Chris Nelson

Java Project Report

For this java project, I decided to do a two-player checkers game through the terminal. The program displays the basic board with the pieces showing as either b or r (black or red) and the users enter which piece they want to move then which direction they want to move that piece. It allows the users to quit just by entering ‘exit’ and will reset after one of the players win and starts a new game.

The board itself is created using a matrix of integers. Originally, I was planning on using a matrix of checker piece objects, but figured out a way to simplify it was just numbers.

* 0 – empty space
* 1 – normal black piece
* 2 – normal red piece
* 3 – black king piece
* 4 – red king piece

By using a matrix, a user can enter the coordinates of what piece they’d like to move (row,column). I set up a couple of functions to make sure that only usable pieces can be selected. This means that I needed to check that the coordinates were usable and within range (1-8) and that the piece the user wanted to select could actually be moved in some way. This involved making sure that there is at least one empty space where the piece could be moved or an opponent’s piece that could be jumped to an open space.

Once piece is selected, the user then needs to enter which direction to move the piece, either left or right, or with kings pieces all four directions. Another function was created to make sure that the direction entered by the user for that piece was a legal move, if not they needed to enter another direction.

Moving to an empty spot just required me to update the matrix where the new space would be changed to the value of the piece moving and the old space was changed to a value of 0. I also needed to check that a piece on the side of the board did not move out of the matrix, since the program would crash if that happened. Again, this was handled when the user entered the direction.

When a piece would jump another piece, I needed to make sure that the piece being jumped was the other players and also that spaces being jumped to was empty. Once again the values of the new space needed to be within the range of the matrix. When a user enters what direction they want to move, they can enter a string of directions to allow for a piece to jump multiple pieces if possible.

Using a for loop to go through each of the characters of the string, I was able to check if the user entered l,r (normal pieces) or 1,2,3,4 (king pieces). Doing it this way, I could check the character and if it wasn’t a correct value, the program would just ignore it. This is all done within the move function, and if a piece was just moving to an empty space the function would do that then return a value of true and the loop would end. If the piece was attempting a jump, the move function would call the jump function. This function would have to piece jump, changing the values to represent the jump and return true. If a jump was not possible the jump function returns false. Doing it this way, I could do a multi-piece jump as long as the jump function was returning true, once false was return I knew there was no more jumps for that turn.

After a piece moved or jump, I check to see if it ended in either the first row for red or last row for black. If it did, I change its value to make it a king piece. This meant that instead of checking if there were possible moves or jumps for two directions, I would need to check for all four directions. Checking for the different directions was done through a series of if else statements and Boolean returns (true for possible move or move complete and false for not possible).

When it came to printing the board, I used two for loops with one in the other. This let me go column by column for each row. Depending on the value within that spot, the print function would display which piece it was or if it was an empty space. The print function needed to be called after each move or jump to show the new changes. I added a .5 second delay when printing the board to allow the users to see the move/jumps being made. This was added because after each move the screen is cleared and the new board is displayed.

Finally, to make sure the user could exit the program easily, I checked the coordinates or directions that were entered. If they entered ‘Exit’, the program would stop to two main loops it runs in (one runs the actual game and the other lets the users restart a game once someone wins).

Overall, the project wasn’t too difficult, just time consuming. If I had to write it over again I could probably make some of the code more efficient instead of checking each move through a series of if else statements. I also ran out of time in creating a computer that a player could play against. I could have made the computer find all the possible moves and determine the best one, then use a random number generator to cause the computer to make some mistakes and not constantly choice the best move.



Initial board layout at the start of a game

How to play:

1. Game randomly selects a player to start
2. Player must enter the coordinates of one of their movable pieces row,column
3. Once a piece is selected, player must enter the direction they want to move the piece
4. Multiple directions can be entered if there are multiple jumps that a piece can move
   * 1. l – left (normal)
     2. r – right (normal)
     3. 1 – up and left (king)
     4. 2 – up and right (king)
     5. 3 – down and left (king)
     6. 4 – down and right (king)
5. Piece will become kings once they reach the other side of the board, kings can move in all four directions
6. Game ends when one player no longer has any pieces left
7. Players can then restart the game or quit

* Players can quit game at any time by entering exit when prompted for coordinates or direction