# Algorithms and Data Structures Report

Louise Skinner
40270316
Edinburgh Napier University - Algorithms and Data Structures(SET09117)

## 1 Introduction

To deepen the knowledge and demonstrate an understanding of Data Structures and Algorithms, a task was given to create a checkers game that would implement certain data structures to be used within the game. The data structures can be used to create the players; the positions and the board itself within the game.

The project is to create a checkers program that implement certain data structures and that will allow a user to play against either another user or to play against the computer or even allow two computers to play against each other. Through the use of data structures, the user can store their latest move and when they so wish they can undo/redo their moves or when the user has taken their opponents they can view that move in a replay.

## 2 Design

Within the game, the board is stored within a list more specifically a 2-Dimensional list. As the user plays the game, the new position and the old position overwrite the existing value. The use of the data structure allows the user to call the index of the 2D list as this particular data structure has depth unlike the other data structures available. Using the 2D list is essentially a list of many lists, allowing the user to view the data structure as a grid which imitates the checkers board. This data structure is perfect for game boards as each item of data within the list represents a column and every list represents a row. Data is constantly being updated and positions are being checked for the checkers game at random position within the list. Both stacks and queues, data would need to be removed from the given data structure to change that position and then data would need to be re added to the end. With the list, data will not be removed or added to the board. The data at given position is simply being overwritten even when a piece has been removed, the board will stay the same size no matter what happens during the game. A list allows the size to remain the same from the start unlike the other data structures where they fluctuate in size.

Using the 2-Dimensional list, each player is stored within a different section of the many lists. As the list contains many sections of that list and is represented by a B or W. When the user wishes to move position, they call the position of their player and then indicate in which new position they would like to save it, overwriting the new and last positions.

Many features that are used with the game are implemented through the use of a stack. When the user wishes to undo a move, the last value is removed. With the stack the user can: replay certain moves and watch as it displays the board before and after the move has been made. To redo a move, a stack has also been implemented to add the existing data to the stack.

Each move that the user makes is stored into a stack this allows the user to either replay certain moves, undo their last move or to redo the move. Each player has their own stack where their moves are saved to so that when calling the players there is no overwriting or misunderstanding of the points. A stack makes a great use of removing and adding items from the end of the positions. Due to this feature, the stack was implemented during the game as replay, undo and redo need access to the last items within the list and can easily export them unlike queues where they need to remove all the data to get the last item. For example, when the user wishes to undo their move, the last item is saved to a smaller stack in case they wish to redo their move. After the move has been saved, the last item of the 'Users Moves' is then 'popped' - removed - from the stack. As stacks deal mainly with the last item no other data is being effected with the rest of the stack. When the user wishes to redo their move, the last item in the smaller stack is saved to the larger stack for the players moves. The item in the smaller stack is removed, saving space.

## 3 Enhancements

In any program there is always changes being made whether they're new features or certain features being fixed. If there was more time was granted to complete the checkers project, there would be certain areas that would have been added and certain sections that would have been improved.

The first item that would have been added is adding a double jump feature. It would have allowed the user to jump over multiple pieces and steal many of their opponents features as many checkers games include this feature. The original design did include a double jump but due to a time constraint it was removed as more time was spent on other sections. This is the only feature that had been left out of the game. Though more researching would be needed as many controversy on how this feature works.

Through further research and more amount of time, the AI player may have had a difficulty mode and an easy mode. Very much similar to the feature that the user is allowed to pick their opponent. Currently the AI is very simple as it randomizes the positions and will only move within the boundaries that has been set. It is a very easy mode as the AI just picks a random route and selects the new position along with it. In researching AI players, there was a better solution found that would choose the best route to take even taking in many factors. The method for the AI choosing the best route is called 'Alpha Beta Pruning'.

Since the AI vs AI is a simple algorithm rather than Alpha Beta Pruning, it simply finds all the moves it can make and adds it to a list, the game is very long and uses up a lot of space as it only moves to legal spaces and once an opponent is close to the AI, it will then jump. It does not have a purpose to look for the better path to take. If there was more time then, the simple AI would be altered so that the game would end as currently many items are randomized and takes quiet some time determine a winner.

Also, the AI vs AI simply runs from start to finish without any user interaction which can be quiet hard to follow the actions that are occurring. If there was more time added, the user would be allowed to view each section with the user pressing a certain key to view the next screen.

In the beginning of the production of the checkers game, the user was allowed to select which side they would like to play either the Black or White. As the development continued, there was an issue around that feature as later on the development the program was set in many ways which would have caused many errors within the program. In doing so that feature had to be removed. If there was more time, the feature would come back and the errors would be dealt with.

If there more time, a consideration that the checkers could be updated to a GUI interface rather than a simple console game. It would be much easier to see the board. Clearly understanding which spots were taking and which positions pieces were in. As currently, the game is quiet therefore being quiet harder to see the positions. Adding a GUI interface would eliminate a large portion of input validation as, the user only be clicking their piece and the section they would move to.

As much as there were features that could have been added, there are improvements that could be made to the program. Firstly, the board could have been made into a class and would of had methods to create the game instead of many pieces as it made the code look cluttered and messy; possibly confusing to those who aren't part of the development.

Secondly, as much as there is input validation, there simply not enough to cope with the many actions that user could input. Adding thorougher validation would improve the experience dramatically as the decrease as the game breaking would not occur at all.

Though, the game has an undo feature, it will only undo the last move made. As an added feature, the user would select how many times they would like to go back and then start from that point. This feature would have been included if there was an extended time period. As undo and redo run in parallel with one another, this feature would also be changed in the same way as undo. The undo will not work with king pieces only regular pieces, if more time was granted. This feature will be fixed to include king pieces as well.

## 4 Critical Evaluation

The AI feature has many sections like finding moves; stealing an opponents piece and moving a king piece. All these feature work very well as the AI will search for an open spot and move to that open section. When the AI approaches an opponent, it will choose to steal the opponents piece as opponents are set as a higher priority than moving a square. The AI will detect when it reaches the top of the grid and will change the piece to a king then will follow all the rules of a king piece.

Although the AI works very well at moving it will run on a loop as it is not clever enough to find the best path so the program just runs without stopping. This takes a lot of time and a lot of space as all the moves are saved to as a stack. The game will take a long time running before a winner can be determined.

The AI feature has its own undo feature to prevent the program from breaking. This feature works well as if there is no more available moves in the stack, it will remove the last move made to the AI and then will remove the option from the list and then try to pick a new move as that option is not the best case. If there was no more moves to make and there is still pieces on the board the undo feature will be used to go back a position and then try to continue the game from there. This feature is not perfect as it has not been properly tested. Although, the program has ran to test the feature there has not been a clear indication that this feature works to its full potential. Both AI and human contain an undo feature but they slightly different on how they effect the game. The undo functions works brilliantly on the user to undo their last move and retry at that point. This function works to an extent but the user can not undo a certain amount of moves, only one last move otherwise it would break. As aforementioned it would be a feature that would later be added to the program at a later stage. If the user wishes to undo a move but this particular steal their opponents piece, there will be an error message that appears saying that they are unable to do undo then it will go straight to Player 2's turn.

The program contains a redo function which allows the user to only call a move if they have called the undo function. This small feature utilizes the removing and adding onto different stacks to return the value back to a last saved state. This feature parallels alongside with the undo as that it can only redo the last moved and not redo from a state the user wishes to.

The program must steal opponents so that the game can complete. This feature has been executed well as the the regular pieces can steal both the opponents and also steal the pieces that are also the king piece. It very much the same with the king pieces but they can steal opponents that are in all directions. This feature works well with the user and the AI. With both opponents the program will check if its valid and remove the opponents by overwriting the piece to a free position.

The user selecting their piece needs a lot of input validation added to it to prevent the user selecting a coordinate that is not their piece. This function works to some extent as the user is not allowed to select the other users piece however the program does accept the free positions represented by an underscore and the invalid positions shown as a pipe (vertical bar) as valid positions and would continue the program with them as acceptable pieces. Although, selecting its new position will only allow the user to move if the spot is free otherwise it will bring up an error message and ask the user to retry until they select the correct spot.

On a whole, the program works well despite minor issues. The program represents a checkers game where pieces can only move in only a diagonal direction and each piece is able to jump their opponent. The game allows the player to select which game they would like to play: against a friend; against the computer; or allow to computers to play against each other. The replay, undo and redo all work and allow the user to maniplute the game.

## 5 Personal Evaluation

Through the development of the project there was many obstacles and challenges that had appeared and many learning opportunities. The first challenge was to learn the rules and guidelines of checkers. It had never been played before so learning the rules and the learning the logic behind each moves the pieces made was very important to the development. To learn how to play checker and understand the moves, repeated playing checkers and noting down the rules and exceptions that have to be implemented within the project helped overcome that obstacle. In the end, the project resembles a game of checkers with the pieces only moving to a legal positions.

During the implementation stage where the user wishes to steal an opponents piece, the logic behind was quiet difficult to understand. Spending some periods of time trying to understand how to implement stealing the opponents piece. To overcome this obstacle, drawing the grid and placing the pieces on the grid to where they start to where they will end up helped greatly. Even still not understanding the logic even though, drawing helped the understanding a lot, talking to fellow peers who might understand the logic, benefited as ideas could pop up that could be adapted to the environment in which the task was written for. Through drawing the pieces out, it became much easier to visualize the pieces and the positions. This method was implemented to help visualizing, the opponents and the king pieces as they all have different ways in which they move on the grid. Talking to fellow peers helped create an understanding of many ways that could tackle the problem.

In the beginning, it was very challenging to select which programming language to use and also how to start the project. Looking at many ways to tackle the project either console or GUI. After many days, of researching the many possibilities that the program could take, after testing out many scenarios, the best option that was go with the easiest way to understand the logic. With the selected choice, the project was created through the console as it would fit within the timescale.

In the beginning of the project, creating an AI Player seemed difficult to implement even after researching many algorithms the task seemed very daunting. Even understanding the logic of the complex algorithms it still seemed very difficult to implement. After talking to many fellow peers and learned their many ways of implementing a simple AI, it was much easier to convert their ideas to fit the environment of the task and write two sections were an AI is playing a game of checkers. Thinking back, it seemed easier to write for the AI player than the human version as there are less data validation aspects to think about.