Coursework Report

Andris Vinkalns
10010604@napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures(SET09117)

1 Introduction

Required solution: Requirement for this coursework is to implement a Checkers game, which would offer user 3 different types of play. They would be as follows:

- **Player versus Player**
- **Player versus Computer(AI)**
- **Computer versus Computer.**

All of the game types need to follow the rules of the checkers game know in UK also as "**Draughts**". It is know that the different countries in the world have different rules for this game, but this project will only make a game based on the rules known in the UK. Worth mentioning is that I have chosen my game piece colours to be white and black respectively. In some sources the colours white and red are used but the decision was made to stick to colours which are used also in chess.

Other features: Also players should be able to undo their moves and redo the undone one if they change their mind again. The user needs to know when it's their turn to act and, additionally, should be able to see all the move history for both players. The user should be notified if he needs to capture a piece, and also should know if the game has ended because of one of the players has won/lost or if there is a draw.

Chosen programming language: The choice of programming language for this coursework was left to the developer and he has chosen to use Java as it's the one he is most familiar with and has had experience with before. He will also provide a singe executable "jar" file which can be run from Java console.

2 Design

Tools used For developing software in the Java programming language, the following tools were used :

- **"IntelliJ IDEA"** : Java JDK created by "JetBrains"
- **Github** : for version control and backup purposes
- **"Trello.com"** : online tool for Creating boards in that way keeping track of features to be implemented and bugs to be fixed

2.1 Classes used

The developer started designing the application by creating few of the main classes which he thought he will be using in the application. He created **Board**, **Player**, **Move** and **Game** classes. Originally the program was created with a **Piece** class, but at the end this class was not needed in the design. When it came to implementing AI, the developer added another two classes called **MoveBundle** and **AI** which was the subclass of **Player** class. This was the only time inheritance was used while designing this application. The developer also had **Prompter** class which was responsible for all the user input validation and interaction with the user. And of course the program still has **Main** class which is launched when it is executed.

The following is a brief description of each Class in the application and what purpose it has.

Board class This class will represent a game board which is 8x8 pieces big. It will also store many methods used to find coordinates on the board and transfer them in different format. It will also have a functions for drawing and updating board

Move class This is one of the core classes, which will represent all the information necessary for the player(or AI) to make a move. It will store starting and ending coordinates of a move, it's captured piece coordinates(if any) and a copy of a board at the start of the move which will be used if a move needs to be undone.

Game class This class has all the game logic, which is necessary for playing any of three game types. It also has methods for AI logic, game starting and finishing methods.

Prompter class As mentioned earlier, this class is responsible for interacting with the human player. It has methods to ask the user for input and methods to validate such input.

Player class This class stores the players name, piece colour and all the moves the player has made or undone during a game. It also has a subclass **AI** which inherits all these properties, but, also adds a field which stores all possible moves for AI at the start of its turn.

MoveBundle class This class is created so that the program can manage moves which had multiple captures. Its only parameter is ArrayList which stores **Move** objects which each represent a single capture move. It also has a function which gets only the last element of this ArrayList.

Main class This class launches the program, and is responsible for the welcome screen and successful run of the program

2.2 Data Structures used

Arrays In this project only a few different data structures are used. First of all, in the Board class it has a **two dimensional string array** of size 8 by 8 which represents each squared piece on the board. At the start of the game this array would look like this:

Listing 1: Game board representation as 2D array

```
public String [][] board =
        {
        {"w"," ","w"," ","w"," ","w"," "},
        {" ","w"," ","w"," ","w"," ","w"},
        {"w"," ","w"," ","w"," ","w"," "},
        {" ","*"," ","*"," ","*"," ","*"},
        {"*"," ","*"," ","*"," ","*"," "},
        {" ","b"," ","b"," ","b"," ","b"},
        {"b"," ","b"," ","b"," ","b"," "},
        {" ","b"," ","b"," ","b"," ","b"}
        };
```

As you can see all the elements have value. **w** and **b** represent white and black player pieces on the board. ***** represents a piece on the game board where players can potentially move to (in Checkers players only use black squared pieces to move their figures). The rest of board has elements of **" "** which is just a space, to make it visually understandable that it's still a place on the board, but, not available for player pieces. To access any of the elements, the program uses a basic **int array** with size of two where first element is x coordinate and second is y coordinate. So to access a piece in the board position [2][4] the developer would enter the coordinates [4][2] as y=2 and x=4 or in other words y is the row number (starting from 0) and x is the column number. The program also uses string array of two elements when it asks the user to input the next moves coordinates. It looks like this:

Listing 2: Splitting user input coordinates

```
String[] coordinates;
coordinates = playersInput.split(",");
```

Then the function is used which converts these coordinates to integer values which can then be used in games logic.

Stacks When thinking about the ways to record each players taken and undone moves it came pretty clear that the best Data Structure to store MoveBundle objects (which is all moves done at players turn) would be **Stack**. Mainly because of its "last one in = first one out" structure. The program was created with two Stacks in **Player class**, one for Moves(Turns) taken and one for undone ones.

Listing 3: Constructors for MoveBundle Stacks

```
public Stack<MoveBundle> movesTaken = new Stack<>();
public Stack<MoveBundle> movesUndo = new Stack<>();
```

That allowed the developer to use already built in methods like **push()** to add new element in the stacks and **peek()** to access the last MoveBundle elements in either of these stacks. If the move needed to be undone, all that was required, was to take the last element of **movesTaken Stack**, put it to the **movesUndo Stack** and update the board accordingly. It also gives the option to keep each players moves separate. Other data structures had some drawbacks for storing MoveBundle objects in them, and Stacks were the best solution for such problem.

ArrayLists The program uses ArrayLists in two occasions in this project. First of all in the **MoveBundle** class I stored all the **Move** class objects in the ArrayList. This data structure was chosen purely because of its simplicity to use and search through the elements(linear search). It was used when it was not possible to use **arrays** because it was not known how many elements it might contain. For that same reason, the ArrayList was used to store all the possible available AI move starting coordinates. In each situation there might be different amount of possible moves to be done by AI. So ArrayList was a good data structure to use for such purpose.

2.3 Algorithms used

Movement algorithm for players (human and AI) The program has its own algorithm which it uses when player wants to move a piece on a board.

If it's a human player, the program will first of all check if this player needs to capture any opponents pieces. If so, then player will be notified about it and asked to enter move coordinates, which will not be accepted until they are valid and contains the piece which can do the capture. Then the capture will be done and pieces will be moved in board array and a new representation of the board will be printed out. Another check if that player's piece can do a capture will be done and if so the capture process will be repeated until no captures with that piece are possible. At the end of the players turn the **MoveBundle** object gets put in a stack and a history of all the moves taken by the player is also displayed.

If the human player can't capture then the move is more simple. The program will verify the coordinates and make a move. Before printing the new layout of the board, it will check if the piece hasn't reached the other side of board, and if it has, it will make the piece "capital". That would represent a "king" piece. Also worth mentioning is that in the starting coordinate validation this is also checked and "king" pieces can move and capture backwards.

For **AI** moves, the program also checks for capture moves to be done. If it finds a piece which can capture, it stops the search and takes the piece's start coordinates and performs the capture move similarly to the human player. If at the start of AI turn there is no capture to be done, then the program collects all the possible moves by finding pieces which can do the move and then setting possible target coordinates for them. All these moves gets stored in ArrayList. Then, using **Random class**, the program chooses one element of this ArrayList and executes that move and adds it to AI's "movesTaken" Stack. In next page there is a code sample showing how the program chooses AI move randomly.

Listing 4: Randomly selecting an AI move

```
 computer.setAllPossibleMoves(getAIMoves(computer,board));
 //computer finds all the possible moves(if it finds a capture it stops looking for other moves
 System.out.println(computer.getName()+ " choosing between "+computer.getAllPossibleMoves().size()+" figures which can move");
 Random random = new Random();
 Move computerMove = computer.getAllPossibleMoves().get(random.nextInt(computer.getAllPossibleMoves().size()));

```

In this example the "**computer**" is AI player, "**getAIMoves**" function collects all the possible AI moves at that moment. And "**computerMove**" object is just that move which is chosen and will be executed.

Undo and Redo algorithms The program offers user to cancel his last move by entering "**undo**" when asked for the next move. That triggers the undo function which first of all checks if the user has done any moves previously. If so, then the algorithm first of all will cancel the opponent's move. Here worth mentioning is that if the user has done the move then the opponent will most certainly have done one as well, because the user will be able to undo his and the opponent's move only at the beginning of his next move. To undo any players move, the program checks the top element of players "**movesTaken**" stack, which is "**MoveBundle**" with one or multiple "**Move**" objects. Each of these objects has a copy of board 2D array representing state of the board before the move has been performed. This copy then is used to update current board status. If "**MoveBundle**" has multiple objects then algorithm starts the undo process with the last one in **ArrayList** to get the right representation of a board. After "**MoveBundle** elements are undone, Bundle gets taken of the "**movesTaken**" stack and put in "**movesUndo**" stack. The same gets repeated for the other player and both player's move history gets updated. The program is not able to undo all the moves at once at this moment, but, it can be done by manually entering the "**undo**" command enough times until there is no move to be undone. In the future enabling such a feature will be considered.

Listing 5: Undo function

```
public void undoMove(Player player1, Player player2)
{

    if(!player1.movesTaken.isEmpty())
    {//takes move off and puts it in another undo move stack
        MoveBundle player2Bundle = player2.movesTaken.peek();
        MoveBundle player1Bundle = player1.movesTaken.peek();

        ArrayList<Move>p2Moves = player2Bundle.getAllMoves();
        for (int index=p2Moves.size()-1;index>=0;index--)
        {//for each move in the latest bundle
            Move move = p2Moves.get(index);
            board.board=move.copyBoard(move.startOfTurnBoard);
        }
        System.out.println("Undoing "+player2.getName()+"'s last move");
        board.drawBoard();
        player2.movesUndo.push(player2.movesTaken.pop());

        ArrayList<Move>p1Moves = player1Bundle.getAllMoves();
        for (int index=p1Moves.size()-1;index>=0;index--)
        {//for each move in the latest bundle
            Move move = p1Moves.get(index);
            board.board=move.copyBoard(move.startOfTurnBoard);
        }

        System.out.println("Undoing "+player1.getName()+"'s last move");
        board.drawBoard();
        player1.movesUndo.push(player1.movesTaken.pop());

    }
    else
    {
        System.out.println("No moves to undo");
    }
}

```

The "**Redo**" function gets triggered the same way as the "**Undo**" one, and its algorithm is almost identical. The only difference is that the program is checking "**movesUndo**" stack for the ""**moveBundles**" which have been undone. Any moves found in that bundle gets executed in order starting from 1st. When applying redo program is not using the copy of the board as the target positions for the piece. It only performs the moves as it would if the player would do them first time.

A **possible issue** for this algorithm is that the "**redo**" function will only work if triggered straight after "**undo**" has been performed. It doesn't allow the scenario if the user undoes a move then performs a genuine move, but, in next round wants to redo his last undone move and the move he performed after it.

Coordinate validation algorithm This algorithm is used the most in the program. When the user enters move coordinates he wants to make, the program then does all the necessary checks to make sure the move is valid. First of all users **String** input gets converted to **int** coordinates so that they can be used to find a specific element in **Board** class 2D array. The program checks if the integer values are in range of the size of an array. Then the program will check if player has a piece in that location. Subsequently it will do the following checks:

- if the target location is free to travel to
- if it is in allowed distance and right direction("non-king" pieces can't go backwards)
- if the target coordinate is not equal to " ", or in other words a piece on board which cannot be used for movements

If any of these checks fail then user is asked to re enter the coordinates.

3 Enhancements

3.1 Improvements and additional features

If there would be more time for the project, further improvements can be made.

One of the main areas needing improvement would be **graphical interface and usability**. At this stage the program is only "console based" so would be better if it would have a nice Graphical User Interface(GUI).

The **AI algorithm** can be improved too, specifically how it chooses moves and captures. It would be good to implement some kind of scoring system for all potential AI moves and then let it choose the one with the best score. That would make AI logic much more advanced.

4 Critical Evaluation

Generally the program performs well. In particular **it plays by all the rules**, which is very important, because the user doesn't want to play a game if it doesn't use the rules he expects it to.

Usability is acceptable, with a few flaws. The graphical interpretation of game board could be much better, but, the information provided to the user is comprehensive and intuitive.

A feature which can clearly be improved is, as already mentioned, **AI logic**. At this stage AI performs well at a basic level. That means it plays by the rules of the game. But it doesn't analyze which move could be potentially better than other. At this stage AI is pretty "stupid", but still competes by the rules of the game.

Another feature this program lacks is immediate "**Replay**" function. It can be done by undoing all the moves manually and if wanted can redo them all, but it would be much nicer if it could be triggered by user command automatically.

The code quality of the program also can be improved. Some of the methods are in the classes, but, it might be more intuitive to perhaps reallocate them to different ones. The developer never had time to evaluate the **effectiveness of the Algorithms**, so, most likely that can be improved too. Program input can be more "foolproof". At this stage program expects user to follow instructions and only detects certain type of wrong inputs. This issue should be easy to fix.

5 Personal Evaluation

Challenges There were a few challenges during development process. The biggest challenge of all was probably to implement all the rules of the game. The Checkers game is not famous of having complicated rules but to implement them in the code was not so easy. Another thing which has never been done by program developer was **undo and redo** functionality. Only towards the end of the development process it got more clear how this can be achieved. Maybe that's the reason why there is still room for improvement.

Another big challenge was to implement AI logic. It seemed to be harder than originally planned. It might be because AI logic was created from "scratch", not using already existing algorithm. And it can certainly be made better.

Things learned This project was certainly challenging and I gained a lot of practical knowledge of using Data Structures and creating algorithms. In particular creating game logic and for first time applying AI algorithms. I am glad that I used Java programming language as that is the only one I have comprehensive knowledge to be able to do this project. But it also highlighted areas which I strongly need to improve.

I have never used "**Git**" version control so that was very useful experience. I still not sure of all the things it can do, but using it in IDE and also as a backup was beneficial for future projects.

And of course time management and working towards the final deadline is also something I can learn from and certainly need to improve.