CS 7320/5320

A4: Minimax – GoMoku

Due: Sunday Feb 23
Extra credit (5 pts) if submitted by Friday (2/21) midnight

The objective for A4 is to extend your work on A3 and get minimax alpha-beta working for GO-Moku with a 19x19 board.

Part A. Refactor Your Code for GO-Moku and Run Test

Rewrite your code to play Go-Moku on a 19x19 board where a win is defined as 5 in row along any row, column or diagonal. Run your tests (which may need modification – depends on your code).

So that we can have a competition, choose a unique name for your AI and name your files with your competition name. For example, if you choose the name "wow" as your competition name, then name your files:

- wowutil.py
- wowmmab.py (where mmab stands for minimax alpha-beta)

Since searching the entire game tree for a 19x19 board will not be possible (given the duration of the class) you will need to come up with a way to evaluate an incomplete board with some value between the max win and min win value. You may change the win values from 1 and -1 to some other value or use values between 1 and -1. Your choice.

When to Evaluate?

E**xperiment** to see how far deep down the game tree you can get in a relatively short period of time. Start with 5 seconds. If you can get to level 5 then modify your code so that when you are at level 5 with no win, you evaluate your node and stop the recursive call. Note that the more pieces on the board, the smaller the search tree.

Suggestions for evaluation:

- two in a row < three in a row < four in a row

Part B. Refactor your test code

The important functions:

- getNameForBoard(b)
- isGameOver(b)
- getWinner(b)
- genNextMovesList(b)

will need to be modified where a win is 5 in a row and the game is over if the board is full with a tie or there is a win. For an empty board the next moves list will have 19x19 = 361 boards!

Part C. Run your minimax-alpha-beta with several boards to test your code:

- **you have XXX and can move to XXXX**
- **opponent has OOO and you can block**
- **you have XXXX and can win**
- **a board with 4 pieces - pick next move**

DO:

Submit to A4 link:

a) a PDF with:
    - your Name & ID,
    - your utility and minimax code
    - output from part C

b) a ZIP file with:
- utility, minimax and testcode
- output from your tests

Grade will be based on:
- **Correctness**
- **Readability (Avoid line wrap)**
- **Understandability (good naming & comments)**
- **Code Quality.**

Notes on testing genNextMovesList()

In our earlier tic-tac-toe example, we used the following code to test if two lists of lists were equivalent, not considering order.

```
def list_of_lists_compare(list1, list2):
    return (len(list1) == len(list2)) and \
            (all(i in list1 for i in list2))

def test1():
    b = [1,2,3]
    a = [4,5,6]
    l1 = [a, b]
    l2 = [b, a]
    print (list_of_lists_compare(l1,l2))
```

Prints: True

However

While this worked for our simple 1D python lists it will not work for more complex NumPy arrays.

You will need to fix this. There are several options where the solution is to modify the data so that the above code can work.

Option 1:

Convert the NumPy array to a string (e.g. "101-1110..." or "X.O..XO.." and compare using the strings rather than the 2D NumPy array directly.

Option 2:

Apply the flatten function to the numpy array and convert to Python list

```
a  = np.array( [[1,2,3], [4,5,6], [6,7,8]] )
fa = a.flatten()   # [1 2 3 4 5 6 7 8]
list = list(fa)
print (type(fa), list, type(list))
```

Output:

```
<class 'numpy.ndarray'> [1, 2, 3, 4, 5, 6, 6, 7, 8] <class
'list'>
True
```