Checkers Report

Chris Reid
40202859@live.napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures (SET091117)

Abstract

This report details the actions taken when creating a console checkers game with optional computer opponents, using appropriate data structures and algorithms. The program was created using Python.

1 Introduction

The main aim of this project was to display an understanding of algorithms and data structures learned in this module by creating a checkers game, in a preferred language, that functions in the console. As previously mentioned the language chosen was Python.

1.1 Features

As well as a functioning checkers game, several advanced features were required, which are detailed below

1.1.1 Players

A user is able to choose how many human players will be included in the game. The game can be played between two users, a user and a computer, or between two computer.

1.1.2 Recordings

Every game records play history. The play history consists of the piece selected and where it was moved to. The moves are stored in order, starting with whites piece, followed by the coordinates where the piece was moved and then directly to the black piece. Play history gets stored into a text file that can then be used to load in and watch previously played games.

1.1.3 Undo and redo

Users have the ability to undo or redo any of the moves they have made. When the user selects undo, the board gets reverted to the state it was in prior to the last move made. Similarly, if an undo has been executed, the user has the ability to redo. The redo simply replaces the undo. If there are two users playing, they can undo completely back to the beginning board state an back.

1.1.4 AI

One of the challenges was to implement an AI opponent feature that can play against a human user, or another AI opponent.

2 Design

The main focus when designing and architecting the checkers game was ensuring that the best data structures and algorithms were chosen and used for each feature of the game.

2.1 The board and pieces

When designing the game, the main focus was identifying suitable data structures for the board, pieces positions and moves. After considering different data structures, a 2D list was chosen to store the board. The board is declared already populated with the pieces in their correct starting positions. Each piece is represented by a string within the 2D list (e.g 'w' or 'b)'. When a piece is crowned, it becomes the uppercase of the teams letter. Pieces locations on the board are their indexes within the list.

Before the teams turn, each of that teams pieces get tested for available moves. Each piece with one or more moves gets their index added to a list called 'available moves'. Each piece is also tested to see if they have any moves which involve taking an enemies piece. If so, they get added to a separate list called ‘available takes‘. If 'available takes' is not empty, it gets displayed to the user at the beginning of their turn - forcing them to capture any pieces. If it is empty, the 'available moves' gets displayed. To find valid moves, a pattern with the indexes was used. An example is, if it is an uncrowned white piece, it moves down the board. Therefore, the row plus one and the column plus or minus one is checked for being empty (A further example is shown below).

```
if board[row+1][col+1] == '-' or board[row+1][col-1] == '-' then
    Moves are available
end
```

This is the same for moving up the board, only the row the piece is on is row minus one. Validation on the rows and columns are added to prevent the user from being able to move outside the board. Similarly, to find pieces that can take an opponents piece, instead of an checking for an empty square, the square is checked for an enemies piece. Also, the piece in the next diagonal square is checked to ensure it is empty.

Crowned pieces simply inherit the functionality of the opponents movements in addition to their own teams movements.

2.2 Recordings and Replaying

For recording the game history a simple list is used. After each player chooses a piece, the index is appended to the list. Following that, when player finished the move, the index of the square they moved to is appended to the same list. This continues throughout the game until it is complete. When the game is finished, the list gets written to a text file. When the user wishes to watch a previously played game, they select the watch mode when the game is first executed. Each move is read from the file into a list, which then gets fed into the game, allowing the user to step through each move made.

2.3 Undo and Redo

The undo feature was created using a stack. The board game state is appended to the stack after each turn. When the user executes the undo, the last move gets popped from the stack and the board gets returned to the last element in the stack.

The redo feature uses was created using a deque. Each state that gets popped from the undo stack gets pushed to the the deque. When the user executes the redo, the first state gets popped from the beginning of the deque (leftpop()) and the board gets reset to the first element in the deque.

2.4 Computer Opponents

The computer opponent simply selects a random element from the list of pieces with moves available, which gets passed at the start of their move (To select a random element, the random module was imported). A separate list is created for pieces that can take an opponent. If the list with of pieces that can take is not empty, it gets sent to the to the computer instead of the list with all pieces that have moves available. This ensures that the computer takes a piece whenever possible.

Following that, another list is created that stores all the possible locations of where the selected piece can go. Again, the computer randomly selects from this list and moves the piece to that location. If the selected piece can take an opponents, the list of locations only contains the the squares which force the computer to take that piece.

3 Enhancements

Although, in the games current state, it fully functions and users can partake in or watch games of checkers being played, there are many enhancements that would improve the game greatly.

One feature that was being considered - before time became an issue - was improving the appearance of the game. Currently, differentiating between pieces is not as easy as it could be. One improvement to help make each teams pieces stand out is to use a module called 'Crayons'. By using 'Crayons', each teams pieces can have the font colour changed. This is quite a small enhancement but would greatly improve the game.

An enhancement that would be a drastic improvement would be to change how the AI opponents work. The original idea for the computer opponents was to implement the 'Minimax' algorithm with a heuristic function. This would make the game much more challenging than it currently is, improving the game-play for the user.

Another enhancement that was initially planned, but never implemented due to time issues, was to change one axis of the board to letters (changing the pieces locations from '10' to '1A'. This would simplify the game and prevent any confusion or user error when selecting pieces or locations.

4 Critical Evaluation

Although there was a great deal of challenges involved in this project - and quite a few proposed enhancements - the game turned out reasonably well. When the enhancements get implemented, it will be a well rounded game, with some great features.

The most useful feature in this checkers game is the lists that get displayed to the user before each turn. The user gets shown which pieces have available moves, and when they choose one of those pieces, a list containing all the squares they can move that piece to is displayed. This reduces the chances of users choosing pieces they cannot move or trying to move pieces into invalid spaces; which greatly improves efficiency.

The feature that is lacking the most is definitely the computer opponent. It works adequately, but with more research and an AI algorithm implemented (Minimax), it would be improved significantly. Researching and planning before hand would have made the implementation of an algorithm much easier.

The way in which the board and it's axis is displayed is not ideal and can cause some confusion at times. This is due to the fact both are simply displayed as integers from 0 to 7. A user can easily make the mistake of entering the row before the column as input. Although validation prevents this from causing any problems and the user is re-prompted for their input, it can become quite frustrating.

All other problems are relatively small and can be fixed or improved with ease, however a lack of time has prevented it. One being how information is displayed to the user. On some occasions text being displayed to the user is shown before the board is updated, which can push the information out of view.

5 Personal Evaluation

Creating the checkers game was very challenging, yet enjoyable. The most important lesson learned is how important planning is. There was planning at the beginning of the project, but not a sufficient amount. This became clear in the later stages of the project and it shows in some parts of the code.

One of the biggest challenges came from over complicating things for a small feature and not considering how changes will affect other aspects in the future. For example, the board is stored in a 2D list with the pieces being accessed from their indexes. To access these, the row is searched first, then the column. The thought process was that it would be an improvement to have the column entered before the row, so the users input gets reversed before being used. For such a small change, the problems that arose was quite surprising. This also increased the amount of time that was spent debugging greatly.

As previously mentioned, a lack of planning created quite a few problems. Features were implemented without considering what way future features were to be added. Implementing a feature, then moving onto the next, only to realize there was another way that would work better with both features happened quite frequently. This is a challenge that was not really overcome during this project, but has definitely been a valuable lesson learned.

Overall, personal performance during this assignment was fairly good. Every feature that was required got added. The way in which some of them were added is not ideal, but they function. The performance improved noticeably towards the end of the project when more experience was gained.