Edinburgh Napier UNIVERSITY

Coursework Report

Dylan Tyrie-dron
40203045@napier.ac.uk
Edinburgh Napier University - Algorithms & Data Structures (SET09117)

1 Introduction

The aim of this report is to describe the implementation of a checkers game done in Java. There are many features that are used to make this implementation possible, same such features being:

- Movement
- Storing of pieces on a board
- The removal of pieces
- The storing of past moves, used for "undo", "redo" and "replay" tools
- An AI which was done using an algorithm to figure out which move to make next

Movement was done by moving pieces from one part of the board to the other using standard European checkers rules (ref). The pieces' positions were monitored by a coordinate class. The pieces were stored in a board using a 2D array so that the row and the column were stored for each piece. This allowed the pieces to be removed quite easily as you could set the board to be null(empty) at the specific piece position. Whilst pieces were moved during a player's turn there past and current positions were also stored so that features like replay were implemented. A deque was used to store these "past moves" to therefore play a replay from the start. The deque was also used to remove the previous move from the top of the list used in the redo fuction or just to look at the previous move at the top of the list which was used to implement the undo feature. The AI made use of all the movement that was previous introduced into the program whilst using a simple algorithm to decide which move to make during its turn.

2 Design

Many features in this program were designed through the use of algorithms and data structures. One such feature was movement, when moving a piece the user could see the potential moves(to an extent) that they could make during their turn as tiles on the board were highlighted based on the European checker ruleset(ref). These "potential moves" were stored in an array list. Although this was not the only thing that the array list was used for, it was used again for checking if another move was valid as the user "clicked" their first destination for the piece. The pieces were again stored in the list so that if another jump was available it was made possible to "double jump" which is valid under the European checker rules(ref).

Another feature that was implemented was the board array. The board was used to store the pieces, which in turn, stored all the piece positions on the board, their colour and their type (king or normal piece). To do this the board stored the pieces in a 2D array, this allowed the pieces to be stored in a row and a column. This also helped out in the painting of the board tiles as a loop was made to print all 64 tiles on the screen. another feature that was affected by this data structure was the removal of pieces. Which set the tile to null so that the piece was removed from the board. Although pieces were just set to null if they were being moved, the removal of "taken" pieces were stored in a variable temporarily to then be used for the undo feature.

One more feature implemented was the undo feature. Undo was used to give the user a way to undo moves that they decided weren't the best choice during their turn. This was done by storing the previous moves into a deque. A deque was used over a stack as it can be accessed at the start and at the end of the list which allowed the program to remove the previous move at the top of the list and to "peek" at the first item on the list to allow the program to replay moves from the start of the game. The piece positions and their destinations were stored for each move to direct the undo feature, when the undo button was clicked, to move the pieces back into their original place.

A feature which was related to the undo feature was the redo feature. This was done in the same way to the undo feature but in this case the piece was moved in the opposite direction from the undo feature. Instead of just looking at the top of the past moves list the top was removed so that the replay would not show the redoing of the same move.

Another feature related to the undo feature was the replay feature. The replay would reinitialize the game, setting all the pieces back to their starting positions. Then, it would access the previous moves deque from the bottom and "redo" every move again to show the user what moves occurred before their last.

An algorithm that was used in feature was the AI. The AI had to go through the same process as the user did when making a move. Therefore, the first step that the AI did was choosing an available piece to move, this was done at random. The next step was to move that piece

and again, a move was chosen at random. Though the process was basic, it allowed for a simple solution to the AI's "decisions".

Another algorithm that was used was the painting of the board. The painting of the board was hard to implement as for every second tile there was a different tile colour. But that was not the only problem, for every second line there was a different starting tile colour. Which meant that two loops were put in place one for every row and one for every column within the row. Then the tiles were painted differently for every second row and every second tile within the row using an if statement to split up the even numbered tiles from the odd, to allow for the painting of each tile.

A gui class was set up at the start of the coding process allow for visual testing. Due to past experience with programming in C-Sharp a gui was easy enough to understand and implement in Java. There was a mouse listener put in place to allow for the user to accurately move the pieces from tile to tile. Buttons were added for the undo, redo and replay features so that the user could easily use these features. A game loop was also set up within the same class which allowed the game to be played until it was stopped by the user. The "GridBagLayout" was used to the place the board and the buttons in suitable places.

3 Enhancements

This program is very powerful and allows the user to play the game of checkers within a computational device, but it is not perfect. A feature that this game does not implement is a "death-map" feature which would show the user a "map" of all the pieces that have been taken by an opponent or themselves when mousing over the replay button. Though this feature would have a fairly small impact to the game it would allow the game to be more complete.

Another feature that could have been implemented given more time would be sound effects. These sound effects would allow the game be more enjoyable. The effects would be played when moving a piece, taken a piece, when there is an invalid click or when there is a button pressed.

Another feature that would run alongside the sound effects feature would be animations. The animations would also allow the game to be more enjoyable. Animations could be used when âĂIJmousing overâĂİ a user's piece: enlarging the piece to make sure the user knows which pieces to use. Another animation could be used when a piece is moved to make the user fully aware which piece is being moved and allow them to strategize more easily for their next move by seeing a more visual representation. Another animation could be used to show the taking of pieces as the user can again easily see what is happening with their pieces and allow to prepare for their next move.

Another feature that could be implemented is a timer. This feature would be used to add a bit more competitive gameplay as the use of time puts the user under a bit more pressure to make a move faster. This feature also allows for another win condition, if there is a "stalemate" then time could be another factor that could settle the difference in a result of a game which could abolish the chance of a draw.

4 Critical evaluation

There are many features built into the game that are useful, such features include: the replay feature, the potential moves feature and the movement feature.

One limited feature is the redo feature, this feature is limited as it doesn't have as much use as the undo feature as in most cases when you undo move, you are undoing because you made a mistake and making two mistakes in a row is quite uncommon.

5 Personal evaluation

A lot was learnt over the past few weeks including: more Java, âĂIJguiâĂİ programming and the implementation of algorithms and data structures.