Algorithms and Data structures Coursework Report

Jonathan Mitchell
40311730@live.napier.ac.uk
Edinburgh Napier University - Algorithms & Data Structures (SET09117)

Abstract

hello.

Keywords – Draughts, Algorithms, Data Structures, Console

1 Introduction

Create the game Draughts The objective of this report is to create the game draughts in a chosen language and a chosen format - console/form/WPF or similar. Particular attention is to be placed on the Data Structures and Algorithms used.

The game, at the very least, should allow for 2 human players to play against each other. Adding "AI" algorithms to allow for Player vs Computer, or even Computer vs Computer are optional additions. Based on development capabilities within the Project time-scale. Additional features such as undo or redo a move are also beneficial.

Adding the ability to record the results of games and even record games themselves is another possibility. Finally Creative freedom is encourage during the development. An individual uniqueness is preferred when developing the game of Draughts.

The application associated with this report was created using a console application. Written in c#

2 Design

2.1 Player vs Player

This began with creating a simple game board. Populated with elements from string array tiles. Allowing the player the ability to select the co-ordinates that will be populated with the draughts counters. Creating the concept of moving across the board was based off a Noughts & Crosses game developed beforehand. The data structures used are the following:

- Arrays: Used for:
  - Creating the elements for the board.
  - Getting the letter & number of end position after successfully capturing an opponents piece.
- Stacks: Used for the undo & redo feature
- Lists: Used for computer AI. Stores the starting locations of all pieces on the board. The idea was to replace the element value with the new position of a piece, after it has been moved successfully. Also items are removed when they have been captured by the challenging player or computer. Lists were chosen for this as they are memory-dynamic data structures in nature and adding or removing elements is done without the need to instantiate new lists of various sizes. A do while loop is used to keep the cycle of player 1 moves -> player 2 moves etc. until either players pieces is 0. The game then ends displaying the winner. The application then closes or goes back to main menu?

Broken down into stages, the design was created:

- movement
- preventing sideways movement
- preventing backwards movement
- Limiting forward movement to only available diagonal positions
- Checking for opponent marker
- Capturing Opponent marker if possible
- Checking if there is a second opponent marker can be taken

The following classes were created:

- Board
- Movement
- PlayerA
- PlayerB
- Error
- Redo/Undo
- Skynet
- Hal

The PlayerA, PlayerB, Skynet & Hal classes all inherit from Movement class, This is due to the fact hey all share similar functions and use the same variables.

Originally, all code created was first developed within the Board class. After testing to ensure it performs correctly. The code was split up into functions onto moved to the relevant class.

For loops, with IF statements, were used to validate movement possible: only forward diagonal possible. Checking for an enemy piece, capturing it, if possible and

checking to capture a second piece were all moved into separate functions.

Undo/Redo was implemented using stacks . This allowed simple push/pop function to add/remove player move. By pushing the move input onto the undo stack. It can be popped to be used to reset the player's move. Subsequently, pushing it to the Redo Stacks if the player chooses to redo the undone move.

2.1.1 Player 1

Player 1 development began by implementing each section of design and testing to ensure it works. First thing is to ask the user to input the starting coords of a player piece and the end coords - where the player wants the player piece to go. These are first checked to ensure they are not blank. After this both string are inserted into respective char arrays, For later validation and used to determinf ifcapturing opponents pieces is possible.

The next level of validation is to check the starting coords are in the array of the checker board. If not tells the user If they are present. The endcoords are then check to ensure they are in the array tiles and this location does not already contain a player piece.

Preventing sideways movement was the easiest to do - simply checking if the letter from the inputed endcoords does not match the starting letter coord by using the first index of the char arrays of both start coords and endcoords. If this passes backwards validation is next.

Backwards validation is done by checking when the value of start coords equals the value of letter array in a for loop. the value of the int, used in the for loop, at this point is then stored. The same is done with the endcoord. The values of these ints are then compared. If the value for the endcoord is less than the startcoord means the endcoord is a coord that is behind the startcoord. the function returns false as backwards move is not a legal move for standard pieces. Else returns true. The movement is not backwards. As sideways movement has also been validated. All that is left is forward movement.

Forward movement is firstly validated by comparing the first index of each char array for startcoord & endcoord. If the destination value is +1 of the start coord then it is forward movement by one row. Anything else assumes it is movement forward by more than one row. Which would not be a valid move. Then the number, second index value, is checked. If the destination coord is +/- 1 of the start coord. It is the forward diagonal of the start coord. Anything else is invalid.

After this, the player piece is moved from the startcoord to the destination coord. An option to undo this move is made available. The undo process is detailed later in this report. The player turn variable is +/- 1 to allow the second player to move a piece.

2.1.2 Capture opponent piece

If there is an opponents piece detected in the destination coord for player piece. Then the capture piece function occurs:

First thing to occur is finding the new destination coords of the forward diagonal AFTER the current destination, based on the position of the starting coord of the player piece. This is done by the checkEnemyMoveToCapture algorithm. SEE Personal evaluation for description.

The new destination coords are checked to see if the exist AND there is no piece belonging to either current player or opponent player. If this condition is met. The player piece is moved, similar to moving a player piece where an opponents piece is not present. The only addition to this is the opponents piece is removed from the board and the opponents player piece count is reduced by one. This is done automatically, and no undo feature is available, presently. Else, the move is not possible and nothing happens. See below:

2.1.3 Capture Second Opponent Piece

If an opponents piece has been taken, the variable for player piece start coord is set to the new destination coord from taking an opponents piece. And two new coords are looked for: the left and right forward of the new start coord. These are then checked to see if they exist on the board AND they contain an opponent piece. If yes then an attempt is made to capture a second opponent piece. This is done exactly the same as capturing the first piece. Moving the player piece further down the board. This is also done automatically. With no input from the user to confirm this move. There is no option to undo this move. Else a second piece is not captured and the opponent player makes their turn.

2.1.4 King Pieces

2.1.5 Player 2

Player 2 development is pretty much the same as player 1. Only certain algorithm arithmetic was changed around: + becomes - & - becomes +

2.2 Player vs Computer

2.2.1 Computer

Computer vs Computer

2.3 Undo/Redo

This section of the game consisted of a class that stores stacks for start/end coords of piece moved. To either undo/redo a move. Only implemented for movement and taking a single opponents piece.

2.4 Enhancements

Firstly. The game development would be done in WPF. Due to the graphic capabilities of WPF, a much better visual representation of the game would be possible. The board, itself, could be created using the Grid & Grid definition in XAML. Each grid spot, 64 in total, could be assigned an index of the player board array. Placing the player pieces, as separate objects, and assigning them to the current array index they are placed upon. Using click event for the mouse, the player can then click each piece and click the destination.

Background code would check they are clicking the correct player piece, based on player turn, and validating whether the move is legal or not. This code would be very similar to that developed. The only real difference would be the position would be based on the array index and not a typed coordinate.

Also, buttons can be implemented for Undo/Redo functions etc. Without the need to ask the user, removing a slight annoyance, they can then just click on the button to perform this action.

WPF also displays the game in a GUI style. Which is more desirable, in the "Modern Age", than a console application.

Second. Sticking to the console application. The array design would be changed. Instead of displaying the coords on the board. Only the pieces would be. Either by number/lettering each one individually.

The user would then enter the letter/number of the piece to move, the background code getting the array index of the chosen piece and then checking if it is a legal piece and if it is possible to move. Giving the user an option of which direction to move the piece; If there is such an option. The same would be for capturing opponent pieces/ turning into a king etc. This is the only option to change at the moment.

Third. For loops. Using the above amendment to the array for the board, use of for loops could be reduced. At the very least they can be used with proper practice: the int variables created inside, and solely used by, each for loop. Instead of the current where they are declared global for used throughout the development. This would be the biggest weakness of this game of draughts.

3 Evaluation

3.1 Critical Evaluation

Some fancy stuff written here

3.2 Personal Evaluation

From a personal point of view. I enjoyed the challenge of creating this game. Originally overwhelmed, I began to break the problem into the above mentioned steps. And found it to be manageable. I have learned a lot from this coursework: my understanding of how to use data structures in a real world setting.

I am most proud of my algorithms for creating the coordinate for the new destination during capturing an opponents piece. These were something I had not done before. My problem solving skills flourished here. As i figured out an algorithm to get the new letter & number coord, by creating 2 arrays: one for letters present on the board and one for numbers present on the board.

By checking if the end coord second index in comparison to the start coord second index value; the algorithm determines if the new destination coord is going to be left or right of current destination coord. Next part of the algorithm is to compare the first value of endcoord char array with the letter char array using a for loop. When the end coord = letter index - 1, the index value is stored in a string. This will comprise the first element, letter, of the new destination coord. The second element, number, of the new destination coord is found in a similar fashion: by comparing the second index of destination coords with the number char array and storing the value of number char array when this = destination index 2 - 1. The two values are then concatenated to form the coords of the new destination of the player piece. This is returned for validating an opponents piece can be taken.

The algorithm:

Listing 1: finding new destination coord for capturing opponent piece

```
1   public string checkEnemyMoveToCapture()
2   {
3       if (board.Endcoord[1] < board.Startcoord[1])
4       {
5           for (int i = 0; i < board.Letter.Length; i++)
6           {
7               if (board.Endcoord[0] == board.Letter[i] - 1)
8               {
9                   coordL = board.Letter[i];
10                  newL = coordL.ToString();
11              }
12              if (board.Destination[1] - 1 == board.Number[i])
13              {
14                  coordn = board.Number[i];
15                  newN = coordn.ToString();
16              }
17              newDest = newL + newN.Trim().ToUpper();
18          }
19      }
20      else if (board.Endcoord[1] > board.Startcoord[1])
21      {
22          for (int c = 0; c < board.Letter.Length; c++)
23          {
24              if (board.Endcoord[0] == board.Letter[c] - 1)
25              {
26                  coordL = board.Letter[c];
27                  newL = coordL.ToString();
28              }
29              if (board.Destination[1] + 1 == board.Number[c])
30              {
31                  coordn = board.Number[c];
32                  newN = coordn.ToString();
33              }
34              newDest = newL + newN.Trim().ToUpper();
35          }
36      }
37      else
38      {
39          newDest = "";
40      }
41      return newDest;
42  }
43
```

References

[1] S. Keshav, “How to read a paper,” *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 83–84, July 2007.