# GLADOS: A Grid-Like Arrayed Draughts Organising System

Brian Cleland
40322683@live.napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures (SET09117)

## Abstract

The purpose of this task is to implement a checkers game in a familiar language (C). Utilising knowledge both from taught interactions and self-directed learning a selection of appropriate data structures were selected to represent the game and all of its necessary components so that a game can be played between two people, a person and a computer, or between two computer controlled players.

**Keywords** – Algorithms, Data, Structures, Data Structures, Checkers, Draughts, Multidimensional Arrays, MiniMax, NegaMax, C-Sharp, AI, Artificial Intelligence

## 1 Introduction

Having been tasked with producing a draughts program with simple AI and suitable data structures and algorithms to support it. The design requirements for the game include the ability to record play history, the game must also support the ability to undo and redo moves, using appropriate data structures and the game should implement a simple AI player. This document intends to justify the selection of data structures and algorithms chosen for these purposes.

## 2 Design

In order to produce a working program key information on the layout, basic game mechanics and turn structure were derived from a physical copy of the game. Using this as a template a number of assertions could be made.

Firstly, the board must be capable of storing the states of all the squares in play; including invalid and empty spaces, spaces containing Red pieces, spaces containing Red Kings, spaces containing Black pieces and spaces containing Black Kings.

In order to do so efficiently a decision was made to use the following data structures:

- Multidimensional Arrays for the board/s - Dictionaries for storing arrays with a particular key - Lists for storing array data that didn't require a particular key

In order to carry out the games main functions a decision was made to use: - Looped selection statements - Weighted move selection

**Design Choices** All major design choices were made by first researching and referencing commonly accepted rules for Draughts, identifying the critical game components (including 2 players, a board, 12 red pieces and 12 black pieces and a turn structure)

By breaking down each of these components into their most basic forms it was apparent that the best way to proceed would be to create an 8x8 board array utilising the C int[,] array method. This would define each row on the board (from 1-8) with a corresponding array (from 0-7), each column on the board is then referenced by the array indices. e.g. column 1 is array index 0 and column 8 is array index 7 etc.

During the inception of this project it was determined that the actual pieces are not required for game play as they merely represent the states of each square on the board. As such the board is just a representation of one of the following states: - Empty - Contains a Red piece - Contains a Black piece - Contains a Red King - Contains a Black King

This makes the handling of the logic of the game much simpler as we now understand that the movement of pieces is actually representative of the transposition of a value from one array index to another.

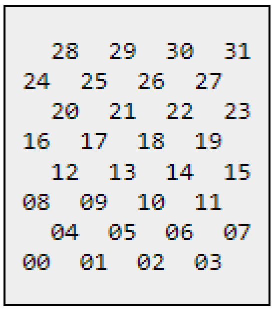
**Weighting** The decision to implement a weighted decision making algorithm in this examples arrives from the fact that checkers can be considered a "game of perfect information", such games include draughts(checkers), tic-tac-toe, chess, Othello etc and are considered such as it is possible to see all the moves that can possibly, and currently do, exist on the board. As such it is possible to calculate all, or some of these moves and return the options to the program to execute.

The decision to utilise a simple weighted AI mechanism comes from its simplicity in function. The basic precepts of move weighting are: - All possible legal moves are calculated - Moves are ranked according to whether the piece being moved lands on a blank square or a square of an opposing colour. - A list of moves is created where the list contains moves that rank highest out of all possible moves. - A move is then selected from this list for the AI to select (in this case at random)

From this premise it is possible to recursively iterate through the board, check for pieces that can be moved. Determine the best move to be made and return that move.

## 3 Enhancements

While the core game mechanic works it relies on a number of conditional statements to evaluate validity of moves within the board. This is due, in part, to the nature of the task and the decision to utilise multidimensional arrays for the board. A more elegant solution would be to use bitboards(fig 1).



```
  28  29  30  31
24  25  26  27
  20  21  22  23
16  17  18  19
  12  13  14  15
08  09  10  11
  04  05  06  07
00  01  02  03
```

Figure 1: **A bitboard** - An example of a bitboard that can be used in draughts. A bit board is generated for each colour and bitwise operations occur on the two boards to determine moves.

This would improve the robustness of the game logic, requiring fewer checks reduces the chances of failure at any of these checks, and would improve the speed of the program as well. as it would be possible to make many more calculations in one iteration of the games logic.

## 4 Critical Evaluation

While the final product works to explore the use of multidimensional arrays and simplistic AI algorithms there are a number of improvements that could be made.

In looking at what features or methods could be improved it would be pertinent to highlight features that would be considered to work well in this application:

- The multidimensional array storage for the board is central to the functioning of the program and works to the advantage of the program, providing a legible and inherently understandable representation of the board. - The use of dictionaries and lists for storage of specific data is a robust way to store information for retrieval with minimal impact to the system. - The storage and retrieval of data is managed easily using .Net libraries - The UX and UI are pleasant and make the application easy to use

Following on from these point it is worth noting that the following features that could be improved: -Currently the AI can take around 250 moves to complete a CPU Vs CPU game, this is because of the choice to use 0-depth weighted move selection. By improving the AI and recursively checking the board states in would be possible to produce a n-depth move list that would result in fewer turns being made by the computer. In this case the computer would likely win all matches Vs a human opponent and draw against itself on every game. -The game logic can be improved and refactored to use fewer conditional statements when checking for valid moves, thereby improving overall stability.

## 5 Personal Evaluation

While the initial logic of the program was easy to discern, implementing the simple logic of the game in a manner which could be used within the confines of the structure proved more difficult than expected. Deciding to use C as the language for this project added a steep learning curve to the development of the application and required learning a lot of new methods and syntax in order to translate the design into something meaningful in C.

Many of the features used in this program were not specifically required and deciding to implement them from an early stage put a lot of pressure on integrating them into the game design correctly. This created a bottleneck where the decision to implement a certain feature for example, using the SoundPlayer method to add background music, forced the development of the application in a specific direction. The most complicated aspect of this was learning how to use StreamReader and StreamWriter methods to correctly store and retrieve data.

There were also hurdles in learning how to correctly pass information into and out of dictionaries and lists. While the logic behind using these was simple the actual implementation proved more difficult than anticipated and a great deal of man hours in the project was directed to learning how to correctly utilise these for the project.

## 6 Conclusion

The design, creation and implementation of this game was both challenging and rewarding. Learning how to correctly use appropriate data structures to store the game states and designing a simple algorithm for AI play may have been taxing but the ultimate outcome (the finished product) have provided valuable insights into; not only development processes but underlying methodologies for game theory and design, good practice in self directed learning and time and resource management.