A study in madness

Lenny Brown

February 16, 2016

1 Introduction

1.1 Background

Tic Tac Toe is a game about getting three things right and those three things had goddamn better be in a row.

Nine Board Tic Tac Toe, or NBTTT as we in the biz call it, is pretty similar, except when one of you moves, the next person has to choose *their* move in the board of the corresponding tile that you've chosen. For obvious reasons, everything goes crazy from here. You'll see why in the next section.

1.2 AI is when sh** gets real

The study of Artificial Intelligence is considered by some to be, quite difficult.

Finding the most optimal move is very much a search problem (which we can infer from how we have to find it). This is very hard on larger sets (hence our upcoming cool strategy), though on 3X3, the entire board and optimal solution can actually be found. For the 9 board, it's infeasible to find the optimal move.

2 Strategy

2.1 What the gosh darn heck is my strategy?

Randomness, dude. As simplistically as possible, we look at each possible next action and generate a bunch of random samples based off it. The choice that leads to the most wins and fewest losses is the one we choose. This is a good idea because it means that the largest percentage of goal states can be found down this path.

2.1.1 Downsides?

1. The best option might not be on the path that leads to the most goal states
2. We'll need to do a lot of frickin trials to make sure we've got the best choice
3. I have no idea how to prove if it's working or not

2.1.2 Upsides!!

1. We can base our utility costs off actual wins
2. We only actually need to check for goal states which is quick
3. It's really neat and theoretically simple

3 Implementation (simple stuff first)

3.1 General Note

All of the difference between the two games is handled in every method and both game use the same calls!

3.2 The Game Class

3.2.1 In General

Controls the actual movement of the board(s) through the game from the head to the bottom of the tree. The human play actually doesn't every

interact with this class and the actual move takes place outside of it. It is only for simulation

3.2.2 The Board

The board is an array of 0s, 1s, and 2s, where 0 is O, 1 is X, and 2 is Unchosen. Nine board is just 9 of these arrays in an array. Super simple stuff.

3.2.3 Goal Checking

I literally made a series of Regex checks that check if the state we have matches a possible win state for either side (X, O). It's really fast. It's great.

3.2.4 Next Actions

Are calculated in the action thing which applies the change to the board, switches the player, the sends it recursively down. As each call finishes, it undoes those applications that the board that is sent in will be the same as the board that comes out.

3.3 Menu

It does the menu stuff. You get to decide how long (roughly) it should take the application to make its choice and also what game you'd like to play. You don't get to go first or be X. No I don't know why I didn't implement these things, I'm the worst.

3.4 Start

If it's the human's turn, they get to choose a thing. If it's the AI's turn, it computes the best move and then chooses it. Player switching is done after each move. If it's NBTTT, the board is switched to match the move

3.5 Computing The Best Move

In essence, all we do in this method is calculate every available next-move permutation and then go wildly from there racking up percentages and averages left and right. The next available move thing accounts for if a move will cause the opponent to win the game and skips that move. For some reason

it messes up occasionally. I gave that search about 2 hours and declared it non-essential. If you see a bug, call me at 609-217-2460, I want to know where it is. Screw that bug. Seriously.

3.5.1 I made it a pool thing

I realized that all those checks were taking up a lot of time and energy, and time is money, and money is how we exchange goods and services. So I made it faster by implementing multiprocessing. Essentially, I created a global pool of 9 workers. As many as necessary take a next-action, board deep-copy, and some other info, and trial test that change for the entirety of the choice time (as opposed to choice time divided by number of choices). I used the Python Pool apply-async method, which allowed me to asynchronously finish all my different trials and return the best average. It was great. Allowing for the starting of the extra processes, it still comes in at about 4 times the amount trials per unit of time.

4 Summary

I can say just two things about what I have created. It creates something that vaguely resembles tic tac toe, and it uses multiprocessing to be fast, fast, fast.

5 Acknowledgements

Just me, but also I guess we're all a part of this. So maybe you have a motivation for liking this????