**Northeastern University**

CS 5100 Foundations of Artificial Intelligence

**Homework and PA #2** [300 points]

***PART A: Theory and Algorithms*** *[100 points] \* See PART B Programming Assignment on Page 3*

Please - clearly write your **full name** on the first page. Submit a single PDF file as your answer for Part A.

Please provide brief but complete explanations, using diagrams where necessary, and suitably using your own words. While presenting calculations and equations, explain the variables and answers in words.

Study Chapter 4 of Russel AI textbook 3rd Edition. Answer the below:

1. Consider the traveling salesman problem (TSP) example, with only a few nodes (4-5).

Then, explain how the following algorithms work and how they can be applied to solve the TSP:

1. Hillclimbing search
2. Simulated annealing

Clearly show the basic math and the steps involved, and how to apply them to TSP. [10 points]

1. Solve 4.2 [10 points]
2. Solve 4.10 [10 points]
3. Solve 4.12 a only [10 points]

Study Chapter 5 of Russel AI textbook. Answer the below:

1. **MiniMax Algorithm** [10 points]

In the context of a simple board game of your choice, answer the following questions:

1. How were the game states represented as a game tree? Explain with pictures.

**References** for MiniMax

https://www.baeldung.com/java-minimax-algorithm

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

http://www.doc.ic.ac.uk/~sgc/teaching/pre2012/v231/lecture5.html

1. **Chapter 5 (**Russel AI textbook): Answer the below questions
2. Solve 5.8 a and b ONLY [10 points]
3. Solve 5.12 [15 points]
4. Study how the Connect Four game works. In connect-four, players alternately drop discs into the columns of a vertically positioned grid; the first player to get four of her discs in a row wins (if the board fills up before this happens, it’s a draw). [25 points]

For this game, explain the below:

1. How is a Game Tree constructed (a small sample of states).
2. Formal definition of this game as an Adversarial Search problem.
3. Explain with a diagram and pseudocode how Mini-Max can be used.
4. How does Alpha Beta pruning work for the game, what does it improve?
5. How would you apply a Cutting Off Search strategy? What would be the heuristic?

***PART B: Implementation – Programming Assignment (PA #2)*** *[200 points]*

1. Implement genetic algorithm in Python for creating a list of N numbers that equal X when s squared and summed together. Please refer to the Genetic Algorithm basics Tutorial:

<https://lethain.com/genetic-algorithms-cool-name-damn-simple/>

<http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3>

1. In this part, you’ll program a minimax module for the board game Othello, also known as Reversi. The main module that you’ll program will calculate the value for a given board position. But, code is also provided for you that will let you play against your AI if you like.

*Provided files: othello.py (stub), clearBestMove.txt, clearBestCounterMove.txt, arbitraryBoard5.txt, arbitraryBoard8.txt, board1.txt, endgame.txt*

The rules of Othello are as follows:

1. The two player colors are white and black. The white player goes first.
2. You capture an opponent’s pieces when they lie in a straight line between a piece you already had on the board and a piece you just played. (A straight line is left-right, up-down, or a 45 degree diagonal.)
3. You can only play a piece that would capture at least one piece. If you have no legal moves, the turn is passed.
4. The game is over when neither player has any legal moves left. Whoever controls the most pieces on the board at that point wins.

Something that is slightly unusual about Othello for minimax is the fact that **a turn might be skipped** if a player has no legal plays. You’ll have to take that into account in your minimax calculations. (Don’t have skipped turns count against the search depth.)

The AI is always presumed to be white for this assignment; if you try the demo mode, you as the human will be playing black.

The input will be the search depth on a single line, followed by an ASCII representation of the board (W for white, B for black, - for an empty space).

1. Download the provided othello.py code.
2. Implement basic depth-limited minimax for the minimax\_value function, ignoring the alpha and beta arguments for now. Your evaluation function, when you bottom out, should just be the difference in piece count between white and black if it's not the end of the game, or WIN\_VAL, -WIN\_VAL, or 0 if it is the end of the game. You should be able to effectively use a depth of 5 or so without waiting too long.

*Tip: While debugging, you’ll find it useful for minimax to print the board state it is evaluating and the value that it assigns to that board. Also, debug small depths before attempting larger ones.*

1. Try running on the following boards that have a search depth of 5 or less: clearBestMove.txt: Value should be 2.

clearBestCounterMove.txt: Value should be -3. arbitraryBoard5.txt: Value should be 1.

You can feed a text file to a program with input redirection:

python3 othello.py < clearBestMove.txt

1. Implement alpha-beta pruning.
2. Check that you still get the same values on the files from part C.

**Your code should now run in a reasonable amount of time on arbitraryBoard8.txt, board1.txt, and endgame.txt. Submit the value of each board in your PDF.** Note that endgame.txt will check that your code behaves reasonably when one player has no moves, and at least one of these tests will check that you're returning WIN\_VAL or -WIN\_VAL instead of a piece count difference when appropriate.

Optional: You can play against your AI if you just run python3 othello.py and type play at the prompt. Set DEMO\_SEARCH\_DEPTH to whatever degree of lookahead you have the patience for.

**Submission Checklist:**

genetic algorithm

othello.py (with minimax\_value fully coded) Answers PDF ( board values)