
Start client (java client.ClientConsole <id>)
Enjoy playing chess! Experience is better with two clients. #HELP gets you started.