

Assignment 3 Due date: November 18

12 marks

Search & Planning in AI (CMPUT 366)

Submission Instructions

Submit your code on eClass as a zip file and the answers to the questions of the assignment as a pdf. The pdf must be submitted as a separate file so we can more easily visualize it on eClass for marking.

Overview

In this assignment, you will implement the Minimax and Alpha Beta search algorithms to solve puzzles in the board game of Connect-4. That is, we will provide a state of the game and you will use Minimax search with and without Alpha Beta pruning to find the optimal move in that state. Connect-4 is played on a board of size 7×6 (7 rows and 6 columns) where two players (which we'll denote 'X' and 'O' for Max and Min players, respectively) take turns placing a single piece in a column on the board. The game ends when one player connects 4 of their own pieces either horizontally, vertically, or diagonally. For example, the board below is a winning position for player 'X' since they have connected 4 pieces in a row along a diagonal:

			О			
		О	X	X	О	
	О	X	X		О	
X	X	О	X	X	О	

Most of the code you need is already implemented in the assignment package. In the next section, we detail the functions you will use from the starter code. You can reimplement all these functions if you prefer, their use isn't mandatory. The assignment must be implemented in Python, however.

Starter Code (0 Marks)

The starter code comes with a class implementing a Connect-4 board. The following section will detail the functionalities that are provided within the starter code.

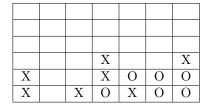
Board Implementation

The Board class (see connect4.py) implements a Connect-4 board containing the following key functions:

- The board is represented as a 2 dimensional array. An instance of Board named b can be created with the instruction b = Board(). It is possible to complete the assignment without creating instances of Board as the test cases already create them for you; see the files testminimax.py and testalphabeta.py for examples. Feel free to create instances of Board if they are needed in your implementation.
- If you want to print instance b then you can simply write print(b).
- The method available_moves returns a list of all moves available at a given board. The allowable moves for a player are the columns in which the player can make a move in, starting indexing from 0. For example, in the above board, the player can move in any of the columns 0 through 6, so available_moves would return a list of all the numbers from 0 through 6, inclusive.
- The methods perform_move and undo_move places a piece in a column and removes a piece from a column, respectively. The ability of removing a piece from the board allows you to implement a recursive depth-first search by following these steps, where you can use a single instance of Board:
 - 1. for each m in available_moves:
 - (a) board.perform_move(m)
 - (b) Recursive call on board
 - (c) board.undo_move(m)
- Method is_terminal returns True if the board represents a terminal state of the game.
- Method game_value assumes that the board represents a terminal state of the game and returns +1 if it is a win for 'X', -1 if it is a win for 'O', and 0 if it is a draw. This method returns None if invoked for a board that doesn't represent a terminal state.

Connect-4 Puzzles

The files testminimax.py and testalphabeta.py contain 21 Connect-4 puzzles similar to the one shown in the grid below. For each puzzle we will provide the number of moves the game will finish if the players play optimally. You will run Minimax Search (in file testminimax.py) and Minimax Search with Alpha Beta pruning (in file testalphabeta.py) to find the optimal move for each puzzle. In the puzzle below, where 'O' is to move next, the player can guarantee a win by playing on the 6-th column from left to right (can you tell why?). The search algorithms should be able to return this move as the optimal move for 'O'.



Implement Minimax Search (4 Marks)

Implement the Minimax search in the function with the same name in testminimax.py. The function receives an instance of the Board class, the player who is to act at this state (either 'X' or 'O'), and the maximum search depth given by variable ply. The function must return three values, in the following order:

- 1. the score of the optimal move for the player who is to act;
- 2. the optimal move;
- 3. the total number of nodes expanded to find the optimal move.

The implementation must be correct, i.e., it must find an optimal solution for the search problems. The algorithm must perform the Minimax search up to the given depth (specified by the input variable ply). If Minimax encounters a state at depth ply that isn't a terminal state (i.e., the value of the game cannot be computed), then the function should return the values of: return 0, 0, 0. The search must not go deeper than the depth specified in ply. If there is more than one optimal move, the algorithm should return the one with lowest index (columns to the left); failing to do so might make your code not pass the test cases.

You can test the correctness of your implementation by running python3 testminimax.py. If your implementation is correct, it should return an output of the following form:

Ran 21 tests in (running time of your implementation)s

If there are any mistakes and mismatches they will be shown here.

Implement Alpha Beta Algorithm (4 Marks)

Same instructions as above, but you will implement Minimax with Alpha Beta pruning in testalphabeta.py.

Analyzing the Algorithms (4 Marks)

Once you have implemented both the Minimax and Alpha Beta algorithms and they passed all 21 test cases, print the number of expansions each algorithm performs to solve the last test case in each file. The last instance is the one instantiate with the following instruction: player = b.create_board('336604464463'). This test case is given by the grid below, where 'X' is to act. You will also see from the return of your algorithms that this is a winning position for 'X' as the value of the state is +1 and the optimal move is to play in 4-th column from left to right.

			О	X
		О	X	О
		О	X	О
X		X	О	X

Considering the Connect-4 puzzle shown above, answer the following questions.

- 1. (1 Mark) How many states did Minimax and Alpha Beta expand for the starting position provided above?
- 2. (1.5 Mark) How can player 'O' win the game from a state such as the one shown above?
- 3. (1.5 Mark) Is it possible for Minimax and Alpha Beta to expand the same number of nodes? Explain why not, or under which conditions this is possible.