CMPT310: Spring 2025:

Assignment 2: Adversarial search

TicTacToa game

In this assignment various algorithms for adversarial search are to be implemented for generalized n x n TicTacToa game:

1. MinMax
2. MinMax with AlphaBeta Pruning
3. MonteCarlo Tree search (MCTS)
4. Currently game is playable with AI playing random.

There is a timer setting for the game that is set by default to -1. This means no time limit. Once you set the time limit to > 0, that time in second is used to limit the depth search. This means a cutoff version of MinMax and AlphaBeta needs to be implemented as part of this assignment. Depth-cutoff version of the above algorithm is to be implemented as a separate function. There are drivers for minmax and alphabeta in games.py to select the right execution path depending on the value of the timer.

1. There are 2 scripts for you edit: games.py and Montecarlo.py. The area you must provide your implementation is specified by a comment saying “*your code goes here*“ plus the points given for that part.

The depth-cutoff versions of MinMax and AlphaBeta which use Evaluation function at the set depth to evaluate the node’s value.

1. MonteCarlo Tree Search (MCTS):

MonteCarlo Tree Search algorithm’s shell is given in monteCarlo.py. There are various parts in there for you to complete.

1. You need to come up with a smart Evaluation function for the game. It must give highest value to the board configuration which has the most potential for a win by the AI player. Find out what characteristics in the current layout are more promising and then try to quantify them in the given shell function eval1() in game.py. There are different ways to achieve this, use your game skill, creativity, and time. There are already a few suggestions given there.

Once done, you should be able to play 3 x 3 TicTacToa for any of the algorithms with no time limit (meaning timer set to -1). For 4x4 and 5x5 you will need to set the timer to a positive value, so it can be playable. Better print out the depth in the iterative deepening so you can adjust the timer. Usually, timer=5 is the limit of playability. Higher value can cause long delays for AI to play. There are 2 run configurations to run 4 x 4 and 5 x 5 version when you load the project in PyCharm.

With proper implementation, AI should be able to always tie the game and never looses. This is what we mean by saying this game is solved!

A couple of questions for you to think about:

1- You will notice that often the time limit set at GUI is surpassed in runtime, meaning AI takes often longer and some time a lot longer to finish its search. Why do you think it is?

2- Any suggestion on how to remedy that so we can have a time limit that is respected by the runtime?

Rubric:

This assignment rubric is based on correct implementation of each algorithm. The resulting performance numbers and quality of the agent would follow if the implementation is correct. In general, based on your computer specs, all algorithms should be playable with time limit of 4 or at most 5 seconds. Sometimes choosing 5 seconds for timer may result in longer time for AI to play due to the last depth may take longer to return. Grades are split among various parts of the assignment and specified in the related areas in the 2 py scripts.

For upload you can zip the 2 files (game.py, monteCarlo.py)and upload it. The TAs will insert them in their game shell and run it in various setting to check performance is in the time range specified above, then check your code for each section to make sure they are implemented as requested. It is **very important** not to modify the 2 scripts where it is not specified.