Artificial Intelligence: Second assignment

Submission: 23/03/2018

March 6th 2018

This assignment will be evaluated and used to calculate your final socore. The material related to this assignment can be found in chapter 5 of the main textbook (Artificial Intelligence, a Modern Approach, Russell and Norvig, 3ed), and in chapter 2 of Artificial Intelligence, a new synthesis (Nilsson). Both books are available in the library. Besides these two books, there is quite a lot of material available about adversarial games in the internet (beware to choose wisely).

In this assignment, you will learn how to design and implement a relatively simple program that is capable of playing connect-four with a human.

Connect Four: The Game (**part of this material was taken from a similar assignment from an AI course in Harvard and available at:** `http://isites.harvard.edu/fs/docs/icb.topic623248.files/Asst3/asst3c.pdf`)

Connect Four is a two-player strategy game similar to tic-tac-toe. It is played using 42 tokens (usually 21 red tokens for one player and 21 black tokens for the other player), and a vertical grid that is 7 columns wide. Each column is able to hold a maximum of 6 tokens. The two players take turns. A move consists of a player dropping one of his/her tokens into the column of his/her choice. When a token is dropped into a column, it falls until it hits the bottom of the column or the top token in that column. A player wins by creating an arrangement in which at least four of his/her tokens are aligned in a row, column, or diagonal. For example, consider the state of the game shown in Figure 1.

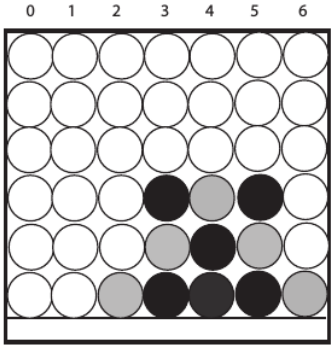

Figure 1: Sample Connect Four Board

The two kinds of tokens here are Black and Gray; it is Gray's turn to move. If Gray drops token into column 3 or column 5, he/she wins. It is quite possible for a game of Connect Four to end in a draw, i.e. in a state where all 42 tokens have been used, the grid is full, but there are not four tokens of either color aligned in any direction at any location.

The interface for the game can be something like the one shown in Figure 2.

```
-------
-------
-------
-------
X--O-XO
X-OXOXO

It is now X's turn.
Make a move by choosing your coordinates to play.
```

Figure 2: Example of interface for Connect Four

After X makes a move, the computer takes this new board with X move added, and uses one of the two algorithms: minimax or alpha-beta to make its own move. When the computer finishes choosing the best move among the possible ones, it will then exhibit a new board with the computer's move (in this case a new 'O' will show in the position chosen by your program), and wait for the human to play.

Besides implementing the minimax and alpha-beta algorithms, your program needs also to implement the possible movements for a given player. For example, given the configuration of Figure 2, player X can drop in any of the 7 columns, and your program needs to decide which is best.

Naive minimax always works in principle, but it fails on large game trees in practice. To address this problem, perform the following:

- Modify your move generator to keep track of the search depth, and have it stop generating new successor states after a given depth limit has been reached. Then, have the move generator call a function to estimate the value of the current state, and return this value.

The speed with which your program plays will depend largely on how efficient the evaluator is. For some games, writing an evaluator is trivial; e. g. for tic-tac-toe, checking whether the middle square on the 3x3 board is occupied by X, O, or neither, and assigning the state a value of +0.5, -0.5, and 0, respectively, is perfectly sufficient. For other games such as chess or Go, however, good evaluators are few and far between. It turns out that the following simple evaluation function for Connect Four (based on R. L. Rivest, Game Tree Searching by Min/Max Approximation, AI 34 [1988], pp. 77-96) performs surprisingly well:

a win by X has a value of +512,

a win by O has a value of -512,

a draw has a value of 0,

otherwise, take all possible straight segments on the grid (defined as a set of four slots in a line horizontal, vertical, or diagonal), evaluate each of them according to the rules below, and return

the sum of the values over all segments, plus a move bonus depending on whose turn is to play (+16 for X, -16 for O).

The rules for evaluating segments are as follows:

```
-50 for three Os, no Xs,
-10 for two Os, no Xs,
- 1 for one O, no Xs,
  0 for no tokens, or mixed Xs and Os,
  1 for one X, no Os,
 10 for two Xs, no Os,
 50 for three Xs, no Os.
```

While this evaluation function is guaranteed to work, it is not necessarily the best or the easiest to implement efficiently. Notably, it doesn't favor quicker wins. That is, if X can win in both 23 and 17 moves, there is no incentive for him to win faster.

What to submit:

1. written report containing the analysis of the two algorithms: minimax and alpha-beta pruning.

   Organization of the written report:

   Introduction
   : Description of adversarial games and algorithms to solve them.

   Minimax Algorithm
   : Description

   Alpha-Beta Pruning
   : Description

   Connect Four
   : Game description, rules and evaluation functions.

   Minimax and Alpha-beta applied to Connect Four
   : Discussion about the two algorithms applied to the game. Description of the implementation. Data structures used, evaluation function used. Results and number of nodes pruned by the search compared with minimax. You should also mention if you imposed a depth-limit and which is this limit.

     Show performance curves (execution times and number of nodes expanded) for both algorithms per each computer move (X axis is the computer move and the Y axis is the execution time or number of nodes expanded).

   Final Comments and Conclusions
   : Comment about the performance of both algorithms.

   Bibliography
   : Don't forget to cite the bibliographic references in the text. If you use figures or text from other sources you should cite that source, and list it in this section.

2. source code, how to compile and run, the input format and environment (OS, machine and compiler versions). Your program should run in my machine that runs fedora. Do not assume any IDE (Integrated Development Environment). Your program should run from the command line.

   Submission will be done through Moodle.

Submit, via Moodle, a zip or similar archive containing the source code of your program and instructions on how to compile and execute (i.e., a brief manual on how to compile and run your program). If you needed special libraries or a specific compiler version, please indicate that in your manual. Please, be careful about the use of graphical accents in the program text (Java and python, in particular, may not run in my machine because of the coding you use: utf8, iso* etc).

This assignment is to be performed by a team of maximum of three people.

All assignments will be presented in a date yet to be defined. All team components must be present.

If the work is performed by a team I will assume that each team component knows what everyone is doing. Please, ensure that every component knows the contents (theoretical and practical) of the assignment to avoid undesired penalizations.