Coursework Report

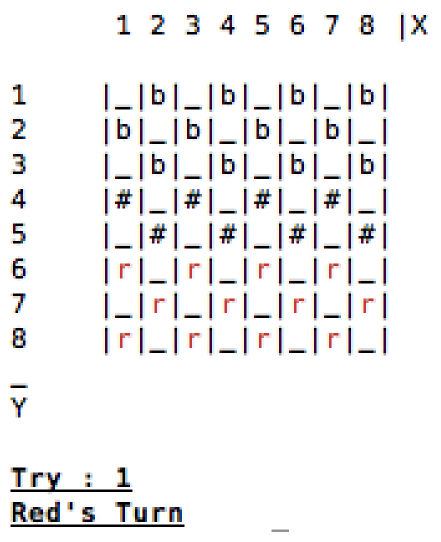
Cool Student
40209068@live.napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures (SET09117)

1 Introduction

This program is a game of Draughts. It was decided that python was a good language to use to make the program, this decision had its ups and down. When you start the program it tells you how to play so no one is confused at the start and the pieces are colour coded so its easier for the users to see what piece is theirs. In the program you can move pieces, take pieces and the computer will not let you make a wrong move out of the play area or an illegal move.

2 Design

For the design of this python program it was decided that a list of lists was the best way to show the data. Originally it looked clunky and not easy to understand but after some work was put into it it became easier to understand and use for the users.



```
        1 2 3 4 5 6 7 8 |X

1      |_|b|_|b|_|b|_|b|
2      |b|_|b|_|b|_|b|_|
3      |_|b|_|b|_|b|_|b|
4      |#|_|#|_|#|_|#|_|
5      |_|#|_|#|_|#|_|#|
6      |r|_|r|_|r|_|r|_|
7      |_|r|_|r|_|r|_|r|
8      |r|_|r|_|r|_|r|_|

_
Y

Try : 1
Red's Turn
```

Figure 1: **Python board** - This is the board that is shown to the players

What has been done to the original list of lists was certain values were put in the list to mean different values. After when the computer sees these values it replaces it with a value easier for the users to understand and in reds case colours it red to make it even easier to see. At the bottom of the picture you also see Try and Turn, the Try shows how many try's the player has had and they have three attempts to make a move that is not going to an occupied spot or it makes the game a draw. The turn is an easy way to show who's turn it is. It was decided to make these bold to make it stand out and make sure the right person took their move at the right time.

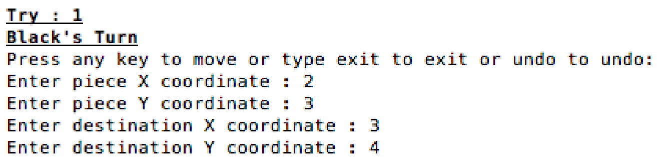
Listing 1: print board function in Python

```
1 def print_board(board):
2     print("\n    1 2 3 4 5 6 7 8  |X\n")
3     for i in range(GRID_HEIGHT):
4         print(i + 1, "    |", end="")
5         for j in range(GRID_WIDTH):
6             print(board[i][j] + "|", end="")
7         print("")
8     print("_\nY\n")
```

This function here is how the board is displayed. This is where we see how the X and Y coordinates are shown. This is also where we see GRID HEIGHT and GRID WIDTH which is set at the top of the program outside of any def. At line 4 you see i + 1, this was decided because as people we like to start at 1 but the computer starts at 0 all the time so this is done and then 1 is taken away from users inputs to get the correct coordinates.

3 Enhancements

If this program had more time to be designed there would of been more focus on the user inputs. Currently when the program starts it asks if it is 2 player game or 1 player with an AI, after this is done on both 2 player and AI it will ask if it wants to exit, undo or move. if nothing is entered it will go to the move. The program will then ask what the coordinates are that the user wants to choose in 4 separate questions.



```
Try : 1
Black's Turn
Press any key to move or type exit to exit or undo to undo:
Enter piece X coordinate : 2
Enter piece Y coordinate : 3
Enter destination X coordinate : 3
Enter destination Y coordinate : 4
```

Figure 2: **Take turn** - This is what is shown when a player puts values in

This could of been improved if it only asked 2 questions instead of 4 for the coordinates by asking for the two source values and then the two destination values after. This could decrease time of play and that could make it more enjoyable for the players not having to spend ages playing the game. This could be done or a GUI could be put in place. This would have made movement easier as it could be implemented to use the mouse instead of the keyboard. This would get rid of the need to work out the coordinates and make game play even faster. This would also make it look better for the users as you could have a proper looking board and pieces that look like actual pieces.

The jump system could also of had more work to it. Currently when you take a piece you should be able to move again if there is an available piece to take. In the current system after a piece is taken the user gets asked if they can move again and if they say yes they could move again. This makes the game very honesty based but if the two players were both playing next to each other they would know if the other one was cheating.

The AI is probably the part of the program that needs the most work. Currently it makes random values and and checks if they work through all the rules of the program. It goes through this again every time it shows a wrong value. It will find a value sometimes but sometimes it loops so much that it crashes the whole program.

Listing 2: error displayed

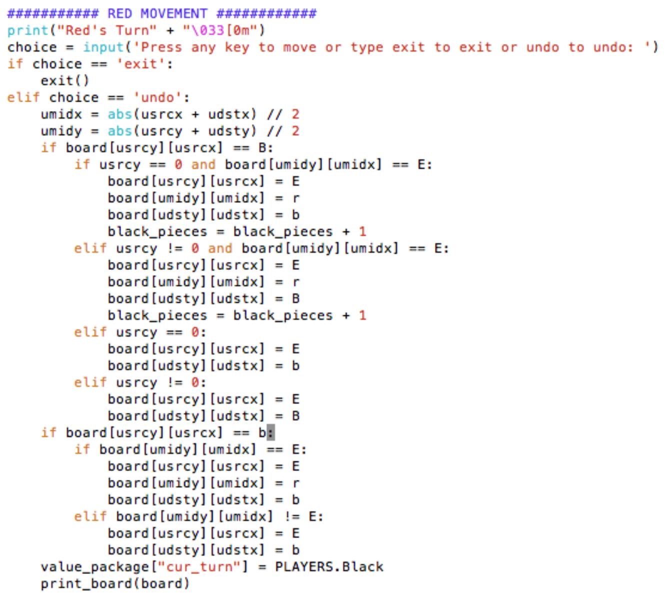
```
RecursionError: maximum recursion depth exceeded while ↩
    calling a Python object
```

This is happening because it overloads the program after making so many values that were not accepted. If there was more time the program could make a random x coordinate and check all the respective y coordinates. If it then find an acceptable square it will then check the surrounding coordinates to see if there is a acceptable destination square to go into and then move the piece.

4 Critical Evaluation

This program has many good parts but equally could be improved in many parts as well. The user input is very good because if you input a number outside of the allowed number range it tells the user and makes them try again. Same with if the user inputs a letter or nothing at all it will tell them to input a number in the the range and make them try again. This helps miss clicks and make sure it doesn't break the program by accident. The rules for the actual game are also very good as they make sure users can only move their own piece and can only move legal moves and force them to stay in the play area. The rules section does everything they need to do apart from after you take a piece. Instead of checking the surrounding blocks if there is another piece to take and space to take it the program just asks the user if there is more pieces to take. This is also bad because this means that the user could just move another piece completely and not take a piece. If the program had more time to be codded this would of been sorted by setting the destination to the new source block and only asking for the new destination after checking if there is space to go.

The undo function is another part that works well. After a piece is moved and or taken it saves to the undo source and destination but reversed as the destination was the source before and the other way round for the other values. The users are asked before they take their turn if they want to move exit or undo, if they say undo it moves all the pieces back even if its a jump and puts the piece that was jumped back as well. It will then change back to the players turn that just went so they can move their turn.The only draw back to this undo function is that if a king piece is taken then the player types undo, it turns the king piece into a pawn piece again even though it was a king before.



```
########### RED MOVEMENT ############
print("Red's Turn" + "\033[0m")
choice = input('Press any key to move or type exit to exit or undo to undo: ')
if choice == 'exit':
    exit()
elif choice == 'undo':
    umidx = abs(usrcx + udstx) // 2
    umidy = abs(usrcy + udsty) // 2
    if board[usrcy][usrcx] == B:
        if usrcy == 0 and board[umidy][umidx] == E:
            board[usrcy][usrcx] = E
            board[umidy][umidx] = r
            board[udsty][udstx] = b
            black_pieces = black_pieces + 1
        elif usrcy != 0 and board[umidy][umidx] == E:
            board[usrcy][usrcx] = E
            board[umidy][umidx] = r
            board[udsty][udstx] = B
            black_pieces = black_pieces + 1
        elif usrcy == 0:
            board[usrcy][usrcx] = E
            board[udsty][udstx] = b
        elif usrcy != 0:
            board[usrcy][usrcx] = E
            board[udsty][udstx] = B
    if board[usrcy][usrcx] == b:
        if board[umidy][umidx] == E:
            board[usrcy][usrcx] = E
            board[umidy][umidx] = r
            board[udsty][udstx] = b
        elif board[umidy][umidx] != E:
            board[usrcy][usrcx] = E
            board[udsty][udstx] = b
    value_package["cur_turn"] = PLAYERS.Black
    print_board(board)
```

Figure 3: **Undo function** - This works out if the last move was a jump and move pieces back respective to what was just done

Out of all these parts of the program the AI needs the most work and is defiantly early stages in its readiness to be a proper AI that works how it was intended.

5 Personal Evaluation

During the process of making this program I have learned a lot about the python language and how to use it effectively. I have come across many problems with my code and have managed to fix a lot of them. During the start of the program my initial aim was general movement. I decided for the board to have a list of lists as this would be the best way to display the game and the users could understand it easily. Then I got the user inputs working so they chose a source block and a destination block at the start it didn't matter what blocks you chose it would allow it. Then for the actual movement I had previously made the board with different letter values in it to represent reds and blacks. All I did was set the source value to equal blank and the destination to equal a piece. This then made me think about the rules and limiting what pieces you can move.

Listing 3: Choose correct piece in Python

```
if board[srcy][srcx] == b or board[srcy][srcx] == B:
    print('Please choose your own piece')
    print_board(board)
    return
```

This is the point where I had the idea to change the board slightly. What was changed was the blank squares that pieces can never go on to was changed to something different than the blank squares that pieces can move on to and just an if for both source and destination that if they equal a piece like this that its not allowed. This then brought forward the next problem of how to limit how pieces move only sideways and how many they can move. This was achieved by seeing if the destination value was more than 2 in both positive and negative values for both x and y to limit the most a piece can move into a jump.

Listing 4: Moves cant be more than a jump in Python

```
if dstx > srcx + 2 or dstx < srcx - 2:
    print('move too large')
    print_board(board)
    return
```

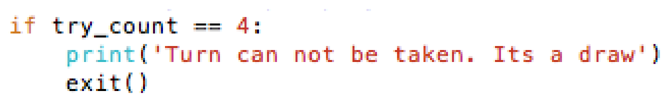
The last part of this was to check if the two x values were the same and same with the y's this stops straight movement into the one square that is 2 away from the source piece.

Listing 5: Moves cant be straight in Python

```
if dstx == srcx
    print('move diagonally')
    print_board(board)
    return
```

After all this was done my focus was on taking a piece and kinging a piece. Kinging was easy all that was needed was a if near the end so that everything is checked that its an allowed move and if the destination y equaled the end of the board for that colour it was kinged. Taking a piece was a little harder though. I had to check the piece being taken was the correct colour so that you cant take your own piece and what was taking the piece. This was needed to be check as if you take a piece and your destination equaled the end your piece should be kinged. This meant there was an if inside the taking a piece part that checked if it was a pawn and it was landing at the end of the board that changed the pawn to a king and if not it was just a pawn still. This also meant there has to be a part that just keeps it a king if a king is taking a piece.

Inside the jump function there is also small bits of code that are working out how many pieces are left and once there are no more pieces left it will display who won. This can happen as the only way to win a game is to take all the pieces. This also brings to question how can a draw be achieved. What I implemented here was a count on not being able to move because there is a piece in the way and if this is shown 3 times in a row then it will force a draw and end the game.



```
if try_count == 4:
    print('Turn can not be taken. Its a draw')
    exit()
```

Figure 4: **Draw function** - Once it hits more than try 3 it ends the game

6 Conclusion

In conclusion this coursework has taught me a lot and made me think. I feel I have done well for my limited knowledge of the language and have improved it at the same time so now I can use it to a higher standard. I know I have much to learn and this coursework could of been improved a lot with a higher knowledge of the language and more time but I am happy with what I have done.