Coursework Report

Alexandra Toro Garcia
40219741@napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09103)

1 Introduction

The purpose of the following report is to provide information about a python based checkers game. This strategic board game for two players involves diagonal moves of the game pieces and captures the opponent's pieces by jumping over them [1].

It's composed of a 8x8 board with 12 pieces for each player as stated by the English and American checkers rules.
In this particular game, Red and Black pieces complete the board. Red pieces are for player one and Black pieces for two.

There is different game modes available, these include an Arcade Mode against the computer, two player mode between users and Computer vs Computer for demonstration purposes.

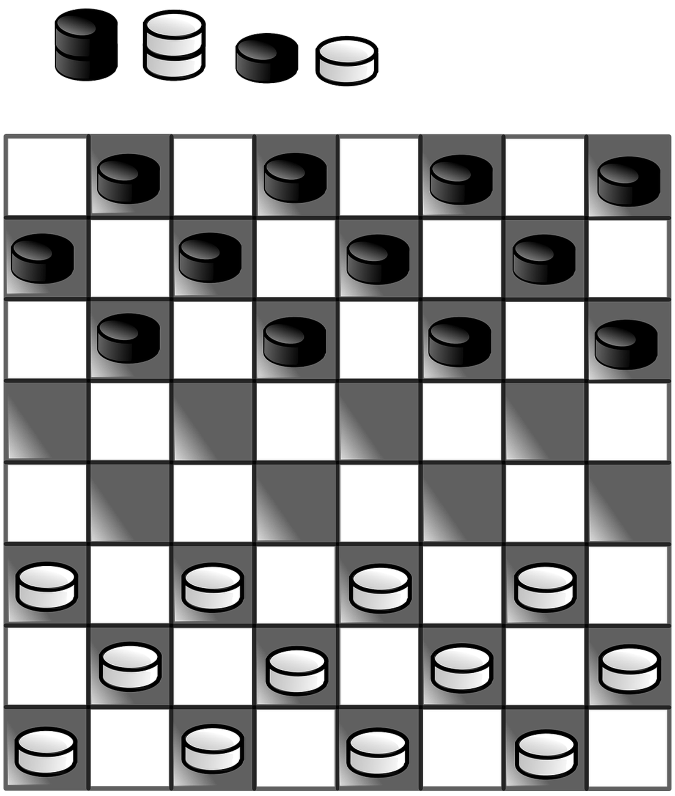


Figure 1: **Game Board** - Shows the board distribution of the pieces and the difference between Kings and men pieces.

2 Design

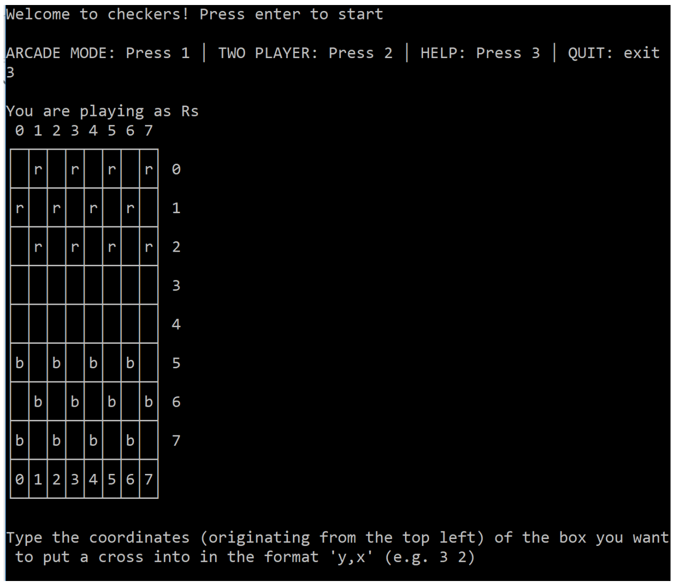
This two dimensional game has been structured using list structures into 17 functions that control the game progress. These have been created taking in consideration the different rules and specifications required (eg. undo operation, redo response,...) and the ease of play through the console to deliver the most straight-forward experience.

The following sections are aimed to outlook briefly the functionality of the game sorted by relevance.

Commands for **Bold Text**, *Italics*, and <u>underlined</u>.

2.1 Game Initialisation

The function welcomeStart() contains the main menu and initiates the game. It takes in consideration the enter of a valid input in the main menu and it will either direct the user to the chosen game mode, a help option or to exit the game. These options are controlled by if conditions but prior to any decisions get executed printGrid() will be run.



```
Welcome to checkers! Press enter to start

ARCADE MODE: Press 1 | TWO PLAYER: Press 2 | HELP: Press 3 | QUIT: exit
3

You are playing as Rs
 0 1 2 3 4 5 6 7
| |r| |r| |r| |r| 0
|r| |r| |r| |r| | 1
| |r| |r| |r| |r| 2
| | | | | | | | | 3
| | | | | | | | | 4
|b| |b| |b| |b| | 5
| |b| |b| |b| |b| 6
|b| |b| |b| |b| | 7
|0|1|2|3|4|5|6|7|

Type the coordinates (originating from the top left) of the box you want
 to put a cross into in the format 'y,x' (e.g. 3 2)
```

Figure 2: **welcomeStart()** - Option 3 was selected to show the game instructions.

2.2 Board Itineration

Under the name printGrid(), a for loop prints the two dimensional list that represents board into the console along with some reference numbers to help the user as a guide. Using a

loop at this state helps achieving a neater and shorter code to read.
This function uses help from the intToText() function to print the board in a more simple and easier to comprehend way, changing the board integers 0,1,2,3,4 to strings that represent the pieces. These integers represent empty spaces, "men" pieces of Red and Black players (1 and 2) and king pieces (3 and 4) respectively. It also calls the checkKing() and checkVictory() functions to update the board when necessary.

2.3 Turns Control

This section refers to the ArcadeMode(), twoPlayerMode() and AIvsAI() functions. They are called depending of the value of the variable "gameMode" in the welcomeStart() method. The turns are controlled by two if conditions in each function that check the "turn" variable status inside the validation functions, whether the turn's value is 0, playeR or playerRK pieces will be moved but if the value changes to 1 playerB or playerBK pieces will execute the next turn.

Listing 1: Turns control of game mode 2 in Python

```python
def twoPlayerMode():
    if(turn==0):
        validateR()
    if (turn==1):
        validateB()

```

2.4 Validation

If the user enters a value in the wrong format or selects a piece that doesn't belong to the player the validateR() or validateB() functions will stop them to continue playing until a valid input is entered (eg. x,y).
A copy of the board is stored at this part of the process that could be restored in case the user changes their mind after making a move. Depending of the type of piece selected, certain moves will be available.

2.5 Moves Control

There are 2 functions that control the moves of the pieces for each player: possibleMovesR() and possibleMovesB() deal with the "men" pieces and possibleMovesRKing() and possibleMovesBKing() deal with the kings special moves.
The structure of this functions is divided by two conditions. Since the aim of the game is to take as many pieces of your enemy as you can before they do so, the first condition checks for any pieces that can be taken and executes the move. If the conditions are met it will also take in consideration another particular action of this game called double jumps. The turn's variable value will be changed to the other player as soon as the full move is processed.

2.5.1 Double Jumps Control

A function for each player with the names doubleJumpR() and doubleJumpB() control the double jumps over enemy's pieces when these are available, no matter the type of piece. For this to happen, the function checks that there's an empty space in between and after the last piece to be taken so the jump can be executed.
To keep track of the piece's current position, coordinates are stored in two variables: "newPosX" and "newPosY". Values are added or deducted to these coordinates to check surrounding positions using expression like board[newPosY+1][newPosX-1] or board[newPosY+1][newPosX+1] to check left and right hand side of the piece's new position.

If no double jumps are made, the second possible situation to be considered is if there are no immediate pieces to be taken. In this case the user will have to decide what position will the piece have to be moved.

Players will also be made aware if there are no other moves available for the chosen piece and will get the chance to select another piece to move.

2.6 King Conversion

In the checkKing() function the "men" pieces from both players (1 for playeR and 2 for playerB) get converted to kings once they reach the opposite side of the board. The way this is done is by itinerating through the 8 rows checking for playeR pieces in the 8th row and for playerB pieces in the 1st row (Note. Numbers in this program start counting from 0). This function gets called every time the printGrid() gets executed to update the values from the pieces.

Listing 2: King Conversion code in Python

```python
def CheckKing():
    for i in range(8):
        if(board[7][i]==1):
            board[7][i]=3
            print('\n' 'WOW! Your piece is now a Red King' '\n')
        if(board[0][i]==2):
            board[0][i]=4
            print('\n' 'WOW! Your enemy has a Black King now' '\n')


```

2.7 Victory Calculation

For the checkVictory() function two boolean variables, "winnerR" and "winnerB", were declared to determinate the game champion by checking the board status. A for loop was used to do so, and a second operation using the two boolean variables declared the winner by the absence of one of the player's pieces in board.

Listing 3: Victory calculation code in Python

```python
def checkVictory():
winnerR= False
winnerB= False
for i in board:
    for j in i:
        if(j==playeR or j==playerKR):
            winnerR= True
        if(j==playerB or j==playerKB):
            winnerB=True
if(winnerR == True and winnerB == False):
    print('\n''WOOHOO! RED PLAYER, YOU ARE A WINNER! ''\n')
    print('---------------GAME OVER---------------''\n')
    welcomeStart()
if(winnerR == False and winnerB == True):
    print('\n''WOOHOO! RED PLAYER, YOU ARE A WINNER! ''\n')
    print('---------------GAME OVER---------------''\n')
    welcomeStart()

```

2.8 Artificial Intelligence Behaviour

validateAIB() is the method that controls the Arcade mode opponent's moves by creating random coordinate values to select a piece that will be moved to a fixed position able to take a piece if possible (including double jumps) or move it to an empty space if not. For demonstration purposes of the artificial intelligence achieved an AIvsAI mode has been added. The user will be notified of the piece selected by the computer and its current position.

2.9 Undo and Redo Function

This function uses the the variable values from the validation functions to be able to restore the boards either if it is to undo or redo a move. undoRedo() is used by the two player and arcade modes.

3 Enhancements

In the following section the program features that could have been implemented in the game will be discussed.
Despite the functionality of the game and its current features, the design chosen could have been more compact and neat for the move control selection. These could have been improved by creating functions to check availability of spaces. For normal pieces, the left and right side of the next row would be checked with an if condition and for the the kings, a for loop would check the four different alternatives backwards and forwards. An output with the available positions numbered would then be displayed for the user to choose between them. This feature would make the game fast-paced since a single value input would be entered rather than coordinates. The functions would have been called and reused inside the possible moves methods for both players.

Another potential feature to the program would have been the Artificial Intelligence (AI) difficulty capacity where the user could have selected between three gaming levels: newbie, talented and expert. It was considered at the time, however due to time schedule only one level of difficulty is available where the user has a good chance to proclaim the winner.

For efficiency purposes a restart commands could have been added so the user could start the game again if they would wish to. This could have been implemented as a condition wherever there was a user input calling the welcomeStart() function to do so.

To keep records of the game progress, a history feature could have been useful. For example, if the users are beginner players, it would allow them to look back and learn through the records. The way this would have been implemented is by storing the changes on the board into a list which is effectively a list of lists. This would be placed into a function that could be called if the correct input is entered after a move is made or the game is over.

4 Critical Evaluation

This section is aimed to analyse the program's performance from an objective perspective taking in consideration its strengths and weaknesses.

The AI designed for this game could be considered as a minor weakness. On the Arcade mode, AI could present to be a challenge for an amateur player however, it would not be a fair opponent against an experienced one due to its weak prioritisation of piece selection and moves. These is manifested through its behaviour on the AIvsAI mode and it caused by the use of random numbers to select pieces and new positions.

The user's game experience is simple and direct, the text has been divided across the game to keep the user aware of the process but without bombarding them.

The program's structure of the functions has been divided with specific purposes keeping the code maintenance in my mind. The names of functions and variables used have been selected to be instantly recognisable. This all together, makes the code accessible and fluent. Nonetheless, the condition operations used in some of the functions can seem complicated and too long to comprehend and some functions could have been more widely applicable throughout the game.

5 Personal Evaluation

This evaluation section will focus on the performance and learning commitments achieved.
The creation and design of this game on high-level programming language not used before by the programmer supposed a challenge solved by research and practice. The handling the data to work with the values across functions was a challenge at the beginning but was resolved with the use of global variables. This variable is specific for python and it was found extremely useful. Although the final program has functions that could have been more general and restricting conditions for the pieces that could have been shorter, this was reconsidered and modified multiple times to achieve the most clear and user-friendly result.

To conclude, a new programming language was learned as well as the important use of lists in a board focused game and how to create a basic AI program.

References

[1] "English daughts association."