EasyAI Checkers

PROJECT 1

Greydi Lora, Jamesly Calestin, Salman Ahmed Barry | CAP 4630 | 6/15/2022
Documentation By Greydi Lora

Group Assignment

Design By Jamesly Calestin

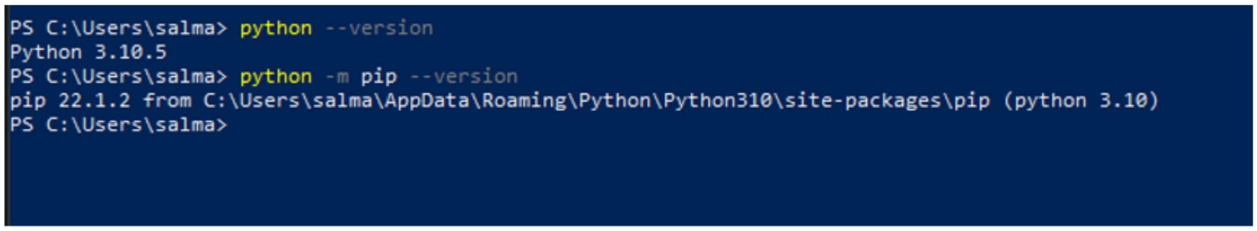
Coding By Salman Ahmed Barry

Documentation By Greydi Lora

For this assignment, we divided the tasks into three parts. The design implementation was constructed by Jamesly Calestin which included the pseudo code needed to plan and organize the code and a flowchart to visualize it. Salman Ahmed Barry assembled the program for the checkers game and added all the functions and code needed to execute the game. Greydi Lora inscribed the documentation detailing the processes, design decisions, and more.

Setting Up Coding Environment

We started off by installing Python and Git on all of our computers. Python was installed on a windows 10/11 machine using the EXE provided by the python website and the bin was added to the system environment path automatically. (the Microsoft store version had some unexplainable glitches thus had to start from scratch, Additionally command prompt was having permission issues, thus window powershell)



```
PS C:\Users\salma> python --version
Python 3.10.5
PS C:\Users\salma> python -m pip --version
pip 22.1.2 from C:\Users\salma\AppData\Roaming\Python\Python310\site-packages\pip (python 3.10)
PS C:\Users\salma>
```

^Version of dependencies on windows powershell

Using the professors (extremely helpful) guidelines we were easily able to copy and paste the commands and install the required dependencies.

IDE SETUP

Next was the choice of IDE and basically we needed something that would:

1. Run on all our group members computer
2. Was modifiable to run python with the dependencies for testing purposes
3. Was fairly easy to install

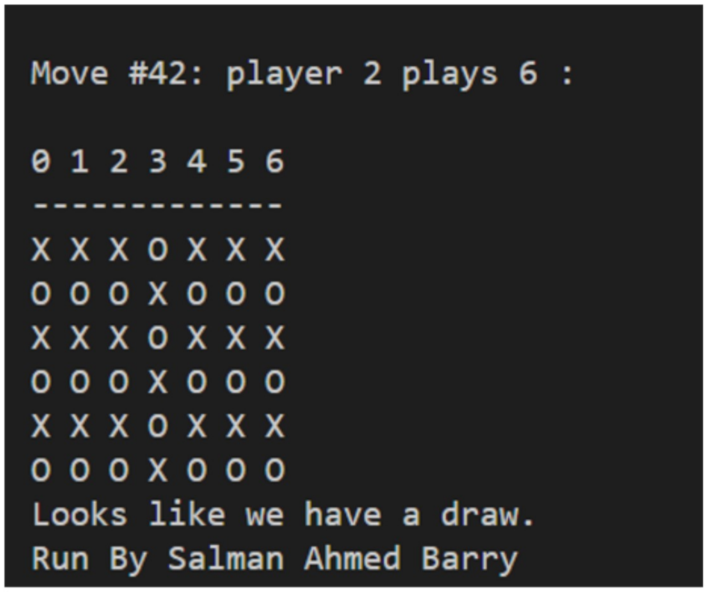
Thus, the choice came down to either Atom Or Visual Studio Code

In the end we decided to use **Visual Studio Code** as it fulfilled all the requirements and easy integration with XCODE for our Mac OS Group members, and there were packages that made the editing and quick running of python very easy and convenient.

TESTING

Once the IDE and EasyAI dependencies were installed we had to test our environment using the provided EasyAI implementations of TicTacToe and Connect four, which was not an issue as they ran flawlessly.

Connect 4

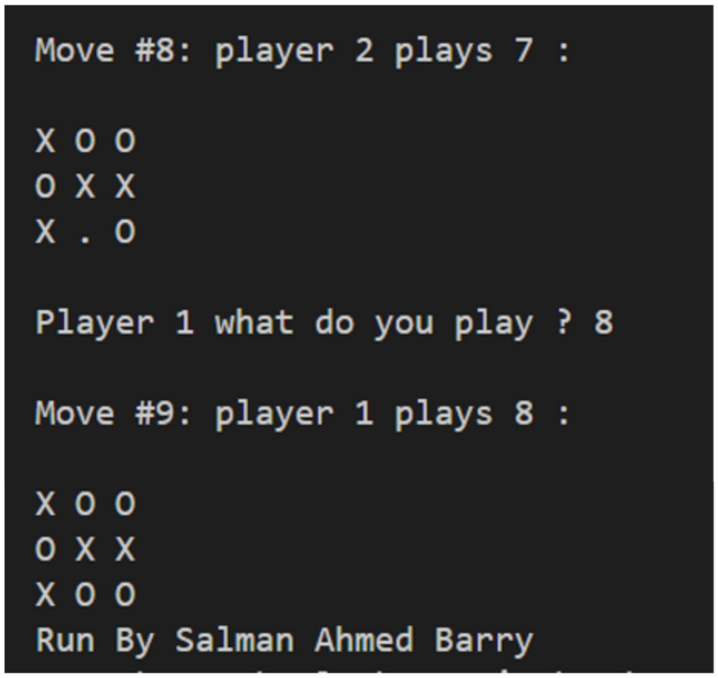


```
Move #42: player 2 plays 6 :

0 1 2 3 4 5 6
-------------
X X X O X X X
O O O X O O O
X X X O X X X
O O O X O O O
X X X O X X X
O O O X O O O
Looks like we have a draw.
Run By Salman Ahmed Barry
```

^*Conecctfour.py run on visual studio code*

Tic-Tac-Toe



```
Move #8: player 2 plays 7 :

X O O
O X X
X . O

Player 1 what do you play ? 8

Move #9: player 1 plays 8 :

X O O
O X X
X O O
Run By Salman Ahmed Barry
```

^*TictacToe.py run on visual studio code*

Design

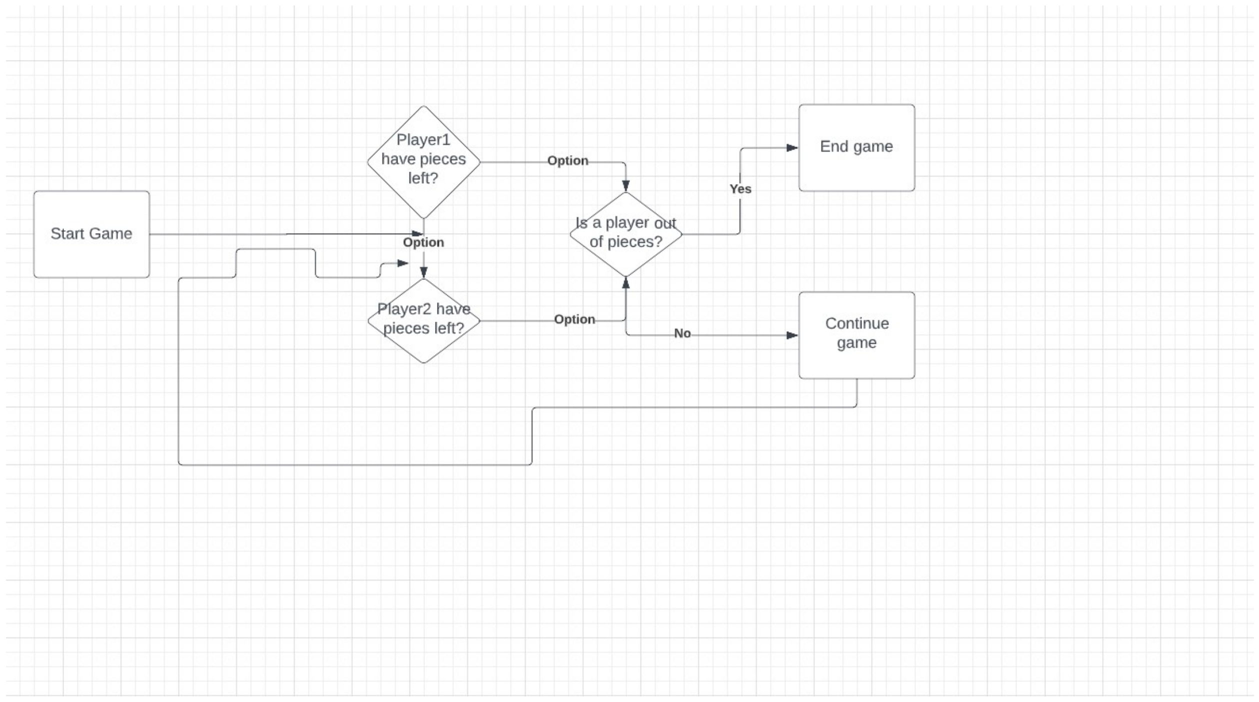
For the design process, the four functions were planned based on the rules of the game. The make_move() function was responsible for assigning the new positions as the current position.

make_move(): When implementing this function, we must keep in mind that the player can only move diagonally. This function will be the driver behind our pieces on the checkerboard. The player is able to jump an opponent if there is an empty space diagonal to the opponent's piece in either direction.

lose(): This function fires if all of our pieces or our opponents pieces are clear off the checkerboard.

is_over():this function runs in tandem with the "lose" function, firing off when the game is over

scoring():if your opponent's piece is taken off the checkerboard, a point gets added to your score. if your piece gets taken off the checkerboard, a point gets added to your opponent's score.


Start Game
Player1 have pieces left?
Option
Player2 have pieces left?
Option
Option
Is a player out of pieces?
Yes
End game
No
Continue game

^*Flowchart Prototype 1*

How we plan to implement easyAi

```
from easyAI import TwoPlayerGame, Human_Player, AI_Player, Negamax
class GameOfChips( TwoPlayerGame ):
def __init__(self, players=None):
      self.players = players
      self.pile = 24 # start with 24 pieces e
      self.current_player = 1 # player 1 starts
```

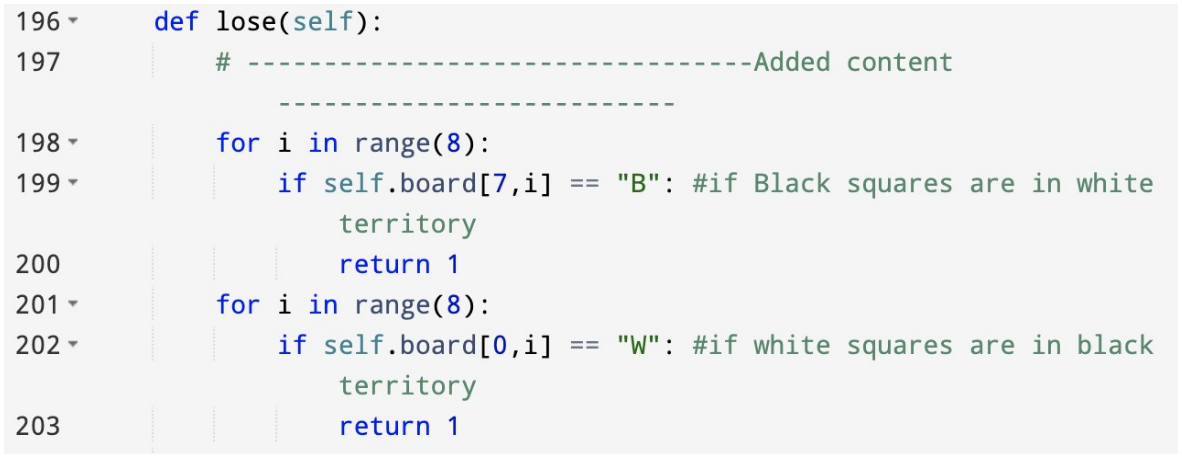
the board is implemented using an 8*8 matrix. Within this matrix, black and red squares are created using two arrays: even and odd. The even is the black and odd is red. The colors are alternated to fill in the matrix, creating the checkerboard. The players are only allowed to move across and land on black (or even) squares. From here, each player takes turns making a move until one of the players has no more chips left. When the opposing player makes a move, the ai checks for the best possible response to the move such as: whether it is possible to make multiple jumps in one move or not. If so, the AI takes advantage. Otherwise, a simple jump is made. The win condition is an infinite loop that checks whether any player runs out of moves OR if an opposing player's piece has moved into the other territory. If true for one player at any given time, it is considered GAME OVER!!!

Coding

Inside of the make_move() function, there is a set of two arrays that holds the new positions for each player. The for loop iterates through the array to search for the pieces and appends it to a list. The new position assigned is now the current position.

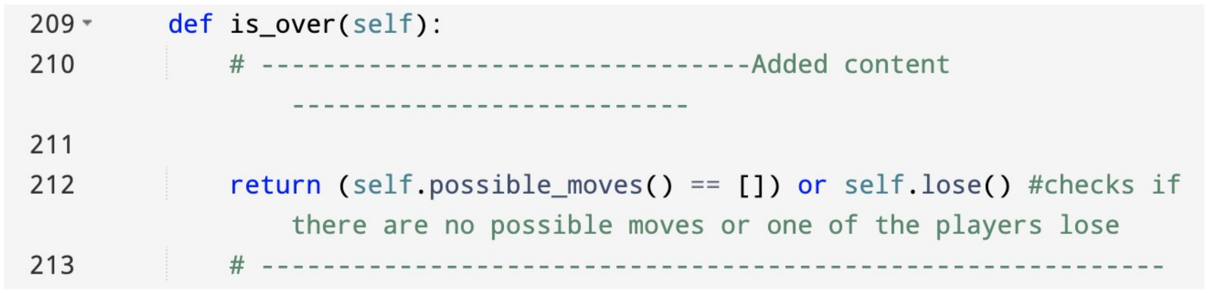
```python
    def make_move(self, pos):
    # ---------------------------------Added content --------------------------
        newpos_1 = [] #list that will hold the new positions for P1
        newpos_2 = [] #list that will hold the new positions for P2
        for i in range(8):
            for j in range(8):
                if pos[i,j] == "B": #Looking for B peices in the checker board array
                    current_pos=(i,j)
                    newpos_2.append(current_pos) #appending list with new positions
                if pos[i,j] == "W": #Looking for W peices in the checker board array
                    current_pos=(i,j)
                    newpos_1.append(current_pos)#appending list with new positions
        self.players[0].pos = newpos_1 #new player positions for player 1 to return to possible moves
        self.players[1].pos = newpos_2 #new player positions for player 2 to return to possible moves
```

The lose() function is called whenever a player is on the other side of the opponents which means that the opponent lost the game. In other words, if the black pieces are in the white space, then the black piece wins. If the function is called, a message prints stating the winner of the game. This function is also called if all pieces are cleared from either player.



```
    def lose(self):
        # --------------------------------Added content --------------------------
        for i in range(8):
            if self.board[7,i] == "B": #if Black squares are in white territory
                return 1
        for i in range(8):
            if self.board[0,i] == "W": #if white squares are in black territory
                return 1
```

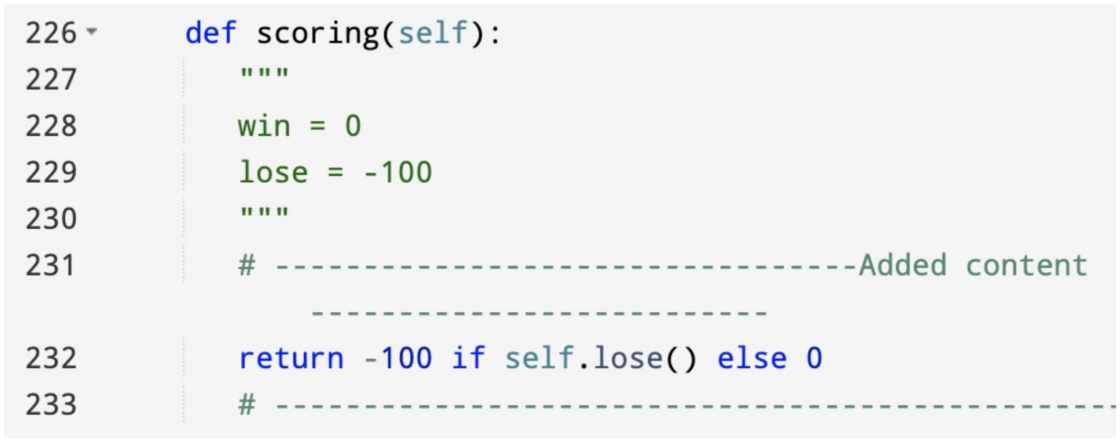
The is_over function determines whether the game ends based on if there are no moves left or if either player loses the game. This function uses the array that carries the number of moves and the lose function to dictate the ending of the game.



```
    def is_over(self):
        # --------------------------------Added content --------------------------

        return (self.possible_moves() == []) or self.lose() #checks if there are no possible moves or one of the players lose
        # -----------------------------------------------------------
```

The scoring function keeps track of the scores throughout the game. Each piece that is taken out of the checkerboard by your opponent will be added to the opponents score and vice versa.



```
    def scoring(self):
        """
        win = 0
        lose = -100
        """
        # --------------------------------Added content --------------------------
        return -100 if self.lose() else 0
        # ------------------------------------------------
```

Future plans

1. Better Ai recognition and priority for taking opposing players pieces, recognizing kill chains and basically prioritizing the value of killing a player over attempting to crown its pieces
2. Human player option
3. A proper GUI
4. Sound Effects
5. Proper text based prompts for AI difficulty

Git Integration

File has been uploaded to GITHUB, however the .py file had to be uploaded manually as i was having an issue uploading it through **git push -u origin main**

URL to github: https://github.com/namlasyrrab/Checkers_easyAI