Introduction

For the second week, I added a depth-first minimax agent to the tic-tac-toe program. In its original version, nothing was optimized, and the agent would expand the entire minimax tree. Then, as this program took around 40 seconds per game, three optimizations were successively added upon the program.

The first optimization was eliminating the `deepcopy` method, where the board was originally copied for each expansion. Instead, moves were made in-place on the board, and were undone when that branch had been checked.

On top of this, I began caching board states. Hence, if a node had already been checked and cached, it would not be expanded. This approach recognizes that the configuration of nodes is truly a graph (not a tree), as there are different permutations of moves that lead to the same state. The cache was cleared at the end of each game.

The final optimization was prioritizing early wins. Instead of scoring each leaf node (end-of-game node) simply by whether it was a win/loss/draw (which were scored 1/-1/0), this method would divide the original score by the depth of the leaf node, hence assigning greater priority higher nodes. The intent here is to end the game as early as possible while still attaining the same result (in terms of a win/loss/draw).

Results

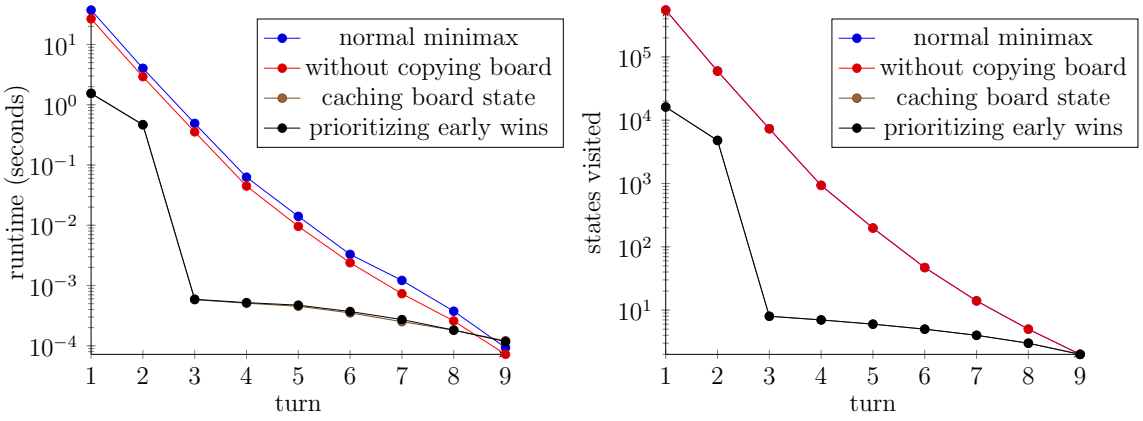
Minimax vs. Minimax

With two minimax agents, no randomness was involved, and hence the game was completely deterministic, ending in configuration below (a draw).

x	x	o
o	o	x
x	o	x

It should be noted that, at the first move, there were multiple moves scored `0` in minimax, but the first (top left) was chosen. This even occured when spots were chosen by priority of depth, as all draws will have depth 9 (a draw is always on a full board).

The two log-scaled graphs below show the results, over 10 games in each of the successive optimizations, of average runtime and states visited. Note that each successive optimization was applied on top of the previous.



In the first graph, we see a general exponential trend: as there are more open squares (at earlier moves), the time increases exponentially (reflecting that minimax trees are exponential in terms of the branching factor). Eliminating board copies slightly reduced the time, in proportion to how many moves were checked in the turn. Caching states reduced the time even more, and drastically after the first two turns (after which the rest of the game would already be cached). These later moves (3-9) are linear, as they depend merely on how many top-level moves (pre-cachced states) they need to check. Although it's impossible to tell with just two points, moves 1-2 are likely less-than exponential, as extrapolating along a linear form on the scaled graph would intersect the more-costly methods. That is, if the cache had been reset at each move, the graph would reflect this less-than exponential form (which makes sense, as branches would be removed from the tree). The last optimization actually slightly increased the time (a negligable amount), as taking depth into account adds constant time at each node.

The same exponential trend is shown in the states-visited graph for the original algorithms, as well as the sub-exponential trend for the cached methods. The only optimization that made a difference in terms of states, as expected, was caching the states.

Minimax vs. Random

When playing against a random agent, the minimax agent never loses, as the worst case scenario (seen in minimax vs. minimax games) for any optimal agent is a draw. Below is a chart of the winning percantages (from 100 games each) of minimax agent when playing as either the first or second player. Surprisingly, prioritizing earlier wins seems to increase the chance of a draw, which is likely an effect of the game not being deterministic (i.e., when the expected outcome is a draw, prioritizing early wins a few branches down happens to disfavor branches with more winning chances). As expected, the minimax agent going second reduces its likelihood of winning, as it has less moves and hence less control of the board.

	minimax first	**random first**
no prioirty	99%	85%
prioritizing early wins	99%	76%

The following graphs depicts the effect of prioritizing earlier wins. Games tend to be shorter, whether the minimax agent goes first or second, when it prioritizes earlier wins. Again, games last longer in general when going second, due to the fact that the 'o' player has less control of the board.