Edinburgh Napier UNIVERSITY

Coursework Report

Tahira Khan
402278070@napier.ac.uk
Edinburgh Napier University - Algorithms And Data Structures(SET09117)

Abstract

The main purpose of this project is to demonstrate the understanding of Algorithms and Data Structures and their importance in computer science. It is based on the detailed research about functionality, scope and limitations of different data structures. It also required analyzing and selecting the best data structures to solve a particular problem in order to achieve the final outcome.

A checkers game has been implemented using C#. Two dimensional array, list and stack data structures have been used to create the board, implement movements for both human and AI players and perform undo and redo operations.

The learning was that the data structures and algorithms are extremely important in software engineering and if used appropriately can make a great difference in efficiency of a computer program.

1 Introduction

1.1 Problem

The task is to develop a software that implement traditional checkers game. It involves: Identifying appropriate data structures to represent the game board, main elements of the board and positions of those elements on the board.The game can be played between two people, between a person and a computer and between two computers. The game should follow the rules known for playing checkers in real life. The game should also include some extra function such as undo and redo to allow flexibility to reverse player's move.The program should record all the moves and re-display the history in the end of game.

1.2 Design Requirements

Following are some of the requirements analyzed by researching checkers rules and also keeping the project brief in mind:

- Game will be played on a board.
- Board would have 8X8 Cells with alternative white and black colors
- players can play the game on the two opposite side of the board
- At the beginning of the game the board will have two sets of pieces with 12 pieces for each player
- At the beginning of the game three rows of the board will be filled for each set of pieces for each player with two dices having one space in the middle
- At the beginning of the game only black cells will be filled with a piece
- When playing each player will have alternative turn to move a piece
- When playing a player can move a piece forward and diagonally only
- When moving a piece, if an opponent piece comes in the middle and the cell after that is free then the opposite color piece will be removed from the board. The current player's piece will then be placed on the empty cell.
- Once the piece reach the end of the board depending on the side, the piece will become a king piece
- A king piece can move forwards and backwards
- A player can undo or redo a move until the next move has happened
- Once all the pieces of one color are removed from the board, the opposite color will win

1.3 System Requirements

The application has been developed in C# using visual studio console appliocation to develop Command line interface and WPF for the GUI(graphical user interface). Computer must have the following minimum capabilities to run this application.[1]

- 1.6 GHz or faster processor
- 1 GB of RAM (1.5 GB if running on a virtual machine)
- 4 GB of available hard disk space5400 RPM hard disk drive
- DirectX 9-capable video card (1024 x 768 or higher resolution)[1]

2 Design

This section describe the design and architecture in detail, used to develop checkers game.It includes the individual designs of main components and interaction between components to achieve all the functionality.

Two different approaches have been used to design this project in order to show difference between simple design

and slightly complex but more logically structured design with better architecture.

The initial design was first implemented in the command line which was also implemented in graphical user interface. Later the design was re-analyzed and amendments made to design. The new design then was implemented in GUI but can easily be implemented in the command line.

2.1 Overview

The GAME of checkers is played with PIECES on the BOARD composed of black and white CELLS. Two PLAYERS take TURNS making MOVES of the checkers. There are 2 kinds of moves: REGULAR MOVES and JUMPS. Regular moves are made to diagonally adjacent CELLS and jumps are made diagonally over the opponent's checkers to vacant cells and can be continued. When an opponent's piece is jumped it is removed from the board. There are 2 kinds of pieces: NORMAL and KINGS for each set of pieces based on the player. A Normal piece become a king when it reaches the LAST ROW on the board. Normal piece can only move and jump forward but kings can move and jump backward as well as forward. If all pieces of a player are removed from the board, he loses the game.[2]

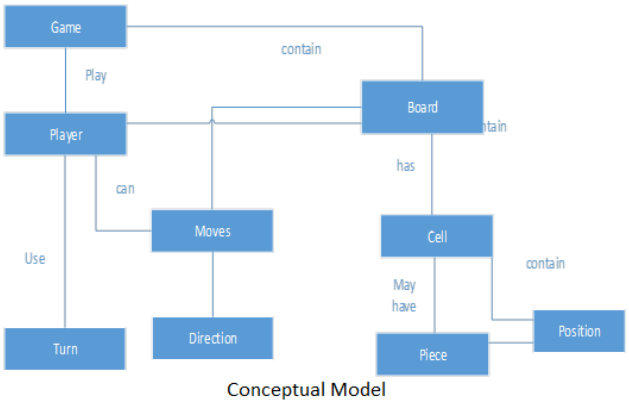

Figure 1: **Conceptual Model** - Some Descriptive Text

2.2 Detailed Description

Starting a Game There are three types of games: player vs player, player vs AI, and AI vs AI that can be chosen by a player from the menu as program starts. The game receives a board from board class and use it to move players on the board using movement class. There is not much difference between two approaches when it comes to starting a game apart from few interface requirements.

In the GUI version, game class that is inheriting from window create and draw all the controls (grid to display the game board on the window and buttons to perform undo and redo). It also creates players based on the game type. The players are created depending on colour of checkers pieces they possess, which side of the board those pieces should be placed and weather they are human players or AI player. By default the player1 has been assigned black colors pieces and placed on the top 3 rows and player2 is an owner of white pieces and placed on the bottom 3 rows.

```
if chosen GameType is HumanVsHuman then
    CreateHumanPlayers();
end
if chosen GameType is HumanVsAI then
    CreateHumanVsAIPlayers();
end
if chosen GameType is AIVsAI then
    CreateAIVsAIPlayers();
end
```

Drawing a Board: When you think of a board, first thing comes to mind is a matrix consists of rows and columns. The Checkers Board is a standard 8x8 board so 2D array has been considered to be best suited to create and display the board since it is a fix sized board with no need to expand in size.

In console application Board class is only responsible for drawing a checkers board and arrange Player1 and Player2 pieces on the board. it was done by simply creating 2D character array and populated with different characters used for Player1 and Player2 objects.

In GUI implementation, Board class is the biggest class that is not only responsible for drawing board using Board cell and Piece objects but also handle all the player's movements. First it determine the cell color and generate alternative color (black and white) cells. Then it decides if a particular cell should have a piece.

```
WhiteCell = false
Cellcolor = 1
for i = 0 to 7 do
    if cellcolor = 1 then
        set cellcolor = 2
    end
    if cellcolor = 1 then
        set cellcolor = 2
    end
    for j = 0 to 7 do
        if mod of j divided by 2 is equal to cellcolor then
            Set white cells = true
        end
        Set white cells = false
        if i is less then 3 or greater then 4 and whiteCells =
          false then
            if i is less than 3 then
                set piece color same as player 1 color
            end
            Set piece color same as player 2 color
            if WhiteCell= false then
                set cell as white board cells
            end
        end
        draw cell on the board
    end
end
```

Setting a Turn: Once the board is drawn the game starts by setting the turn which is initially set to player1. The turn will be changed after every successful move that will allow players to take alternate turn.

Moving Pieces: The main function that differentiate Player vs Player and Player vs AI is the movement. For the Player vs Player the main driving function for any movement operation is the MouseDown and MouseUp events which is only triggered if color of a selected piece desired to move, matches the player's turn. The PieceMouseUp method then pass the selected cell to SelectCellForMove method that decide if selected piece can be moved to destination cell and pass the source and destination cells to move.

For AI player the movements have to be performed automatically that required extra functionality to decide on possible moves. It is done by simply going through all the active pieces a player have and check if it can be moved. Then select first move from a list of AllMoves.

The text version lack the advantage of mouse event but it also simplify the movement of a piece by getting a player to input source and destination cells. Once the location of the source and destination cells are known, they are passed to movement class to validate all the movements.

Validating a Move: Next step is to process all the validations before a piece can be moved. These validations are done in different stages. First it checks if the destination cell is black as pieces should only be moved on the black cells. Next it validates for the direction in order to specify the directions of movement for player1 pieces and player2 pieces. In the end, it checks for the distance. A selected piece can move (one diagonal move or diagonal jump) depending on the condition.

In the text version, each move is validated depending on the player's turn. Then it checks if the position provided for a player is correct and replace the content of source cell with the destination cell.

Making of a King: One of the checkers rules is when a piece reaches the first row of opponent player's side, then it becomes a king. ProcessKingMove method checks if the BoardRowStart is player1's side and the cell moved belong to player2's or vice versa then change the pieces into king.

Completing a Move: Once the validation is completed and as a result a selected piece can be moved, the complete move method in the move class replace the destination cell with the source cell. If there is a valid jump then a piece of the opponent player would be captured and stored in the list of killed pieces. This list then can be used to restore the piece in case of undo.

Checking a Winner: To check if there is winner CheckWin method simply checks if any player has lost all the active pieces. this way the other player will be declared a winner.

Undo Move: One of the requirement was to implement undo feature to let players reverse their move. Stack also called LIFO data structure is widely used for this purpose due to the functionality it provides. It simply stores the data in a way that allows only the last item to be removed first. The game class contains move stack that store state of a completed move (object) at every move and then used by Undo method in the move class to pop the last move.

Redo Move: Once the undo was implemented, it was very easy to implement the redo which simply put the game to the state before last undo. It was achieved by creating another stack to push all the undo movement popped by movements stack.

3 Enhancements

The developed software provide most of the required features but there are many possibilities to make this more robust and user friendly. Some of the features which can be implemented are as following:

- Completing the text version and allowing player to choose between command user interface and graphical user interface
- The replay feature can be implemented by storing all the moves in the queue (FIFO) data structure and then re-display it using dequeue function
- To make a clever AI move, more functions can be implemented such as Move to save, Move to Kill, Move to become King
- The player can be asked to enter their names and store points for every capture of the opponent's piece
- Validating and performing multiple jumps
- Each game can be paused and saved in a text file in order to play that can be retrieved and played
- The GUI interface can be improved using wpf controls

4 Critical Evaluation

The basic architecture used is very logical that made it easier to implement undo, redo and AI features. The graphical user interface is also a plus point that makes it easier and more enjoyable to play. Appropriate data structures have been used to draw board and implement undo and redo.

Some of the features e.g. undo, redo and AI in console application have been implemented properly due to time limitations.

overall the design and solution of the problem meets most of the requirements.

5 Personal Evaluation

It has been a great learning experience that required me to research on different data structures and algorithms in order to achieve set goals. I used past software engineering experience and also learned to adopt more modular approach to develop a software.

An agile software development methodology has been used throughout software development process that allowed flexibility to redesign the software and make amendments to implementation. However it is extremely important to get initial requirements right and design the architecture that allows implementing all the basic requirements and also flexible enough to include extra functionality. Initially I only took a game board and movements into consideration without considering the undo and redo features that required saving the previous states of the game movements.

Considering it is data structures and algorithms project, I should have used more data structure, for example Hash table to store location of all the cells can contain pieces and binary tree to implement multiple jumps.

I also learned that decision making and time management are very important factors that contribute to a failure or success of any project. By not deciding on a particular interface and using specific approach towards a problem, I made it difficult for myself.

The main challenges faced during GUI implementation were implementing undo, redo and AI operations that required changes in architecture. For console application (not completed), testing was very time consuming.

6 References

How to Write a Design Report[3]

Learning Draughts/Checkers[4]

Tic Tac Toe in C[5]

Appendix

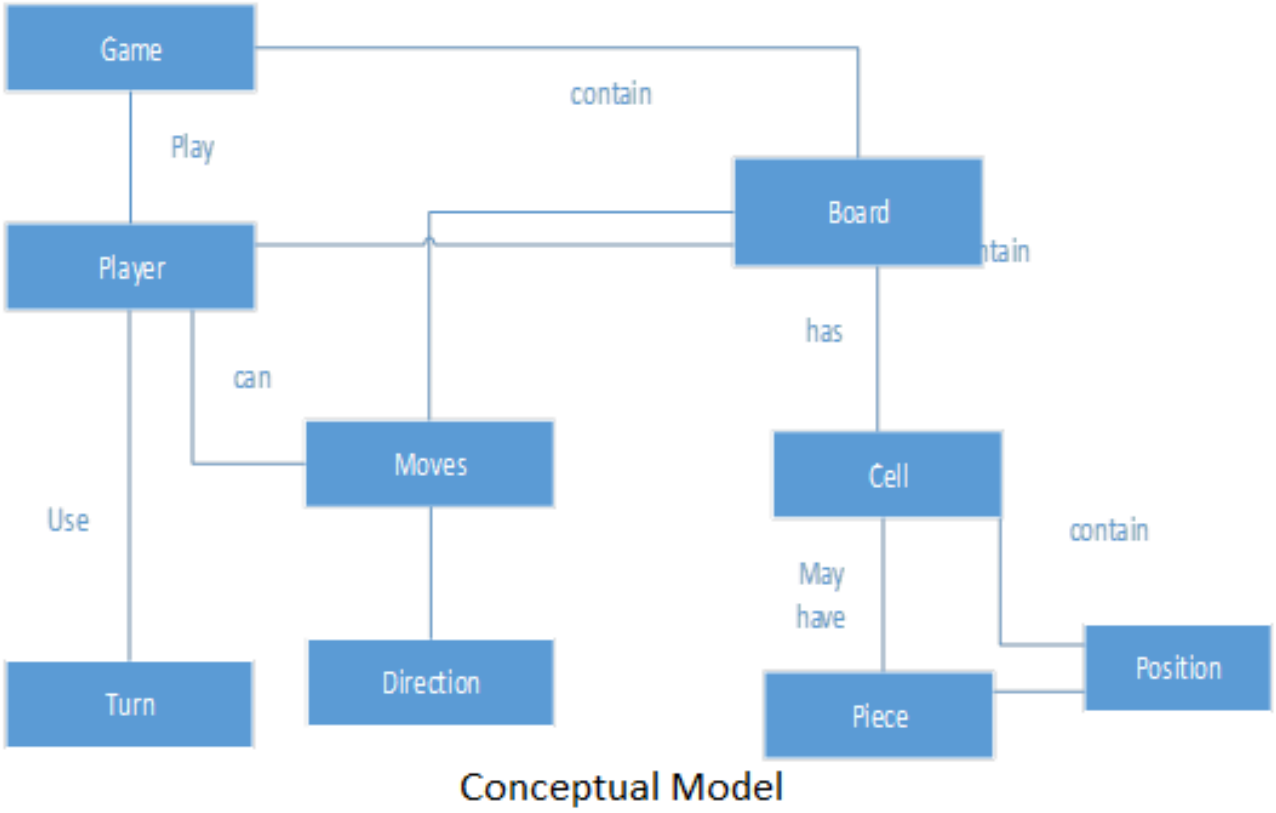

Figure 2: **Conceptual Model** - Represent Design Architecture

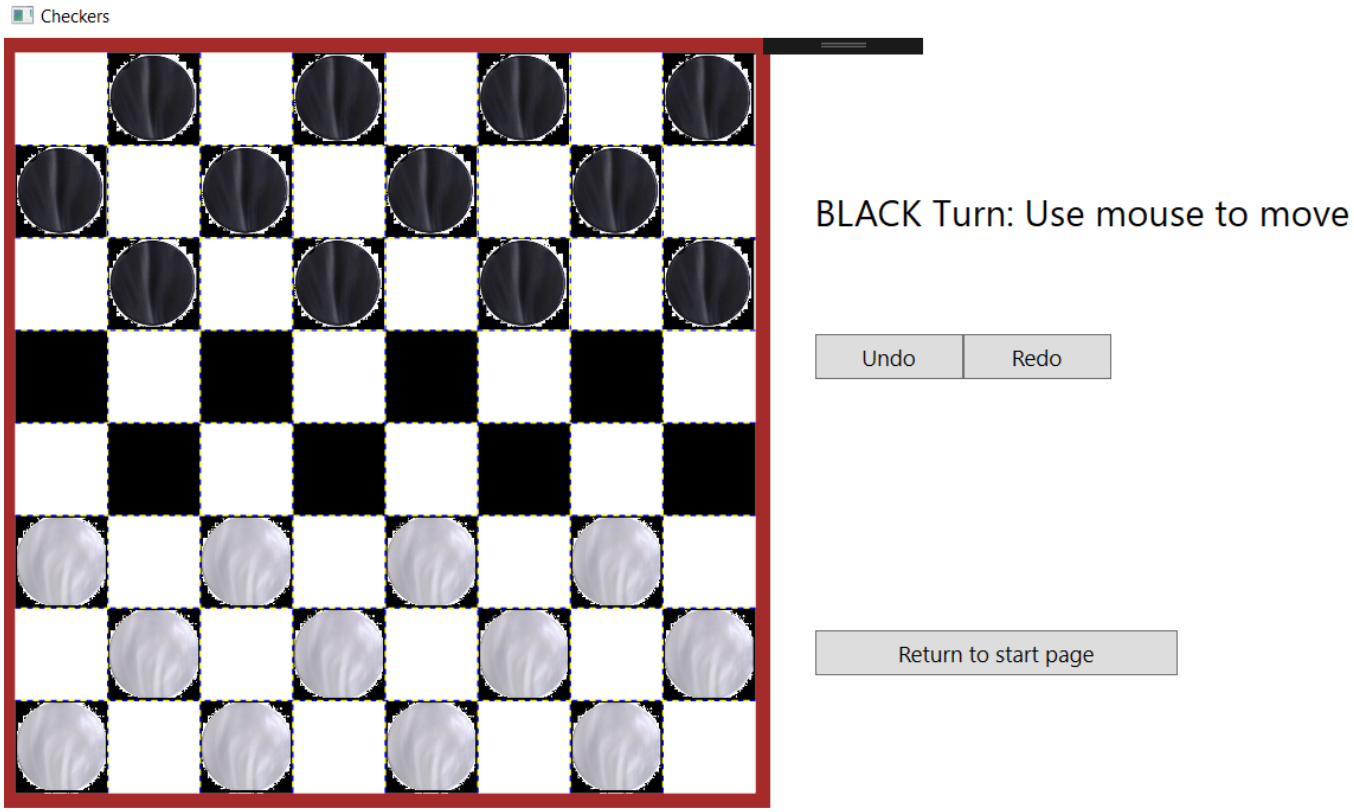

Figure 3: **Board** - Pieces organized on the Board

References

[1] "System requirements."

[2] J. Sigle, "Using a checkers program to illustrate object oriented design."

[3] "How to write a design report."

[4] pseudonym67, "Learning draughts/checkers."

[5] "Tictactoe in c."