JavaChess: An RMI Implementation of Chess

Gabe Wohlwend

Southern Illinois University

December 4, 2016

Chess is a game that has been played for around 1500 years. It has many layers of logic in it and has always been considered a game to test the wits and strategic ability of the player. Implementing this language with a Java JFrame based GUI is something I’ve been wanting to do for a long time and since this class is about distributed apps, this was an opportunity to finally start and finish the project.

I’ve packaged the program as two executable Jar files: JavaChessServer and JavaChessClient. The JavaChessServer can be run on any network enabled PC that has Java installed. The JavaChessClient needs to be in a directory with the picture files used but can be run simply by double clicking just the same as JavaChessServer. Both of these need to be run on PCs that have access to the same network. Once JavaChessServer is run, it will let the user know in the JFrame that comes up whether it was successful in hosting all 3 games. If this says “Hosting Successful” at the bottom, then the user can start up to 6 instances of JavaChessClient either on the same PC and/or on any PC connected to the same network. When the user starts up an instance of JavaChessClient, it will ask the user for the IP address of the server, the user will enter the IP address and click enter, if it fails to connect, it will let the user know in the prompt that it failed to connect and ask for the IP to be entered again. After it connects, this window disappears and a new one will appear listing the game sessions, how many players are in each session, whether it’s open to join with a green or red icon, and a button to join that will be disabled if the session is full. This info updates every 50 milliseconds. When the user clicks the “Join Game” button the Session window will disappear and the game board will pop up in another window, this has a board of buttons, a label at the top telling the player what to do to play the game, a bottom label that tells the player what color pieces the player control, and a “Quit” button that the player can click to exit the game at any time. The first player to enter the game will always get the white pieces. This player will be prompted to wait until the other player joins. Once both players have joined, the turn will be handed over to the white pieces. If it isn’t the player’s turn, the player will have to wait until the other player moves a piece. When it says it’s the player’s turn, the player will be prompted to select a piece to move, it the player selects a piece with no valid moves, the prompt will let the player know the piece has no moves and prompt the player to select a different piece. After the player selects a piece, the prompt will ask them to select a place to move the piece, only legal moves will be enabled to click. Once the piece is moved, the turn will be passed over to the other player. This cycle continues until one of two contingencies occurs: a player clicks the quit button, or either players’ king is taken by the other player. When either of these happens, both players are let know who won. After a second, the board disappears and the session window is brought back up, the game both players were in is reset, and it will show up as empty and open.

This program uses nine classes. Five in each .jar file as one of them is shared by both in order to use RMI. The ***JavaChessServer.jar*** file has the following classes packaged in with it: **ChessServer.java** which is the main method holding class for the server and is the JFrame the user sees when they run it. It also creates 3 RMI objects; **Piece.java** holds an X and Y position, and a String representing what kind of piece it is on the board, it also has methods to send and set data back and forth; **Board.java** holds a two dimensional array of *Piece* objects, variables for what player’s turn it is, whether the game is over, and what player won if so. This class also has more methods to send and set data; **GameSession.java** is the RMI object hosted by the server. It holds the occupancy of the game and a *Board* object. It has methods to communicate with the Client; **GameSessionInterface.java** is the interface defining the methods used in the *GameSession* RMI objects being hosted. This class is shared by both client and server. The ***JavaChessClient.jar*** file has the following classes packaged in with it: **ChessClient.java** which is the main method holding class for the client and is the JFrame where the user plays the game; **GameSessionInterface.java** as mentioned before is shared by the client program; **ButtonListener.java** is a class used to handle button operations in all three JFrames in the client application; **AddressFrame.java** which is the JFrame the user sees first that asks for IP info; **SessionFrame.java** which is the JFrame that shows the running game sessions.

This project took roughly 70 hours to complete and totals in at around 1800 lines of code. Almost half of these lines are move limitation logic, or simply cosmetic and were necessary to make the program stable. The first working build which didn’t have move limitations implemented was only 950 lines. In the future I would like to streamline the logic a bit more and lower the size of the code. Another shortcoming I would like to address in the future is the amount of RMI calls I make; this causes the program to operate with a bit of lag over Wi-Fi. The total design of the code is a bit haphazard and I didn’t do as much commenting as I would have liked to due to time constraints. In the future I would also like to implement a method that checks to see if the user is in check or checkmate and limit options accordingly. But for the moment, I am satisfied with the product I created in the time I had.