Algorithms and Data Structures Report

Lewis Hamilton
40270314@napier.ac.uk
Edinburgh Napier University – Algorithms and Data Structures(SET09117)

Introduction

The objective of this coursework is to create a checkers game in any programming language of choice and apply the use of appropriate data structures and algorithms for the game. The checkers game should allow the player to play against an AI competitor, the game should also record the moves made and allow the user to undo and redo their moves. The game could also have a replay feature, where all the moves made in the game will be replayed. The game will have to played in a text based interface however a GUI could be used if needed.

Design Approach

Language of Choice

Python was chosen as for the programming language. This language was as it allows to make for very readable code and is also quite a simple language to learn to use. Python uses some simple data structures such as lists however data structures like stacks will need to be created from scratch.

Board and Player Movements

The first part that had to be implemented into the game was the board. To implement the board and allow the user to move their pieces on it was to use a 2d array to store the values for the board, each index in the array would store what was in that position on the board i.e. board[0][1] would store the values of the board position on column 0 , row 2. The size of the checkers board is 8 by 8 so the 2d array would need to have to be 8 long and 8 high. To populate the board it was decided to make a function so the board can be re-populated if the player wants to play the game again once it is over. To do this computer would need to loop through the array and check what row and column it is in, if it is in the top 3 rows (0-2) it will need to add the counters for the AI on these rows, it does that by checking what row it is on and to determine where the pieces will need to be added. If it's the middle two rows(3-4) only add free spaces and then if it is the last three rows(5-7) put the player pieces in the correct places in a similar way he AI pieces where added. The AI counters are represented with "B" counters, free spaces are represented with "|" and player counters are represented with "R". If the player wants to move a piece on the board they will enter the row and column by typing the row then a comma and then the column. This position that has been entered will then be checked to see if there is a player counter in that position and will check if it can be moved. If it is a valid position then the player will have to enter the position on the board that they want to move it to. That position will be checked and if a valid move it will be done, this is done by overwriting the pieces previous position with a free space and the new position with the player counter. To deal with the players inputs a separate class was created to deal with the players moves and manipulate the board accordingly. To check if a piece could be moved first it would get the coordinate entered by the player and put the values it got into variables called row and column to easily access the position on the board. It will then check if the row above its position and if the column on the left and to the right of the current position are free. If either position is free then

allow the user to enter their move they will like to take. The new position that is entered will also need to be checked to make sure that it is a valid position, this is done by making sure the new position is exactly one row above the previous row and one column to left or right from the previous position, it also checks that new position is on the board and if the new position is a free space, if all these conditions are met then the piece is moved and the board will be changed accordingly.

Jumps

If the new position on the board has an AI counter that position will need to be checked if a jump can be made. This is done by first checking if the piece is within the boards range. It then checks if the row above the AI counter and the column opposite it is free, if these conditions are met then the AI counter is overwritten with a free space and new position is overwritten with a new piece. For double jumps it checks if there are AI counters that can possibly be jumped and then will check them the same way as before, it will keep doing this until it cant.

Kinged Pieces

When kinging a piece every time a normal piece is moved it will check it is at the other side of the board, when it is it will overwrite its new position with a kinged piece. When moving a kinged piece it needs to check if above the piece and below the piece has free positions it can move to, they also can only do one jump over the competitors pieces rather than having more like regular pieces.

AI Control

For the AI I decided to use my own method for the AI to select pieces to move and decide if it wants to jump the player. The AI first checks the board for valid moves and puts them into one of two lists. One list that stores coordinates of valid moves to free spaces. The other list stores the coordinates of pieces that can be jumped. First it selects a random coordinate from that list containing only jumps if it isn't empty, so it can take a player's piece. If it can't perform that move, then it just picks a random piece to move from the other list. This should make for somewhat competitive AI but also keeps it simple.

Undo's/Redo's

For Undo's and redo's, it was decided to use a stack for both. Before a move is made the current boards state is pushed into the undo stack. Whenever the player performs an undo the current state of the board is pushed into the redo stack and the board changes to the board's state that gets popped form the undo stack. When performing a redo, the current boards state is pushed to the undo stack and the board changes to the board that is popped from the redo stack.

Version Control

It was decided to implement some sort of version control to easily access older versions of the code in case there was a problem with the code and need to fall back on older code that works to an extent. Ii was decided to use push the git repository to Github as it seemed like the best service to use.

Enhancements

There are many aspects of the program that can be enhanced. The AI could have been a lot more intelligent if a proper algorithm was used such as the minimax tree algorithm. This is a recursive algorithm that would assign a value to all the available moves and then pick the move that has the best possible outcome. A replay feature could have been added in the game, so the player can view the game they just played and see all the moves that were played during the game, this could be implemented by using a stack that can be popped at the end of the game. The game could have used a GUI to make it look better and make its functionality better as it would be easier to play the game and be more intuitive, colours or Unicode characters could also have been used to represent the counters. The game could also have had the option to play against a human player as it would have been easy to implement, this could have been done by improving the player class by changing the way it reads the board depending on what player it is i.e. player 2 would use the "B" counter and move down the board instead of using the "R" counter and move up board like player 1 would.

Critical Evaluation

There are many features in my game that work well and many that could be improved. The player movement through the board works quite well as it only allows valid moves to be made and will automatically do double jumps if it can do them. The undo and redo feature in the game works well and does what was intended, it easily allows the user to do undo's and redo's and tells the user when they can no longer do undo's or redo's. The way the user inputs the moves they want to make is easy to understand but can be hard to get used to, this could be improved by instead doing row and column of the piece maybe change the coordinates to an x and y axis or even just change the range from 0-7 to 1-8 as it is easier to see in your head when you want to select a move, however this isn't a huge issue as the board is shown to the user with values of the rows and columns of the board, on the left of the board and top of the board. The way the board is easily re-populated when the player wants to play a new game is implemented well, however this function could easily be improved upon by allowing different sized boards and adding the correct number of pieces to the board when the board is upscaled. The AI could have been a lot better however the AI player that is currently implemented can play against the human player and keep to the rules of the game and will try to actively capture the players counters. There are some bugs with the AI that sometimes occur like sometimes the AI will fail to do double jumps and will also will try to jump a player's counter but will fail in doing so (this isn't a huge deal as the AI will just select another counter to move). And due to the AI's randomness, it sometimes makes for uncompetitive AI that will do dumb moves. Instead of making the player type "move" when they want to operate a move, the program should just accept a coordinate thrown at it as this would have reduced the number of steps it takes to enter a move. Error handling in the game works well and helps make the game more robust as it stops the player from entering possible data that could crash the program. The look of the game on a text based interface isn't bad and it also has instructions at the beginning of the game to help the player navigate their pieces on the board.

Personal Evaluation

Reflecting on how I performed on this coursework, I believe I performed well in most aspects however there are a few areas that could do with improvement. This was the first time I used python properly and I feel I done well at adapting to the language and got to grips with it quite quickly, however due to not using it before some things that should have been easy to implement ended up being quite difficult. One issue I had due to not fully understanding python was when I came to implementing stacks, Due to python not having a stack library built in to the language I had

to make my own Stack class for the program, this wasn't difficult to do as I had already written a stack class for python in one of the practical classes. However, when it came to pass the board into the stack, I realised that whenever I went to do an undo, it would pop the correct amount of the stack however the all the data coming off the stack was the same as the current state of the board. I realised that the data inside the stack was being changed after a move was done even though it shouldn't be. I soon realized that the data from the board wasn't being passed into the stack but that it was being passed into the stack by reference. This meant that the memory location of the board was being passed in instead of the actual data, this allowed it to easily be changed whenever the board got changed in another part of the program. This was easily fixed as python has built in libraries to allow the user to pass the variables data instead of passing them by reference. Another problem that was faced was making the AI, I decided that I would make my own AI instead of using an algorithm that already exists. I feel like this was a bad decision because if I had used a pre-existing algorithm for this then I would have learnt more from the coursework and would have allowed me to use more advanced data structures. It was the first time I used a git repository/Github for version control, I learnt a lot from using this and was very useful whenever I needed to look at older solutions to help me fix bugs.