University of Pittsburgh
CS 2710 Foundations of Artificial Intelligence — Handout 5
Professor Milos Hauskrecht — September 25, 2019

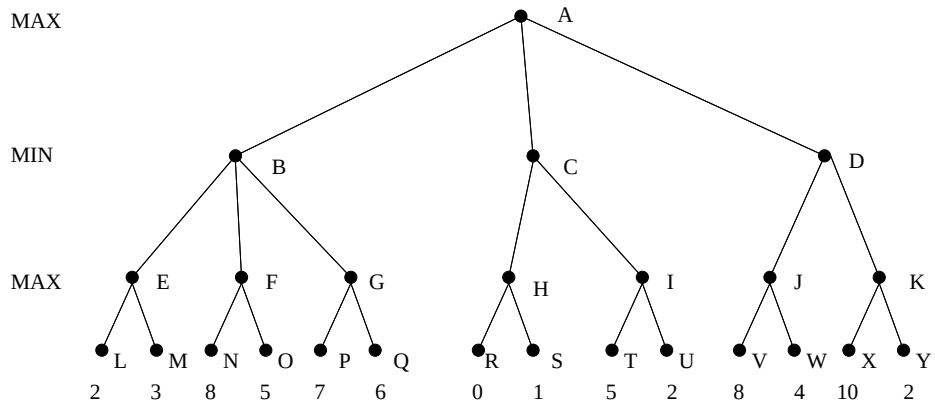
---

**Problem assignment 4**

*Due: Wednesday, October 2, 2019*

## Problem 1. Adversarial search.

Consider the game search tree in the figure below




Assume the first player is the max player and the values at leaves of the tree reflect his/her utility. The opponent wants the same utility to be minimized.

Part a. Compute the minimax values for each node in the tree? What move should the first player choose? What is the solution path the rational players would play.

Part b. Assume we use alpha-beta algorithm to explore the game tree and we do this in the left-to-right order and determine the players strategies. List all nodes that are cut off from the tree and are never examined by the alpha beta procedure (see lecture notes for an example of how the alpha beta procedure works).

Part c. Assume we use alpha-beta algorithm but explore the tree in the right-to-left order. What nodes would not need to be examined by the alpha-beta algorithm and pruned away?

## Problem 2. Tic-tac-toe-10 player

The goal of this assignment is to implement a program capable of playing Tic-tac-toe on the $10 \times 10$ board. The $3 \times 3$ version of the game is discussed in the textbook. Both $3 \times 3$ and $10 \times 10$ versions of the Tic-tac-toe are related to the old Japanese game of Go.

**Rules.** The game is played on a rectangular $10 \times 10$ board. There are two players, one plays crosses, the other plays zeroes. The players alternate their moves. The first one to achieve five of his marks in a row, a column or on a diagonal wins.

**Programs.** The programs you were given include: *tictactoe.py*, *player.py*, *heuristics.py*, *naive_heuristics.py* and *main.py* files. The *tictactoe.py* file defines the board and the game that is played by two specific players.

The *player.py* class defines a player and is performing $k$-ply game search up to level $k$ for the current board position to make its move.

The *heuristics.py* defines the basic (but decent) board evaluation heuristics. Check comments in the file on how to define self.patterns variable that drive the evaluation. Briefly, each pattern in the patterns is associated with a weight that defines how much (if it is present in a row, a column or on a diagonal) it contributes to the configuration score. The class *naive_heuristics.py* defines a subclass of that class and is equipped with a simpler heuristics. Please note a player may be customized with its own heuristic and you will be asked to write one yourself.

Finally, the *main.py* file initializes the two players, their heuristics and the game the two players play. It runs two games where each player starts once. The settings given for the players are $k = 2$ (2-ply search). Also note you can see or hide each move by setting *print_step* parameter of the *Player* class. The main file you received currently runs one player (player A) equipped with a naive heuristics, and one (Player B) with the basic heuristics.

**Part a.** A tic-tac-toe player: naive vs basic board evaluation heuristics.

Our objective is to create a decent player of the game. Ideally a player would search the complete game tree and compute the best move while considering the best (rational) responses of its opponent. However, in this and many other games we do not have the luxury of exploring the full game tree before making the move. Hence we consider a limited-depth exploration of the game tree, where nodes at the depth (or cutoff) limit are evaluated using a board evaluation heuristic. If the depth of the tree is k we refer to the procedure as to k-ply search. The default level is k=2. Please note the both $k$ and heuristics influence the quality of the player.

Modify *main.py* (you do not need to submit the modified file) so that it plays 10 matches of Player A (basic heuristics) and Player B (naive heuristics), so that both Players start 5

times. Report the table with results you have achieved in terms of wins, draws and losses of Player A vs Player B. Summarize and analyze the results. Explain why the specific set of results were obtained.

**Part b.** A tic-tac-toe player: basic vs basic board heuristics.

Now change the Player B who played the naive heuristics in Part a and let it use the same basic heuristics as used by Player A. Play Player A vs B 10 times by always starting Player A. Summarize and analyze the results. Explain the observed results.

**Part c.** A tic-tac-toe player: basic vs basic board heuristics using different k-ply levels.

Change the Player B from Part b that plays the basic heuristics to use $k = 1$ ply search Player A should still use the basic heuristic and $k = 2$ ply search. Play Player A vs B 10 times so that both Players start 5 times. Summarize and analyze the results. Explain the observed results.

Optional. Try to play (one against the other) two players with the basic heuristics, such that one uses $k = 3$ and the other one $k = 2$ ply search. Analyze the results and compare to $k = 2$ and $k = 1$ players.

**Part d.** A tic-tac-toe player: my heuristics vs basic heuristics.

One way we can improve a tic-tac-toe player is to improve its evaluation heuristic. The basic heuristic function (file heuristics.py) analyzes the game configurations and identifies different line configurations of X and O symbols (in rows, columns and on diagonals). The line configurations are assigned a score which is then used to calculate the utility of the position.

Write a subclass of Heuristics class (defined in *heuristics.py*) called MyHeuristics and include it in file *my_heuristics.py*. Hint: Check how we defined the NaiveHeuristics class as a subclass the Heuristics class.

Define Player A that uses MyHeuristics and Player B that uses the basic heuristics. Set $k = 2$ for both. Experiment with the different versions of MyHeuristics by trying: (1) different scoring of line configurations (2) addition of new line configurations. Once you think your heuristic performs well play 10 games of Player A vs B such that each player starts 5 times and report the results. Clearly describe in the report the changes in heuristics you have made including the intuition behind these changes. Please submit your final version of *my_heuristics.py* file.

**Play for credit.** The heuristics submitted by all students in the respective $my_heuristics.py$ files will be run against each other in a tournament. Pairs of programs will play each 10 times. The competition will consist of a group stage and knockout rounds. The winners (first three submissions) will receive extra-credit for the assignment and will be announced in the class.