TIC TAC TOE GAME by mateusz jakusz (sba21173)

https://github.com/bloobsky/TicTacToe-Python-AI-minimax

Contents

[Introduction 2](#_Toc67924182)

[Insane AI (Minimax Algorithm) 2](#_Toc67924183)

[Why Minimax? 3](#_Toc67924184)

[Easy AI -> No Algorithm 3](#_Toc67924185)

[Design Decision 3](#_Toc67924186)

[Structure of the code 3](#_Toc67924187)

[Contributions 3](#_Toc67924188)

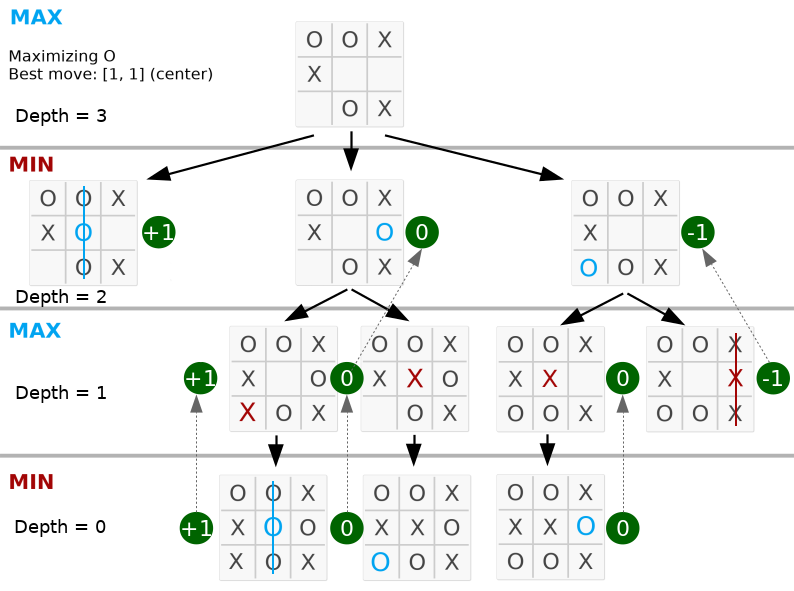
[References 4](#_Toc67924189)

# Introduction

To solve game problems using AI, I have to introduce the concept of a game tree followed by minimax algorithm. The different situations(states) are represented by nodes in the game tree – something like planning problems. In the game tree, the nodes are arranged in levels that correspond to each player's turns in the game so that the “root” node of the tree (usually depicted at the top of the diagram) is the beginning position in the game. In tic-tac-toe, this would be the empty grid with no ‘X’ or ‘O’ played yet. Under root, on the second level, there are the possible states that can result from the first player’s moves, be it X or O. We call these nodes the “children” of the root node. Each node on the second level, would further have as its children’s nodes the states that can be reached from it by the opposing player's moves. This is continued, level by level, until reaching states where the game is over. In tic-tac-toe, this means that either one of the players gets a line of three and wins, or the board is full and the game ends in a tie (Heineman, Pollice and Selkow, 2009).

# Insane AI (Minimax Algorithm)

The algorithm search, recursively, the best move that leads the *Max* player to win or not lose (draw). It considers the current state of the game and the available moves at that state, then for each valid move it plays (alternating *min* and *max*) until it finds a terminal state (win, draw or lose). As we mentioned in introduction, I will provide game tree for the game. Below, the best move is on the middle because the max value is on 2nd node on left (Cruz, 2020).



# Why Minimax?

In theory, minimax is a decision rule used in artificial intelligence or game theory. It can be also used towards philosophy, statistics. Originally formulated for n-player [zero-sum](https://en.wikipedia.org/wiki/Zero-sum) [game theory](https://en.wikipedia.org/wiki/Game_theory), covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision-making in the presence of uncertainty. In our case it fantastic to use it for the tic tac toe game. Other games it could be use for is: Go, Backgammon.

# Minimax Pseudo Code

minimax(state, depth, player)

if (player = max) then

best = [null, -infinity]

else

best = [null, +infinity]

if (depth = 0 or gameover) then

score = evaluate this state for player

return [null, score]

for each valid move m for player in state s do

execute move m on s

[move, score] = minimax(s, depth - 1, -player)

undo move m on s

if (player = max) then

if score > best.score then best = [move, score]

else

if score < best.score then best = [move, score]

return best

end (Heineman, Pollice and Selkow, 2009).

# Easy AI -> No Algorithm

In the game presented player has opportunity to play with novice AI. I am not sure if we can call it real AI as it does not use algorithm. Instead, it uses a built-in function called random. In case of our application, it looks for not occupied space and put a sign ‘X or O’. The AI will not block you, so it is practically impossible to lose the game against it.

# Design Decision

The game could be presented in terminal command; however, I went one step ahead and did some GUI for it. I used 2 modules, pygame – for creating and maintaining a visual board and enables user to use mouse to play (instead of typing it manually). The second module is called pygame\_menu that enables programmer to create simplified menus without digging much in the code.

# Structure of the code

The entire game is written in Python. The app was divided into 4 main modules and requirements.txt file:

1. launch.py – simple code that will start the game.
2. menu.py – code presented in that file contains code that allow to present menu for the game.
3. start.py – code that contains actual game, minimax algorithm, two AI agents (one for minimax, other is plain random)
4. gamesettings.py – all the constants that are required for rendering the game.
5. requirements.txt – file that contains all requirements for the proposed project.

All the code I presented is well commented thus I am not going to include step by step comments. Instead, just read the code and its comments through the files.

In some cases, some of the tutors are required to use Class so it will present as OOP, however the game is so simple that would be pointless to put everything in Class. Instead, I make each of the game part in individual modules.

# Contributions

I would like to say thank to a python community:

Cledersonbc (https://github.com/Cledersonbc/tic-tac-toe-minimax) for presenting minimax algorithm.

CodingSpot (https://www.youtube.com/channel/UCLqXQLK6zKZg0trhanjAkkQ) for information how to use PyGame.

Pablo Pizarro R. @ppizarror (https://pygame-menu.readthedocs.io/en/4.0.1/) for creating pygame\_menu model to make my life easier when designing the game.

# References

Heineman, G., Pollice, G. and Selkow, S., 2009. *Algorithms in a nutshell*. Beijing [etc.]: O'Reilly.

Cruz, C., 2020. *Cledersonbc/tic-tac-toe-minimax*. [online] GitHub. Available at: <https://github.com/Cledersonbc/tic-tac-toe-minimax> [Accessed 30 March 2021].