Artificial Intelligence

Adversarial Search Coursework
Assigned: 26 March
Due: Thursday, 2 April, 19h30m

Prof. Felipe Meneguzzi
João Paulo Aires (assistant)
Maurício Magnaguagno (assistant)

March 20, 2015

1 TicTacToe

You must work on this project **individually**. You are free to discuss high-level design issues with the people in your class, but every aspect of your actual implementation must be entirely your own work. Furthermore, there can be no textual similarities in the reports generated by each student. Plagiarism, no matter the degree, will result in forfeiture of the entire grade of this assignment.

TicTacToe[1], is an ancient game, that, if Wikipedia is to be believed, originates in Ancient Egypt, and was played in Ancient Rome. More importantly, it is simple enough that one can easily generate the entire game tree within a computer, which lends itself to implementation without complex optimizations to the basic `MiniMax` algorithm. An example of a set of legal moves is shown below in Figure 1.

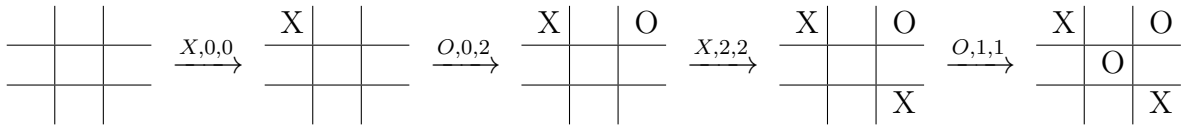

Figure 1: A sequence of valid moves for a tic-tac-toe problem

In this assignment, you will calculate the next move for the computer player using the MiniMax algorithm. Since the command line allows you to select which player moves first (i.e. the human or computer), your code must be able to handle the scenario where the human moves first and the scenario where the human moves second. You can also specify that two computer players (i.e. either Random, MiniMax or AlphaBeta) will play against each other, so your code must be generic enough to handle being the X player or the O player.

The game of tic-tac-toe is small enough that we can generate the entire game tree while doing the MiniMax search. As a result, you do not need to create an evaluation

[1] A.k.a. *Jogo da Velha*

function for non-terminal nodes (i.e. the `H-MiniMax` algorithm). What you will need to create is a utility function which returns the utility at terminal states.

You will also need to create a successor function. This function takes the current state of the game and generates all the successors that can be reached within one move of the current state. Both the utility function and the successor function need to be written. You get to decide what these functions should look like. Once these two functions are in place, you can begin coding the actual MiniMax algorithm.

2 Overview

For this assignment, you will be implementing the `MiniMax` adversarial search algorithm to compute the optimal move for either one of the players in the game. In this assignment you can also implement the more advanced Alpha-Beta search algorithm to speed up the search by cutting unnecessary branches of the game tree. For those who have never seen the game, the final state of the game occurs when either player has managed to align three of his/her symbols vertically, horizontally or in diagonal, or when there is no more blank squares on the board, as shown below.

X	X	X
	O	
O		

(a) X wins horizontally

X		O
X	O	
O		X

(b) O wins diagonally

X	O	X
X	O	O
O	X	X

(c) Draw

Figure 2: Endgame examples

3 Implementation and Deliverables

At the bare minimum, you are required to implement only the API in Listing 1, below. However, it is highly recommended that you create additional methods to represent elements of the algorithm, such as the utility function and the successor function that generates the moves available to each player. We note that the API shown below is deliberately simplified to give students the maximum freedom to develop their own internal APIs (and minimize coincidental similarities in implementation), and that well developed code is a component of the final mark. By way of comparison, the instructor's implementation has 3 additional methods, and uses the method of Listing 1 to invoke the minimax function. This interface **must be implemented** for every player you may add.

At the end of this assignment, you will upload **one zip file** named `s<ID>.zip`[2] containing:

- your implementation of `player_minimax.py` and any other classes you use to implement your solution (evidently excluding those already present in the standard Python distribution);

[2]Where `ID` is your student id number, e.g. if your ID is 13303580, then you must create `s13303580.zip`

Listing 1: `player_minimax.py`

```
from common import *

class Player_Minimax:

    def __init__(self, index):
        # Index is the player representation in the board
        # Clear = 0
        # O = 1
        # X = 2
        self.index = index

    def get_next_move(self, board):
        # TODO return movement as (x, y) or None
        return None
```

Do not modify the other python scripts in your project, your zip file should unzip into a directory **exactly** like the one you received.
You are **not allowed** to use any external libraries besides the one supplied with the assignment package.

- one `readme.txt` file answering the questions in the appendix;
- any unit tests you developed to test your helper classes.

Your deliverables must be handed in via Moodle using the appropriate upload room: Adversarial Search Assignment.

Note that the code you develop must be able to execute and run the tests using Python 2.7 scripts from the assignment package.

4 Grading

In order to properly evaluate your work and thought process (and to help improve future iterations of this course), you will complete a `readme.txt` answering the following questions.

1. Explain briefly how you implemented the helper methods for `get_next_move()`.
2. Explain briefly how you represented the utility of the end game. Did you use different values for different end configurations?
3. Explain briefly how you generated the successor moves.
4. How often does your player wins when it plays as X against a randomizing player, and how often does it wins when it plays as O?

5. If you wanted to solve some other games using variations of minimax (such as the ones seen in class), which ones would you use, and how would you integrate them into this code? Why?

6. If you did the extra credit, describe your algorithm briefly and state the order of growth of the running time (in the worst case) for `get_next_move()`.

7. Known bugs / limitations.

8. Describe whatever help (if any) that you received. Don't include readings, lectures, and precepts, but do include any help from people (including staff, classmates, and friends) and attribute them by name.

9. Describe any serious problems you encountered.

10. List any other comments here. Feel free to provide any feedback on how much you learned from doing the assignment, and whether you enjoyed doing it.

Grading will take into consideration elements of your code, testing and reporting of the work done. The criteria, as well as their weight in the final grade is as follows:

- Basic Correctness (30%) — correctness of the algorithm as determined by the unit tests provided with the assignment, this means your implementation must win as the X player more than 5 times out of 10;[3]
- Full Correctness (40%) — correctness of the algorithm for a more comprehensive suite of tests, which includes winning as the X player more than 9 times out of ten and **timeout of a few milliseconds** to play one hundred games against the randomizing player;
- Coding style (10%) — code organisation, object orientation, commenting and readability;
- Overall report readability (20%) — this relates to the `readme.txt` file you must fill out, we will assess how accessible and coherent the explanation of your code and solutions is; and
- Efficient alpha-beta search (bonus 20%) — writing an efficient algorithm to search for moves using alpha-beta pruning (i.e. an efficient implementation of `player_alphabeta.py`).

5 Code-Specific Advice

- In file `common.py`, you can check many conditions of the endgame, including whether a particular player has won;
- We already helped you figure out part of the problem of how to make your `Player_Minimax` be generic by providing the `index` which you can use to check whether the current player has won in the current `board`;

[3]Do note that an implementation that does not actually implement `MiniMax` will result in forfeiture of the entire score.

- All classes and methods which you must complete yourself are marked with the `TODO` keyword (search for that using your favorite editor).

6 Miscellaneous Advice

Here are some lessons we learned in creating our own solution and writing papers/reports:

- It took the instructor approximately 30 minutes to code and debug the solution for this problem from scratch based on the pseudocode at AIMA, plan your time accordingly;
- The Python standard library has a number of classes that greatly facilitate developing the data structures required for an efficient implementation;
- The instructor's implementation takes about 5 **milliseconds** to play a single game using the non-optimized minimax algorithm and less than one millisecond to do so using alpha-beta pruning;
- You can automatically test your own code with the `test_tictactoe.py` script that comes with the project package, simply open a terminal, go to your project folder and run `python test_tictactoe.py` in the command line. This assumes you have python installed in your system and the current directory is the assignment folder.