Algorithms and Data Structures Coursework 1

Ross Chapman
40209091@live.napier.ac.uk
Edinburgh Napier University - Algorithm's and Data Structures (SET09117)

Keywords – Python, Checkers, Game, Data Structures, Board Game,

1 Introduction

The Problem The task was to create a checkers game that allowed for two players to play against each other or an AI, have an undo function and be able to record play. It is meant to be able to be played on command line. It cycles through each turn and if the player makes an error it allows that player to try again. Each player has 12 pieces and whoever loses all their pieces will lose the game.

```
    0    1    2    3    4    5    6    7
0 ['E', 'b', 'E', 'b', 'E', 'b', 'E', 'b']
1 ['b', 'E', 'b', 'E', 'b', 'E', 'b', 'E']
2 ['E', 'b', 'E', 'b', 'E', 'b', 'E', 'b']
3 ['E', 'E', 'E', 'E', 'E', 'E', 'E', 'E']
4 ['E', 'E', 'E', 'E', 'E', 'E', 'E', 'E']
5 ['w', 'E', 'w', 'E', 'w', 'E', 'w', 'E']
6 ['E', 'w', 'E', 'w', 'E', 'w', 'E', 'w']
7 ['w', 'E', 'w', 'E', 'w', 'E', 'w', 'E']
```

Figure 1: **The board** - This is what the board looks like

2 Design

Board The way the board is made is that it is a list of lists so it is easy to reference a specific board square. Such as if you want to reference coords, y-coord = 4, x-coord = 5 then you would reference that with Pieces[4][5] which would get whatever piece is in that coordinate. This also means it is easy to change what is in which coordinate by using an if statement to check what piece is where then changing what is in that coordinate.

Win condition The way the win condition works is that it searches through the list of lists and takes a count of how many white pawn and king pieces currently on the board and same with the black pieces and from this you get an accurate total that is always correct to what is on the board. This means there can never be a miss-communication between the score and what is on the board.

Listing 1: Win condition in Python

```
1  for i in range(100):
2      black_pawn_total = sum(x.count("b") for x in Pieces)
3
4      white_pawn_total = sum(y.count("w") for y in Pieces)
5
6      black_king_total = sum(v.count("B") for v in Pieces)
7
8      white_king_total = sum(z.count("W") for z in Pieces)
9
10     white_total = white_king_total + white_pawn_total
11
12     black_total = black_king_total + black_pawn_total
```

Movement The way the movement works makes good use of the list of lists board as it checks that the correct piece has been selected then uses the coords of the destination to set that destination to the selected piece. There are also if statements that limit how far they can move like no further than 2 squares per jump/move. Also pawns cant move backwards but kings can.

Taking a piece The way that pieces are taken is that a mid value is taken from between the source coords and the destination coords. This mid value is checked by an if to check if it is either a pawn or a king as both can be taken by any piece. Then the square that the mid value references is changed to an empty piece

Kings When a pawn gets to the other side of the board it gets upgraded to a king which enables the piece to move back across the board and continue taking pieces however a pawn can still take a king piece

Listing 2: Making a king in Python

```
if dst_y == 7:

        print("Congrats you have made a king Piece")

        Pieces[src_y][src_x] = SYMBOL_EMPTY

        Pieces[7][dst_x] = SYMBOL_KINGBLACK

        return
```

The loop In the main there is a loop that goes for 100 turns and it calls the move function and checks the current total of the pieces on the board and if one of them is 0 then it decides the winner and asks if you want to end the game so that you can.

3 Enhancements

Undo-system Sadly there was not time to implement an undo system but this would be useful if a player made a move they did not mean to.

Multiplayer online It would be beneficial to implement some way to play with someone online so that both players don't need to be there. This would also allow some sort of leader-board to be implemented which would allow some competition to be fostered. This would allow long distance Friends to play against each-other

Auto-take a second piece In the rules it states that if you take a piece and can take another piece you must do this. This function would make it so that the user would not need to think about this and would make it more streamlined.

Write to file It could have been done that you could use a list of lists to record the coordinates and then write that to a file then load that in backwards so that you could use those values to work into the undo function.

4 Critical Evaluation

Win-condition This works well because it is always accurate to the board so it can not often be wrong. However it does not allow for different colour pieces to be used as it has to be black and white pieces.

Taking a piece The way the taking of a piece works is that it checks what the piece that is being taken is and only goes through with the taking if it is the correct piece to take. However you can jump over an empty space which is incorrect.

Movement logic The way the movement error handling works is that it does not allow a piece to jump further than 2 spaces and it does not allow a pawn piece to move backwards. Also if a player makes one of these mistakes they get to try again.

Listing 3: Movement error handling in Python

```
if Pieces[src_y][src_x] == SYMBOL_BLACK and src_y > dst_y:

        print("Pawn cannot move backwards")
        return move(Turns, white_total, black_total, Pieces)
    if src_x == dst_x:
        print("You cant move a piece verticaly")
        return move(Turns, white_total, black_total, Pieces)
    if src_y == dst_y:
        print("You cant move a piece horizontaly")
        return move(Turns, white_total, black_total, Pieces)
    if dst_x > src_x + 2:
        print("You cant move farther than 2 spaces in a jump")
        return move(Turns, white_total, black_total, Pieces)
    if dst_x < src_x - 2:
        print("You cant move farther than 2 spaces in a jump")
        return move(Turns, white_total, black_total, Pieces)
    if dst_y > src_y + 2:
        print("You cant move farther than 2 spaces in a jump")
        return move(Turns, white_total, black_total, Pieces)
    if dst_y < src_y - 2:
        print("You cant move farther than 2 spaces in a jump")
        return move(Turns, white_total, black_total, Pieces)
```

5 Personal Evaluation

Learned In this coursework i learned that error handling is important as it will wreck a game if anything goes wrong. In order to make the board i used a list of lists so that the board is permanent and i learned that having a correct data structure to store values can be the corner stone of an app

Challenges Some challenges i faced was to get a consistent way to record "score" and the way i solved it was to actually count the pieces on the board then add those up and since it loops through then it will always be accurate to that specific iteration which means the moment someone wins it is acted upon.

Methods When i was deciding what data structure to use for the board i was beginning to panic but then i mentally took a step back and just thought about the problem and came to the conclusion that an array would be good but for both columns and rows i would need an array of arrays but then i realized that that is basically what a list of lists is so i used that

Performed I feel i performed poorly to be honest, however i feel i have learned from my mistakes in how i go about understanding what structures i have learned and how best to use them.