Florida Atlantic University

CAP4630: Introduction to Artificial Intelligence

Project 1 : EasyAI Implementation

By:

Charles Emile Seaman - Architect

John Salsinger - Developer

Steffano Martinez - Reporter

Table of Contents

Part 1: Work Allocation and Setup **2**

Part 2: Running tic-tac-toe.py and connect-four.py **3**

Comments: 4

Part 3: Implementing a modified version of Checkers **5**

Completing the Assigned Functions: 5

Other Design Decisions: 6

Testing/Running our Solution: 7

Additional Remarks **8**

Documented Limitations: 8

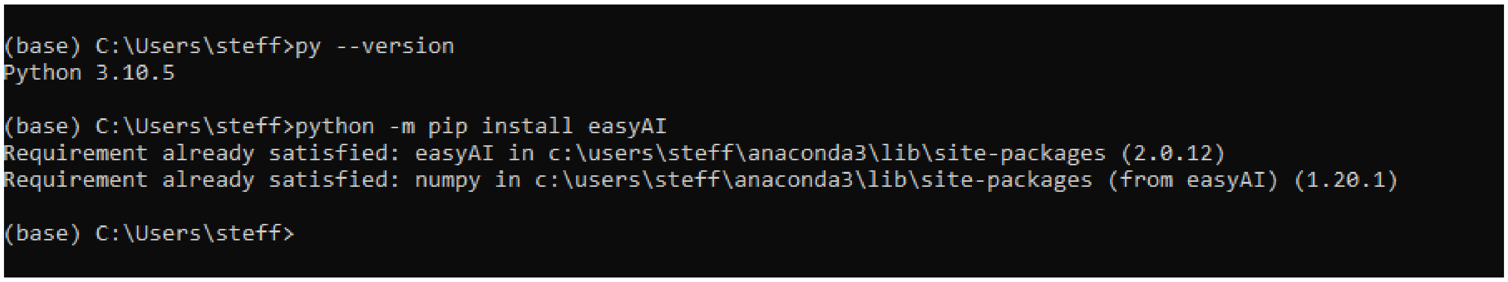
Future Improvements: 8

Final Thoughts: 8

Part 1: Work Allocation and Setup

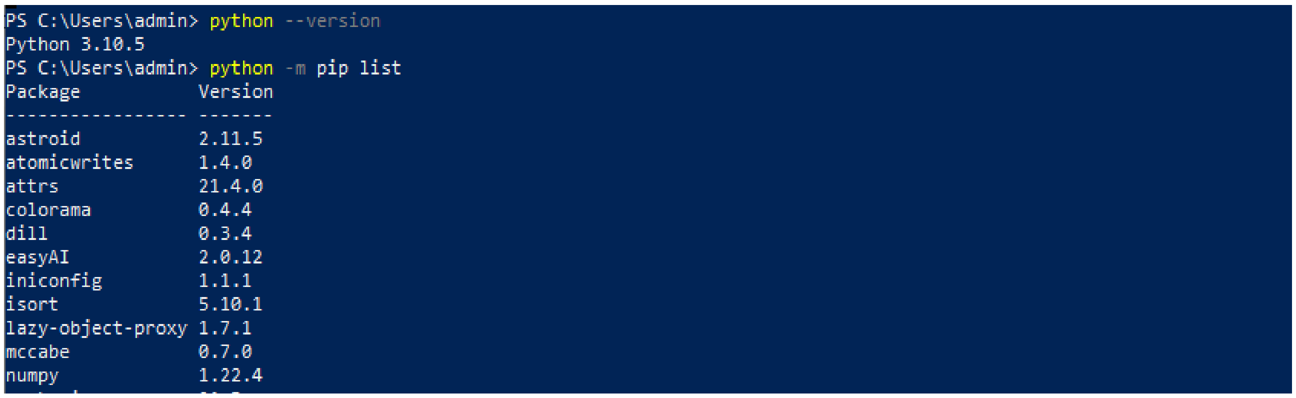
We decided that Charles would come up with the logic for the functions, John would code it, and I (Steffano) would get everything together and organize the report. Even though we split the work up, we each tried the assignment on our own and gave input in every aspect since it was necessary to understand how the EasyAI library worked and what the prewritten functions in the skeleton did. As instructed, all three of us got Python installed and got EasyAI and Numpy setup:

Evidence from Steffano (Reporter):



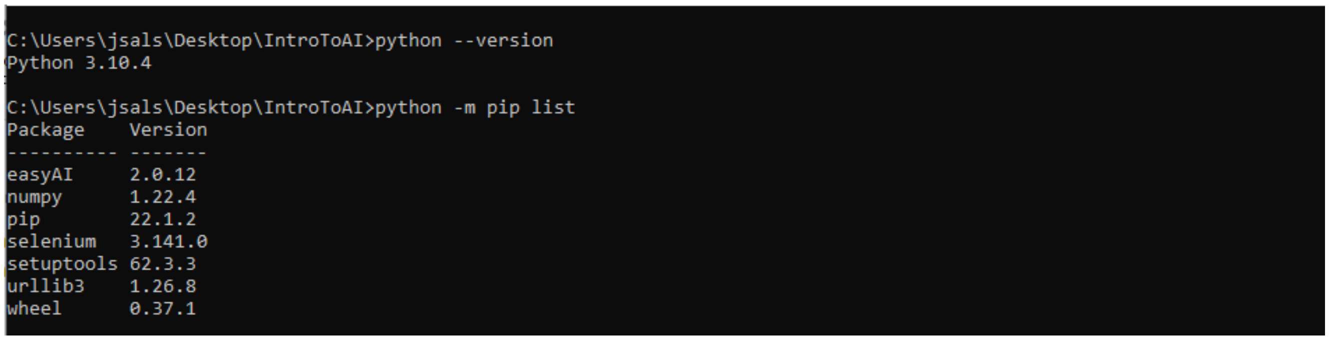
```
(base) C:\Users\steff>py --version
Python 3.10.5

(base) C:\Users\steff>python -m pip install easyAI
Requirement already satisfied: easyAI in c:\users\steff\anaconda3\lib\site-packages (2.0.12)
Requirement already satisfied: numpy in c:\users\steff\anaconda3\lib\site-packages (from easyAI) (1.20.1)

(base) C:\Users\steff>
```

Evidence from Charles (Architect)



```
PS C:\Users\admin> python --version
Python 3.10.5
PS C:\Users\admin> python -m pip list
Package           Version
----------------- -------
astroid           2.11.5
atomicwrites      1.4.0
attrs             21.4.0
colorama          0.4.4
dill              0.3.4
easyAI            2.0.12
iniconfig         1.1.1
isort             5.10.1
lazy-object-proxy 1.7.1
mccabe            0.7.0
numpy             1.22.4
```

Evidence from John (Developer)

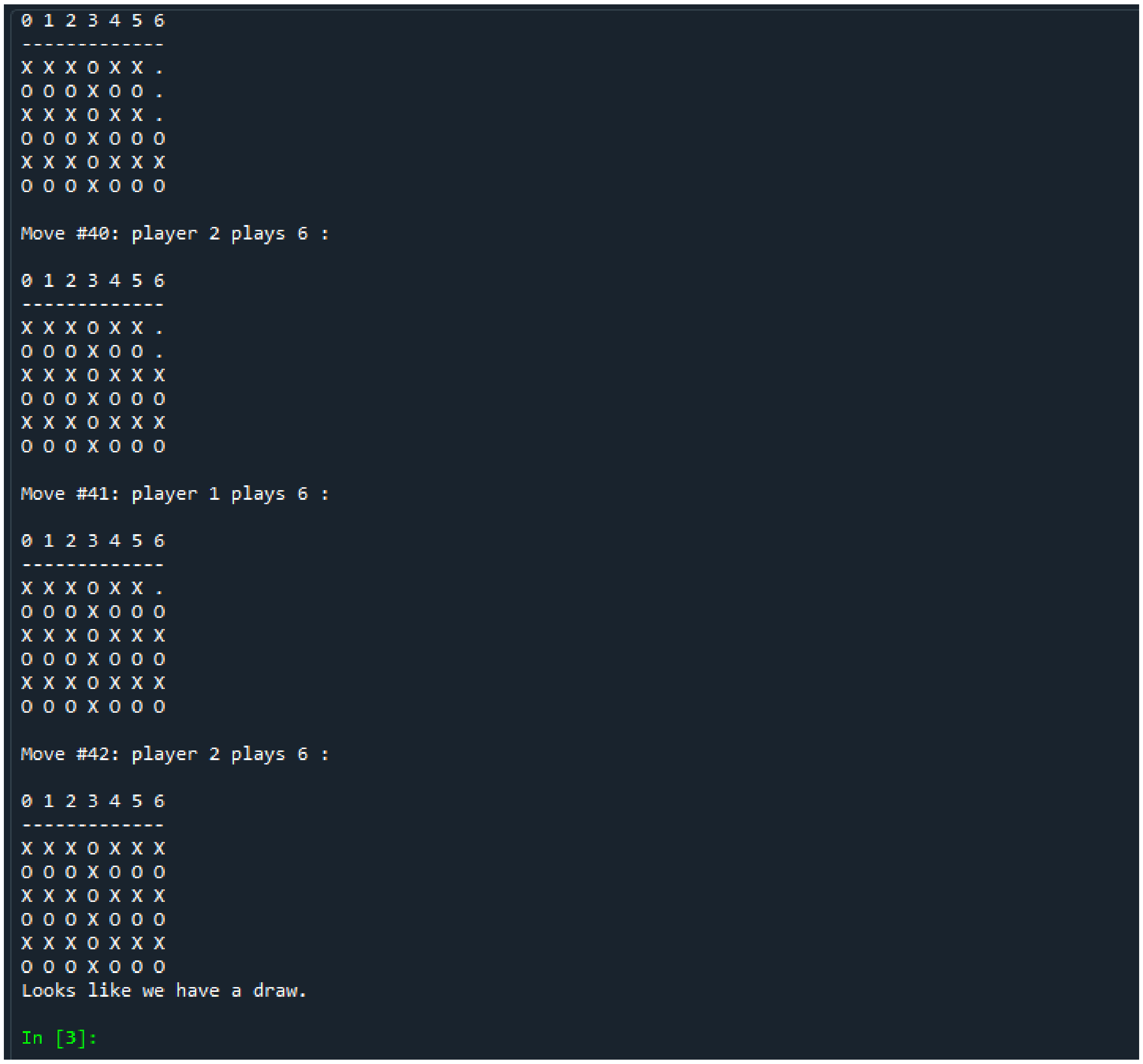


```
C:\Users\jsals\Desktop\IntroToAI>python --version
Python 3.10.4

C:\Users\jsals\Desktop\IntroToAI>python -m pip list
Package    Version
---------- -------
easyAI     2.0.12
numpy      1.22.4
pip        22.1.2
selenium   3.141.0
setuptools 62.3.3
urllib3    1.26.8
wheel      0.37.1
```

Part 2: Running tic-tac-toe.py and connect-four.py

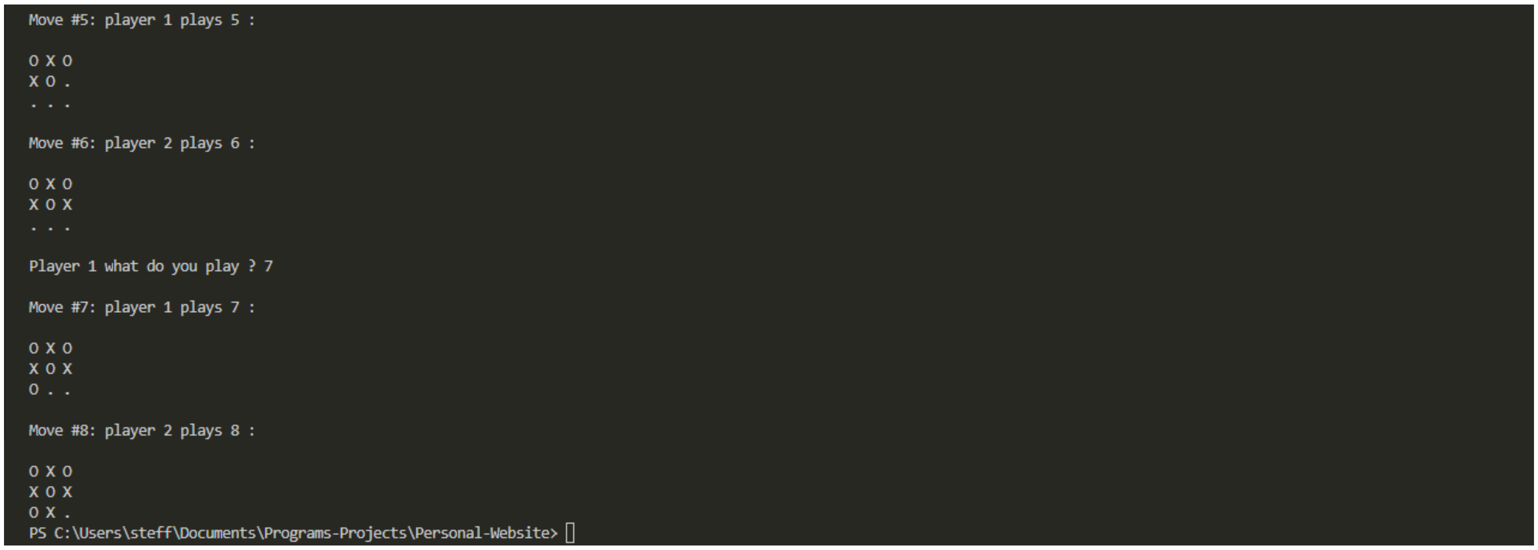
As we've proven that we all installed the appropriate libraries to run the example code, we're only using screenshots from one person's POV. All of us still ran the examples on our end just to see what it did and make comparisons. Moreover, we're also using snippets of the output for each script just to conserve space on the report.

Running connect-four.py:



```
0 1 2 3 4 5 6
-------------
X X X O X X .
O O O X O O .
X X X O X X .
O O O X O O O
X X X O X X X
O O O X O O O

Move #40: player 2 plays 6 :

0 1 2 3 4 5 6
-------------
X X X O X X .
O O O X 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

Move #41: player 1 plays 6 :

0 1 2 3 4 5 6
-------------
X X X O 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

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.

In [3]:
```

Running tic-tac-toe.py:



```
Move #5: player 1 plays 5 :

O X O
X O .
. . .

Move #6: player 2 plays 6 :

O X O
X O X
. . .

Player 1 what do you play ? 7

Move #7: player 1 plays 7 :

O X O
X O X
O . .

Move #8: player 2 plays 8 :

O X O
X O X
O X .
PS C:\Users\steff\Documents\Programs-Projects\Personal-Website> 
```

Comments:

- Examples from the Author actually make it so that the AI plays against you but it's very hard to win
- The scripts aren’t made to announce the winner at the end it seems (or at least, it didn’t show up on any of our ends)
- The functions that the game uses to run are pretty simple so that probably means we shouldn’t overcomplicate things when working on the checkers game.

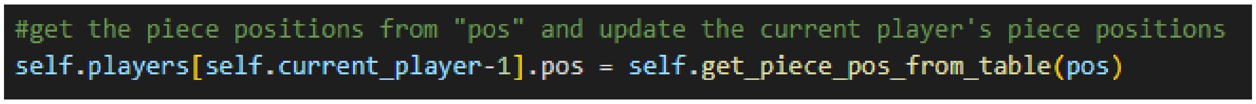
Part 3: Implementing a modified version of Checkers

- *We used the skeleton code provided here: https://github.com/Awannaphasch2016/checker_easyAI/blob/main/checker_questions.py*
- *The EasyAI Getting Started page also helped a lot: https://github.com/Zulko/easyAI/blob/master/docs/source/get_started.rst*
- *Negamax was set to 1 by default in the skeleton provided.*

Completing the Assigned Functions:

make_move(self, pos):
Architect's approach: Judging from the code from the other examples given and the EasyAI Documentation, the make_move function should only update the board. Everything else will be handled by some other EasyAI scripts. The function receives "pos" as an argument, where "pos" represents a board that reflects the chosen possible move scenario. Then it's just a matter of using "pos" to update the current player's piece positions. **It should be noted that the skeleton code allowed for a player to jump their own pieces. Since that was not prohibited by the rules, we assumed this was intentional and left it as is.**

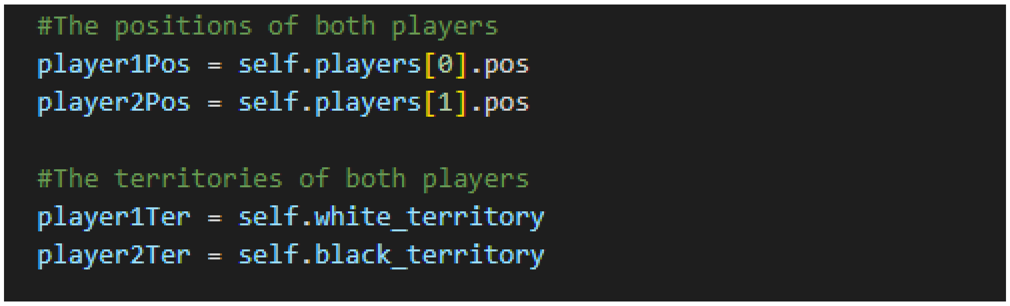
Developer's code:



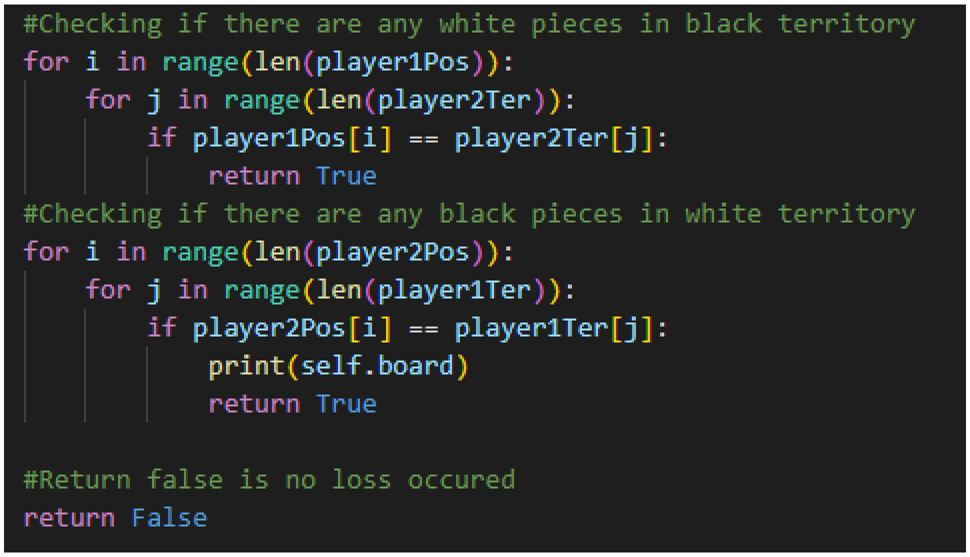
```
#get the piece positions from "pos" and update the current player's piece positions
self.players[self.current_player-1].pos = self.get_piece_pos_from_table(pos)
```

lose(self):
Architect's approach: Based on the rules given and some hints in the starter code, the only way a loss (and conversely a win) occurs is when the current player's piece ends up in their opponent's territory, so it was just be a matter of comparing the current locations of each players' pieces with their opponent's territory locations to see if there's a match; if there is that would mean that a loss has occurred. Judging from the name of the function and the context it's used in, it would return a boolean value.

Developer's code:



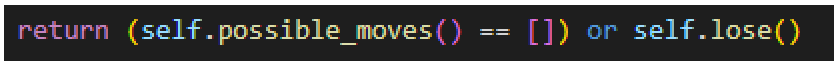
```
#The positions of both players
player1Pos = self.players[0].pos
player2Pos = self.players[1].pos

#The territories of both players
player1Ter = self.white_territory
player2Ter = self.black_territory
```



```
#Checking if there are any white pieces in black territory
for i in range(len(player1Pos)):
    for j in range(len(player2Ter)):
        if player1Pos[i] == player2Ter[j]:
            return True
#Checking if there are any black pieces in white territory
for i in range(len(player2Pos)):
    for j in range(len(player1Ter)):
        if player2Pos[i] == player1Ter[j]:
            print(self.board)
            return True

#Return false is no loss occured
return False
```

is_over(self):
Architect's approach: Based on the rules given and some hints in the starter code, the game is over when there are no longer any valid moves for the current player to make (which may result in a draw), or if the current player lost. Again, this is a matter of Judging from the name of the function and the context it's used in, it would return a boolean value.

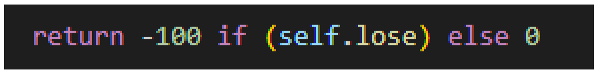
Developer's code:



```
return (self.possible_moves() == []) or self.lose()
```

scoring(self):
Architect's approach: This scoring pattern seems to be a recurring theme across the EasyAI examples. It was also given to us in the comments so there wasn't much to think about. As stated, return -100 if the current player lost; if not return 0.

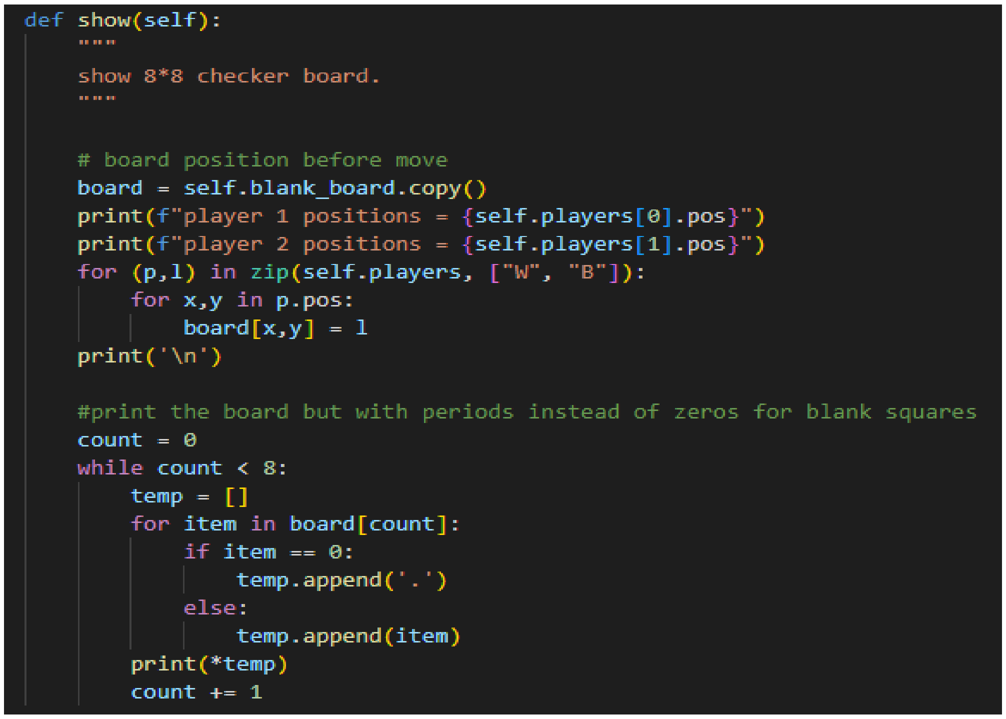
Developer's code:



```
return -100 if (self.lose) else 0
```

Other Design Decisions:

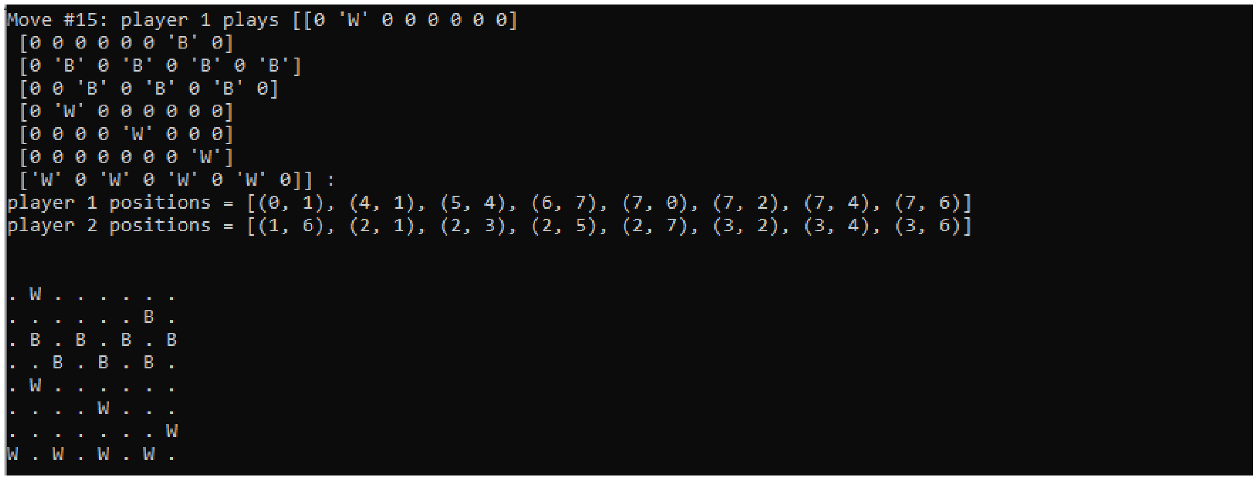
As has already been shown in the solution output, the show(self) function was modified to represent blank squares on the board with periods instead of zeros (similar to how the board was displayed in tic-tac-toe.py and connect-four.py) since the latter was quite difficult to look at and discern piece movements. The modified code is posted below.



```python
def show(self):
    """
    show 8*8 checker board.
    """

    # board position before move
    board = self.blank_board.copy()
    print(f"player 1 positions = {self.players[0].pos}")
    print(f"player 2 positions = {self.players[1].pos}")
    for (p,l) in zip(self.players, ["W", "B"]):
        for x,y in p.pos:
            board[x,y] = l
    print('\n')

    #print the board but with periods instead of zeros for blank squares
    count = 0
    while count < 8:
        temp = []
        for item in board[count]:
            if item == 0:
                temp.append('.')
            else:
                temp.append(item)
        print(*temp)
        count += 1
```

Testing/Running our Solution:

According to one of the TAs (Anna), due to the nature of the skeleton, as long as the code can run without error and works as intended after we've completed the functions, that should be the solution. Our python script **(p1_jsalsinger.py)** should be available to run to see the full output so here we'll provide a snippet of the winning move.



```
Move #15: player 1 plays [[0 'W' 0 0 0 0 0 0]
 [0 0 0 0 0 0 'B' 0]
 [0 'B' 0 'B' 0 'B' 0 'B']
 [0 0 'B' 0 'B' 0 'B' 0]
 [0 'W' 0 0 0 0 0 0]
 [0 0 0 0 'W' 0 0 0]
 [0 0 0 0 0 0 0 'W']
 ['W' 0 'W' 0 'W' 0 'W' 0]] :
player 1 positions = [(0, 1), (4, 1), (5, 4), (6, 7), (7, 0), (7, 2), (7, 4), (7, 6)]
player 2 positions = [(1, 6), (2, 1), (2, 3), (2, 5), (2, 7), (3, 2), (3, 4), (3, 6)]


. W . . . . . .
. . . . . . B .
. B . B . B . B
. . B . B . B .
. W . . . . . .
. . . . W . . .
. . . . . . . W
W . W . W . W .
```

Additional Remarks

Documented Limitations:

- This is a limitation noted in the documentation of EasyAI but the Negamax AI “may make suicidal moves if it has found that it will eventually lose against a perfect opponent”.
- The announcement of what move the player makes can’t be modified on our end, which is a shame since it’s not very visually appealing.

Future Improvements:

It’s definitely possible to extend this program by making a better GUI using some other Python library; one that allows you to create a nice board and pieces. So, instead of being run on the console it would look a lot nicer. In that case, the piece position reporting would need to be culled since one look at the board would answer that question.

Final Thoughts:

Overall, this assignment was way simpler than we expected and that’s mainly due to the helpful functions in the skeleton provided as well as the simplicity of the problems we were to solve. It was also nice learning about and working with EasyAI. Who knows, maybe we might have to utilize it again in the future.