Coursework Report

William Daglish
40216858@napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures (SET09117)

1 Introduction

The aim of this coursework was to design and code a Checkers/Draughts game using any methods & coding language that was desired.

Game such as Checkers, TicTacToe, etc. require the use of specific data structures to produce things like the game boards and player moves/pieces. By undergoing this project, I will have to demonstrate my knowledge of different data structures and show how they can be implemented to aid in completion of certain tasks.

Additionally, there are several functions that my game is required to have, such as: undo/redo moves, watch replay of old games, and an AI (computer) player. These functions can be accomplished with algorithms that make use of the data structures I have chosen to use.

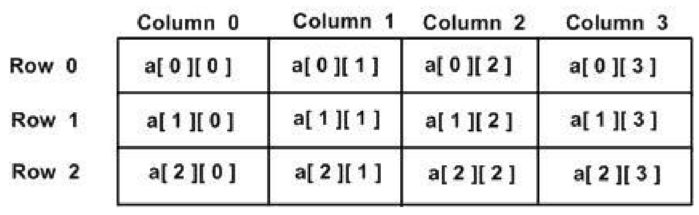
As a bonus, my aim was to get these functions working within a GUI environment. This will obviously be more visually appealing and more user friendly than a console/text based game. I prefer the GUI to be clean and clear, with only the required interface buttons and labels etc.

Keywords – Algorithms, and, data, structures, coursework, report, checkers, draughts, game

2 Software Design

The code language I decided to use for the creation of my game was Java, as this is a language I wanted to get more use out of and try and improve my knowledge on.

The main feature of the game was a playable game board, and this is where the main data structure comes into place. The obvious choice was the use of a 2D multidimensional array, as it is immutable and can be thought of quite simply as a table of objects. (Fig 1)



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Figure 1: - Multidimensional Array Example

To make the array user interactive, it is an array of type "JButton", a button used in a Java Swing application. (Fig 2) Each button in the array is a position (X & Y coordinate) on the game board.

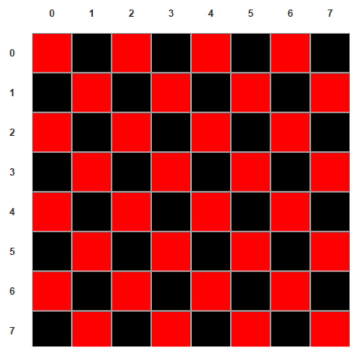


Figure 2: - Multidimensional JButton Array for Board

When loading a new game, each of these positions can be assigned a Checker Piece where required. The checker pieces themselves are "Icons", so when a piece is loaded at coordinate 2,4 for example, that square is assigned an icon, e.g. Black Checker. (Fig 3) Each button is then set with an ActionListener which handles clicks.

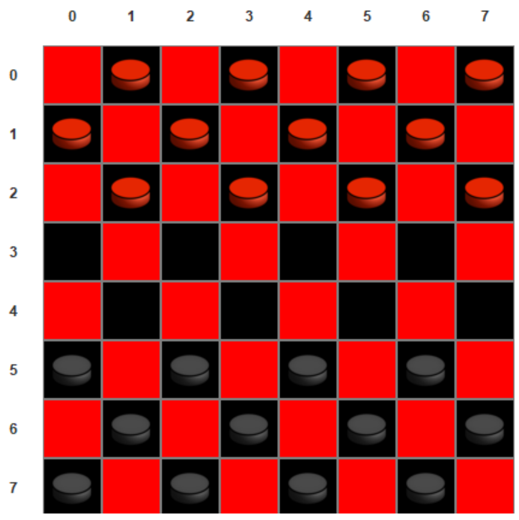


Figure 3: - Pieces (Icons) Loaded to the Board

To start a new game, quit a game, or access some other functions, the user can interact a menu which is implemented with a "JMenuBar". (Fig 4 & 5) The different items in the menu make use of specific "ActionEvents" and call methods which setup a game, clear a game, and carry out the additional functions specified in my introduction (undo, redo etc.).

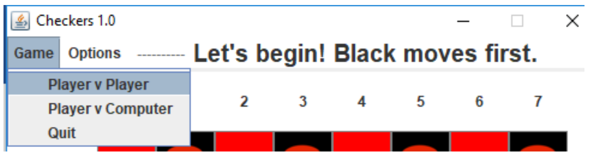


Figure 4: - Choosing a Game Type

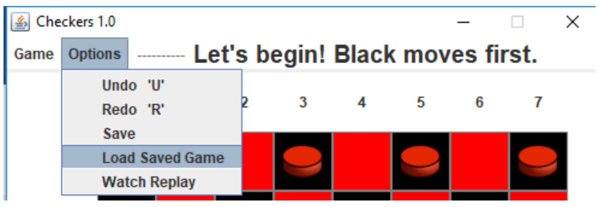


Figure 5: - Different Game Functions

Probably the most complicated part of this project was to setup and restrict the moves a user could make.

The actual legal moves and jumps themselves are monitored by methods in the Move class. When a user clicks on a square to try and move a piece, the ActionListener passes these source and destination coordinates to the makeMove/makeJump methods in this class. The method then checks if it can move/jump (Listing 1), if it is a legal move then it returns true and the piece is moved.

If the move is not legal then an "Invalid Move" message is returned to the user. All messages like this are shown next to the Menu bar with the use of a "JLabel".

Listing 1: canJump Algorithm - Checking if a Jump is Legal

```java
public static boolean canJump(int player, int fromX, int fromY, int jumpX, int jumpY, int toX, int toY)
{
  if (toY < 0 || toY >= 8 || toX < 0 || toX >= 8)
  {
    return false; //Move is off the board.
  }
  if (Board.boardSquares[toX][toY].getIcon() == Piece.BLACKCHECK || Board.boardSquares[toX][toY].getIcon() == Piece.↩
    REDCHECK ||
      Board.boardSquares[toX][toY].getIcon() == Piece.BLACKKING || Board.boardSquares[toX][toY].getIcon() == Piece.REDKING↩
    )
  {
    return false; //Move already contains a piece.
  }
  if (player == ActionClicks.REDPLAYER)
  {
    if (Board.boardSquares[fromX][fromY].getIcon() == Piece.REDCHECK && toY < fromY)
    {
      return false; // Regular red piece can only move up.
    }
    if (Board.boardSquares[jumpX][jumpY].getIcon() != Piece.BLACKCHECK && Board.boardSquares[jumpX][jumpY].getIcon() != ↩
    Piece.BLACKKING)
    {
      return false; // There is no black piece to jump.
    }
    return true; // The jump is legal.
  }
  else
  {
    if (Board.boardSquares[fromX][fromY].getIcon() == Piece.BLACKCHECK && toY > fromY)
    {
      return false; // Regular black piece can only move down.
    }
    if (Board.boardSquares[jumpX][jumpY].getIcon() != Piece.REDCHECK && Board.boardSquares[jumpX][jumpY].getIcon() != ↩
    Piece.REDKING)
    {
      return false; // There is no red piece to jump.
    }
    return true; // The jump is legal.
  }
}
```

The other functions in this game such as the AI player, the undo/redo methods and the replay, all make use of a "List" data structure.
This is used as the number of items that are being stored will vary a lot throughout the game, so the data structure needed to be mutable.
All pieces that are owned by the AI Player are stored in a List at the start of every move (Listing 2), so that the AI can try and move from one of these locations.

Listing 2: - getAIPieces Method - Fill a List with Coordinates of Any AI Controlled Pieces

```java
public static void getAIPieces()
{
  for(int i=0; i<8; i++)
  {
    for(int j=0; j<8; j++)
    {
      if(Board.boardSquares[j][i].getIcon() == Piece.REDCHECK || Board.boardSquares[j][i].getIcon() == Piece.REDKING)
      {
        //Search for pieces that are red and save their X & Y coords to list
        computerPieces.add(Integer.toString(j) + "" + Integer.toString(i));
      }
    }
  }
}
```

The algorithm that makes the AI player move makes use of the Random function (Listing 3) in Java, calling a random piece from the list and calling a number between 0 - 3, with each number pointing to a specific move/jump direction. If this move is not possible then the number simply increments and tries again. After a successful move, the AI Pieces list is cleared so that it can be updated next turn.

Listing 3: - Methods for AI - Getting a Random Source & Destination Coordinate

```java
private static void getRandomPieceSource()
{
    //Get a random x & y source coordinate from the list of controlled pieces
    Random randomPieceSource = new Random();
    randomSource =  computerPieces.get(randomPieceSource.nextInt(computerPieces.size()));
}

public static void getRandomPieceDest()
{
    //Random int from 0−3
    Random randomPieceDest = new Random();
    randomDest = randomPieceDest.nextInt(4) + 0;
}
```

Finally, the History class makes use of a List to store the moves that are made during a game (Listing 4). These elements in the list contain the source and destination coordinates of each move. This allows the user to call the undo/redo functions.

Listing 4: - Methods for History List - Stores the Move Coordinates to the List

```java
private static List<String> historyOfMoves = new ArrayList<>();

public static void addToHistory()
{
    historyOfMoves.add(Integer.toString(Move.originX));
    historyOfMoves.add(Integer.toString(Move.originY));
    historyOfMoves.add(Integer.toString(Move.destinationX));
    historyOfMoves.add(Integer.toString(Move.destinationY));
    historyOfMoves.add(System.lineSeparator());
}
```

Not only can a user undo/redo moves but they can also save a particular game and load it later to continue play or watch a replay of all the moves carried out.
If a user chooses to save a game, the current list of moves taken is stored to a text file, line by line. This can later be read using a buffered reader.

If the file is selected to "Watch a Replay", then the buffered reader is loaded into a new thread and the thread "sleeps" for 1 second after each move. (Listing 5) This allows the user to easily see the game being played out.

The reason that multi-threading is being used for this algorithm is that it prevents any issues the "Sleep" method causes when used in conjunction with a Swing application.

Listing 5: - Method for Action Replay - Loads the Selected File and Plays Out Each Move with a 1 Second Delay

```
public static void openHistoryThread()
{
    new Thread(new Runnable()
    {
        @Override
        public void run() {
            try
            {
                readFromHistory();
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }).start();
}

private static void readFromHistory() throws InterruptedException
{
    Piece.setupNewGame();
    Thread.sleep(1000);
    try (BufferedReader br = new BufferedReader(new FileReader(selectedFile)))
    {
        String line;
        while ((line = br.readLine()) != null)
        {
            int originX = Integer.parseInt(line.substring(0,1));
            System.out.println(originX);
            int originY = Integer.parseInt(line.substring(1,2));
            System.out.println(originY);
            int destinationX = Integer.parseInt(line.substring(2,3));
            System.out.println(destinationX);
            int destinationY = Integer.parseInt(line.substring(3));
            System.out.println(destinationY);
            if((destinationX == originX + 2) || (destinationX == originX − 2))
            {
                Move.makeJump(originX, originY, destinationX, destinationY);
                ActionClicks.resetBorderHighlights();
                System.out.println("Jumping: " + originX + " " + originY + " " + destinationX + " " + destinationY);
                Thread.sleep(1000);
            }
            else
            {
                Move.makeMove(originX, originY, destinationX, destinationY);
                ActionClicks.resetBorderHighlights();
                System.out.println("Moving: " + originX + " " + originY + " " + destinationX + " " + destinationY);
                Thread.sleep(1000);
            }
        }
    }
    catch (FileNotFoundException e)
    {
        e.printStackTrace();
        System.err.println("Error: File can't be read.");
    }
    catch (IOException e)
    {
        e.printStackTrace();
        System.err.println("Error: File can't be read.");
    }
}
```

3 Enhancements

Possible improvements that could be made to this game would be the use of "Minimax" algorithms for the AI Player. These would help make the AI a more effective player by letting it evaluate each of its current possible moves by determining whether that move will end up resulting in a win.

Another possible feature that could be added to this game would be the option to select different international rule sets. Certain countries use different rule sets to the ones that are used in this game.

4 Critical Evaluation

The main feature that I feel works well is the actual use of a GUI over a text based game.
As I had initially planned, I feel that it makes the game more user friendly and more interesting to play.

The buttons worked well for moving different pieces around the board, and the algorithms which determine jumps/moves that are legal also highlight these possible moves (Fig 6). This again makes the game more user friendly.
I also feel that the AI makes quite an effective player. Even though I didn't use a minimax algorithm, the fact that the AI moves are completely random means that they are unpredictable.
This prevents the user from guessing how the game will play out.

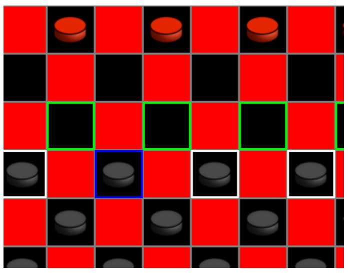


Figure 6: - Highlighting Possible Moves & Selected Piece

5 Personal Evaluation

This project has been very challenging in some areas, particularly in getting the actual gameplay and movement working. It has however forced me to think of different ways I can accomplish the tasks set out in the specification, and as described, what data structures I will need to use and why.

I also found that creating a game was more interesting and fun than some other projects I have been given in the past. Overall I think that I have managed to completed the majority of the tasks set out in the original coursework specification, and I am fairly happy with my final product.