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

2 Analysis

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#

3 Design

3.1 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.

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
- RedoUndo
- 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.

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.

Player vs Computer

Computer vs Computer

4 Development

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.

4.1 Player vs Player

4.1.1 Player 1
4.1.2 Player 2
4.1.3 Player vs Player
4.1.4 Undo/Redo

4.2 Computer

This is where shit gets scary

Like Skynet level scary

4.3 Player vs Computer

5 Testing

6 Evaluation

References

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