School of Computer Science
The University of Adelaide

Artificial Intelligence Assignment 1

Semester 1 2021
Due 11:59pm Tuesday 30 March

I see only one move ahead, but it is always the correct one.

José Raúl Capablanca
World Chess Champion, 1921–1927

1 Tic-tac-toe

Write a program (in C/C++, Java or Python) that can play tic-tac-toe based on the minimax algorithm.

- In the case of C/C++, apart from the program source file, you must supply a makefile (named `Makefile`) with a rule called `tictactoe` to compile your program into a Linux executable binary named `tictactoe.bin`. Your program must be able to be compiled and run as follows:

```
$ make tictactoe
$ ./tictactoe.bin [state] [path]
```

- In the case of Java, write your program in the file `tictactoe.java`. Your program must be able to be compiled and run as follows:

```
$ javac tictactoe.java
$ java tictactoe [state] [path]
```

- In the case of Python, write your program in the file `tictactoe.py`. Your program must be able to be compiled and run as follows:

```
$ python tictactoe.py [state] [path]
```

Note: the web submission server may not be running the latest or your favourite version of Python. If you choose Python for this assignment, you are responsible for checking and ensuring compatibility of your and the server's Python versions.

The input to the program is a string `[state]` that encodes a board state, as well as the path `[path]` to an output text file. We will use raster scanning to encode a board state. For example, the state below

o	x	x
x	o	
o	x	

is encoded as the string `oxxxo-ox-`, i.e., raster scan the board left-to-right then top-to-bottom, and represent a blank space with a '`-`' character.

Given the input parameters, your program must be able to

1. Determine if there are further moves possible starting from the given input state. If there are no further moves, your program should quit immediately and not print or write anything.

2. If there are further moves, determine whose turn it is, i.e., either Max's (`x`) or Min's (`o`). Assume that Max is always the first to move at the start of the game. Remember that the input board state can be completely blank.

3. For the current player to move, calculate the best move based on the minimax principle. Then print to the *terminal* or *command prompt* the board state resulting from the best move. Note that a win to Max has utility 1, draw has utility 0, while a win to Min has utility -1. For example, given the input state `oxxxo-ox-` from above, your program should behave as follows

   ```
   $ ./tictactoe.bin oxxxo-ox- /home/tj/visited.txt
   oxxxo-oxo
   ```

   (modify the program invocation above accordingly for Java or Python) which corresponds to making the move

o	x	x
x	o	
o	x	o

Further, your program must print to the output file `[path]` the state of all the nodes visited when traversing the game tree (*excluding* the root node which contains the input state), as well as the *minimax value* of each visited node. Following the above example, the file `visited.txt` should contain

```
oxxxooox- 0
oxxxoooxx 0
oxxxo-oxo -1
```

Note that different nodes in the search tree may correspond to the same state (in other words, different sequences of moves can lead to the same outcome), so a state may be listed more than once. However, do not list states that are not visited.

If there are more than one move with the optimal minimax value, choose the move that fills in a blank that occurs the earliest in the raster scan order.

CAUTION: Even for a simple game like tic-tac-toe, the game tree generated can be huge. At this stage, be conservative when testing your program by inputting only states with a small number of remaining moves. Once you are convinced that your program always terminates in a finite number of steps, test it with a blank state.

1.1 Alpha-beta pruning

Rewrite your program above by incorporating alpha-beta pruning. Alpha-beta pruning is enabled by adding the string `prune` to the input arguments, i.e.,

```
$ ./tictactoe.bin [state] [path] prune
```

(modify the program invocation above accordingly for Java or Python). Alpha-beta pruning should result in exactly the same optimal move. However, the list of nodes visited under pruning will be smaller than or equal to the list of nodes visited without pruning. Make sure that your program satisfies these properties.

The amount of pruning, however, depends on the order in which the child nodes (moves) are visited at a particular node in the game tree. In this assignment, to ensure consistent behaviour, child nodes must be visited in the order that blank cells earlier in the raster scan are filled in first. For example, at a node with state `oo--xx---`, for alpha-beta pruning the child nodes must be visited in the order `oox-xx---`, `oo-xxx---`, `oo--xxx--`, `oo--xx-x-`, and `oo--xx--x`.

1.2 Early termination

For further speedups, it is useful to cut-off/terminate early the minimax tree traversal based on a heuristic or evaluation function. However, the cost of early termination is to introduce inaccuracies in the minimax algorithm, and may yield suboptimal moves.

A possible evaluation function for tic-tac-toe is as follows:

$$E(s) = M(s) - O(s),$$

where s is a board state, $M(s)$ and $O(s)$ are respectively the number of *possible* winning lines for Max and Min after state s. For example, if s is the state

x	o	

$M(s) = 4$, since the following winning lines are possible for Max after s:

x		
x	o	
x		

x	x	x
x	o	

x	o	
x	x	x

		x
x	o	x
		x

Similarly, $O(s) = 6$ yielding $E(s) = -2$. Evidently s is more advantageous for Min!

Implement $E(s)$ and use it for early termination of minimax. Your program must accept an additional parameter as follows

```
$ ./tictactoe.bin [state] [path] prune [ply]
```

(modify the program invocation above accordingly for Java or Python) where `[ply]` is a positive integer that specifies the maximum number of moves (in total from both players) to look ahead. Specifically, if a node s on a search path is `[ply]` moves away from the root node, treat s as a leaf node and evaluate the "utility" of s using $E(s)$. The approximated utility is then "backed-up" as per the original minimax algorithm. You must use alpha-beta pruning in this version of the program.

If a node along a branch contains an end state *at* or *before* the maximum number of moves `[ply]` is reached, terminate the recursion at the node, but take the **evaluated utility** of the node (i.e., can be out of the set $\{-1, 0, 1\}$ instead of the true utility.

1.3 Interesting test cases

Try the different variants of your program on the following inputs. Do the results surprise you? Why?

x	o	
o	x	

x	o	
	x	

1.4 Web submission

You must submit your program on the Computer Science Web Submission System. This means you must create the assignment under your own SVN repository to store the submission files. The SVN key for this submission is
`2021/s1/ai/Assignment1`
The link to the Web Submission System used for this assignment is
`https://cs.adelaide.edu.au/services/websubmission/`

For more details on the online submission procedures including SVN visit the home page of the school and look for SVN information under the "For Students" tab.

1.5 Due date and late submission policy

This assignment is due by **11:59pm Tuesday 30 March**. If your submission is late the maximum mark you can obtain will be reduced by 25% per day (or part thereof) past the due date or any extension you are granted.

1.6 Grading

I will compile and run your code on different tests. If it passes all tests you will get **15%** (undergrads) or **12%** (postgrads) of the overall course mark.

1.7 Using other source code

You may **not** use other source code for this assignment. All submitted code must be your own work written from scratch. Only by writing the solution yourself will you fully understand the concept.

2 Three Men's Morris

Note:

- This section is **compulsory** for postgraduate students. It gives you the remaining **3%** of the assignment component.

Three Men's Morris is a played on a 3×3 board. Each of the two players is allowed three game pieces. The objective is to place the three pieces on a straight line (horizontally or vertically) on the board. The game begins with a blank board, and the players take turns to place the pieces one-by-one on the board. If the game has not ended by then, the players take turns to move the pieces until one of them wins. Each piece can be moved horizontally or vertically (but not diagonally) to an adjacent empty location.

We will use symbol `x` for Max and symbol `o` for Min. Raster scanning is used to encode the board state into a string, with '`-`' to represent blank spaces. For example, the following board state is encoded as `xxo-ox--o`.

x	x	o
	o	x
		o

Write a program (in C/C++, Java or Python) that plays Three Men's Morris based on the minimax principle.

- In the case of C/C++, apart from the source file, you must supply a makefile (named `Makefile`) with a rule called `tmmorris` to compile your program into a Linux executable binary named `tmmorris.bin`. Your program must be able to be compiled and run as follows:

```
$ make tmmorris
$ ./tmmorris.bin [state] [path] [ply] [turn]
```

- In the case of Java, write your program in the file `tmmorris.java`. Your program must be able to be compiled and run as follows:

```
$ javac tmmorris.java
$ java tmmorris [state] [path] [ply] [turn]
```

- In the case of Python, write your program in the file `tmmorris.py`. Your program must be able to be compiled and run as follows:

```
$ python tmmorris.py [state] [path] [ply] [turn]
```

The input parameters are defined as follows: `[state]` is a string that encodes the board state; `[path]` specifies the path of an output text file into which the state of visited nodes and its minimax value should be written; `[ply]` is an integer that specifies the number of moves to look ahead (if set to a large value, the program will explore the full depth of the game tree); and `[turn]` can be either `x` or `o`, and this tells your program whose move it is now (after the first six moves, this information cannot be calculated from the board state). By default, use alpha-beta pruning in the program.

Given the input parameters, the program must calculate the best move and write the board state resulting from playing the best move onto the terminal. If there no further moves possible after the input state, do not write or print anything.

2.1 Evaluation function

By now it should be clear that the "magic" in game playing is in specifying the heuristic or evaluation function (also, the utility value at the end states). Once these quantities or functions are determined, the rest of the program is pretty mundane, at least for simple games. Design and test your own evaluation function for Three Men's Morris. Try and see if you can get your program to beat a human player.

2.2 Submission policies

Submit your program in the same way as the previous section. The due date and code reuse policies are also the same.

2.3 Grading

I will test your program manually to see if it satisfies the required functionalities.

∼∼∼ The End ∼∼∼
by Tat-Jun Chin, 1 Mar 2021