CS 682 – Artificial Intelligence

Project 1 – 3×3 Tic-Tac-Toe

Monikrishna Roy

September 24, 2021

Task Description

Write a program to play 3×3 tic–tac–toe. The program should use minimax search. You do not need to play the entire game, you just need to make one move. For example, the program should be executed like:

```
./tictactoe <infile> <outfile>
infile: file to be loaded with a board state
output: file to be saved with board state after a move is made
```

Your program should make a single move (x,y) which will be written to a file $< outfile >$, and exit. If there is an immediate win in the moves available, make that move. Otherwise, do minimax on all available moves.

Purpose

The purpose of this assignment is to provide a basic programming experience with an algorithm commonly-used in AI. You should be able to demonstrate the ability to utilize a recursive algorithm in a game-playing scenario.

Algorithms

This program uses the minimax algorithm with alpha-beta pruning to find the best available moves.

Algorithm Details

A minimax algorithm is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is A's turn to move, A gives a value to each of their legal moves.

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Pseudocode

Pseudocode for the minimax algorithm with alpha-beta pruning.

```
function minimax_optimization(node, alpha, beta, maximizingPlayer) is
    if node is a terminal node or don’t have legal moves then
        return the heuristic value of node
    if maximizingPlayer then
        value := -INF
        for each child of node do
            value := max(value, minimax_optimization(child, alpha, beta, FALSE))
            alpha := max(alpha, value)
            if value >= beta then
                break
        return value
    else
        value := +INF
        for each child of node do
            value := min(value, minimax_optimization(child, alpha, beta, TRUE))
            beta := min(beta, value)
            if value <= alpha then
                break
        return value
```

```
(* Initial call *)
minimax_optimization(origin, -INF, +INF, TRUE)
```

Pre-requisities for run locally

Some pre-requisities to run the program.

- cmake >= 2.6
- make
- gcc/g++ >= 5.4

Compilation Instructions

Instructions to compile the project. First, need to create a build directory, then need to run make to build the program.

```
<this section should only need to be done once>
$ mkdir build && cd build
$ cmake ..

<each time you want to compile (from the build directory)>
$ make
(if successful, the executable will be in the ’build’ directory)
```

Test/Play Instructions

Details description of how to play the game. Options are not mandatory to play the game. However, if an input file is provided, then an output file must need to be provided.

```
Usage
    ./tictactoe [inputFile] [outputFile] [--play]

DESCRIPTIONS
    A program to play the 3×3 tic{tac{toe game interactively.

    The program used a minimax search algorithm with alpha-beta
    pruning to get the best moves if no immediate win move is available.

    If the initial state is in losing position,
    it will return the first available move.


OPTIONS
    inputFile
        Name of the input file. It contains the initial board state.

        The board value should be -1, 0, or 1,
        where -1, 0, and 1 means O, empty, and X, respectively.

        If you give the the inputFile, you must give the outFile as well.

    outputFile
        Name of the output file. Programs will save the final state of the board.
        If file is not exit, it will create one.

    --play
        If you want to play interactively with AI.
        Otherwise, the program will make only one move.
```

Run Commands

The commands are to run the program or conduct the entire game.

```
<need to run from buld directory, skip if already in there>
$ cd build

<command to make one possible best move>
$ ./tictactoe test.board out.board

<command to play interactively with AI>
$ ./tictactoe test.board out.board --play

<command to play from the ZERO state>
$ ./tictactoe
```

Results and Discussion

Here is the output for the provided test board without any immediate winning move. The program made the move (0, 0) which is one of the best moves.

```
$ ./tictactoe test.board out.board

<initial state>
  |   |
----------
X | O |
----------
  |   |

Best move by AI (User X): 0 0

<state after best move>
X |   |
----------
X | O |
----------
  |   |
```

Here is the example for a test board with any immediate winning move. The program made the move (1, 2) as it's the immediate winning move.

```
$ ./tictactoe test.board out.board

<initial state>
O |   |
----------
X | X |
----------
O |   |

Winning move by AI (User X): 1 2

<state after best move>
O |   |
----------
X | X | X
----------
O |   |
<winning results>
You Lost!! (User O).
```

The below example is to conduct the full with AI interactively. In this example, User X is AI, and User O is human. The game was played to lose intentionally.

```
$ ./tictactoe test.board out.board --play

|   |
----------
X | O |
----------
  |   |
Do you want to start? (y/n): n
Best move by AI (User X): 0 0

X |   |
----------
X | O |
----------
  |   |

Enter the x, y (0 - indexed) (User O) : 2 0

X |   |
----------
X | O |
----------
O |   |

Best move by AI (User X): 0 2

X |   | X
----------
X | O |
----------
O |   |

Enter the x, y (0 - indexed) (User O) : 2 2

X |   | X
----------
X | O |
----------
O |   | O

Best move by AI (User X): 0 1

X | X | X
----------
X | O |
----------
O |   | O

You Lost!! (User O).
```