**CS 7320**

**Homework 3: TicTacToe Adversarial Search.   
MiniMax vs Alpha Beta Comparison.**

All files are in hw3.zip

This assignment uses NumPy arrays to represent the game board. Information on working with NumPy may be found at:

* <https://www.w3schools.com/python/numpy/numpy_intro.asp>
* Canvas Module 3 Link: Deitel.07.NumPy.zip contains Jupyter Notebooks for learning NumPy
* NumPy Documentation: https://numpy.org/doc/

**Part 1 Implement functions for minimax and test them (12 pts)**

Write code for the following functions needed to run minimax for TicTacToe

**def get\_child\_boards(board, char):**  
- return a list of all the children of a board (passed as parameter) for either ‘X’ or ‘O’ (the char parameter).

* Return a **list** of the child boards as NumPy arrays where you should always fill empty slots with 1 or -1 depending on whether the value of char parameter is 'X' or 'O'

**def evaluate(board)**:  
- return 1 for an X win, -1 for an O win or 0 for no win. A win is a column, row or diagonal with all 1s or all -1s.

**def is\_terminal\_node(board):**

'''return True if there is a winner OR all positions filled'''

pass

DO:

* Run the test program: *test1\_get\_child\_boards.*py to validate your code for *get\_child\_boards*.
* Run the test program: *test2\_evaluate.*py to validate your code for *evaluate*.
* Run the test program*: test3\_is\_terminal\_node.py* to validate your code
* Paste output into hw3 template Part 1

**Part 2. Execute RunMiniMax.**py **using your functions from Part 1. (5 pts)**

* Step1: After passing the part 1 tests, copy your three tested functions from Part 1 to ***RunMiniMax.py***
  + This code includes a Python implementation of minimax as discussed in class.
  + Run the program on the 3x3 board (b1) provided where you should as output, the minimax score of the board
* Step2: Modify the program so that it reports a COUNT of number of boards explored (i.e. tested for win or terminal node) and the time to run minimax
* Step3: The function run\_minimax is set up to always allow X the first move. This is incorrect.

def run\_minimax(board):

# set max\_depth to the number of blanks (zeros) in the board

max\_depth = np.count\_nonzero(board==0)

print(f"Running minimax w/ max depth {max\_depth} for:")

show\_board(board)

is\_x\_to\_move = True <<< only works for some boards

Replace this line with code that adds the total value of the start board. If sum==0, then   
 set: **is\_x\_to\_move = True**

If sum == +1, set value to False (means there is one more X than O on the board and it is Os turn)

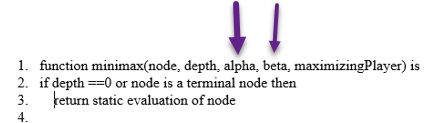
All other values means illegal board (which you can ignore, if you only send valid boards)

* Step4: Repeat for each of the other 3 boards (b2,b3, b4) you find in the multiline comment provided in the function **run\_code\_tests()**

Add code and output (score, number\_boards, time) for each of the boards to the hw3 Submission template Part 2

**Part 3. Add Alpha-Beta pruning to your the minimax code from part 2 (5 pts)**

In the class video we looked at the minimax algorithm with alpha-beta pruning. Note for alpha-beta pruning we added two additional parameters (alpha and beta) to the recursive minimax function.



To explore the performance of the alpha-beta algorithm:

* Modify the minimax function to include alpha-beta pruning. Be certain to add the *additional alpha and beta parameters* both in the function definition and for all recursive calls made to minimax. It is recommended that you do this in a separate file (e.g. RunMiniMaxAB.py) so you can go back to the original minimax.
* Note that the FIRST time you call minimax, be sure to set alpha to -math.inf and beta to math.inf
* Repeat the tests on the boards from part 2. You will need to make some modifications to the function ***run\_code\_tests ()*** to handle alpha beta
* Record the results for the different boards in the template part 3
* Prepare a bar chart that compares minimax vs. minimax w/alpha\_beta on boards b3 and b4, comparing time and number of boards explored. Add to template.

**(3 pts**)

## Submit: (two files: a pdf and a zip)

## 1. Template as PDF or doc file

2. A **ZIP file** that contains your code files

## Extra Credit: [5 pts]

## Improve the performance of the alpha-beta algorithm.

As discussed in the lecture, alpha-beta works best when the depth first search examines the best boards first. Come up with a heuristic for evaluating partial game boards (this is what chess engines do!). Based on whatever heuristic you come up with, return the child nodes in heuristic order.

Measure time and space savings for alpha-beta with heuristics, if any, and add results to the template.

While your heuristic may reduce the number of nodes explored, it may add significant time to the overall timing.

In the template, create a graph showing the comparison and provide an explanation of results.