Introduction

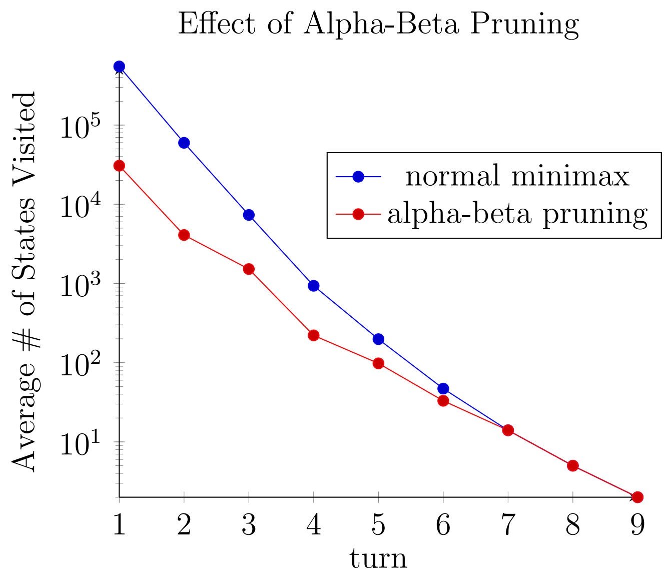
For the third week, I worked on alpha-beta pruning for the minimax algorithm. Once this was completed, I updated the board hash to store a `hash` variable, which updates with each move, rather than rehashing completely each time. When using the alpha-beta algorithm, only some states could be cached (i.e., those without pruned descendants), in order to avoid copying faulty scores across the tree to a state where pruning should not be applied.

I also added an evaluation function, by which moves would be ordered. This works separately from the result function (leaf nodes), but does begin by checking if the game is in a terminal state and returning $-\infty$, 0, or ∞ when appropriate. Otherwise, all game lines are considered, and if the opponent has the opportunity to win in the next move, $-\infty$ is returned. If the current player has a fork (two opportunities to win, so the opponent has no chance), ∞ is returned. Otherwise, for each line that the player or opponent has control over, the number of squares taken in that line is added/subtracted (respectively) from the score. This score is then returned as the evaluation.

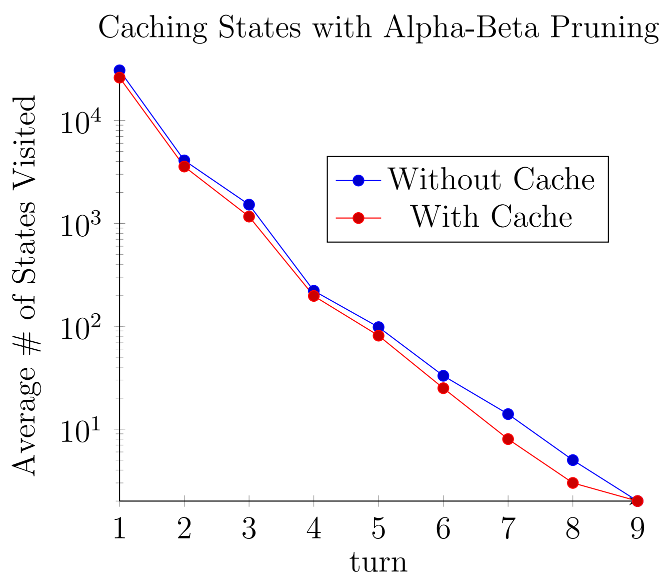
As shown by the following results, more improvements may need to be made to the evaluation function. This can be achieved through further testing and experimentation.

Results

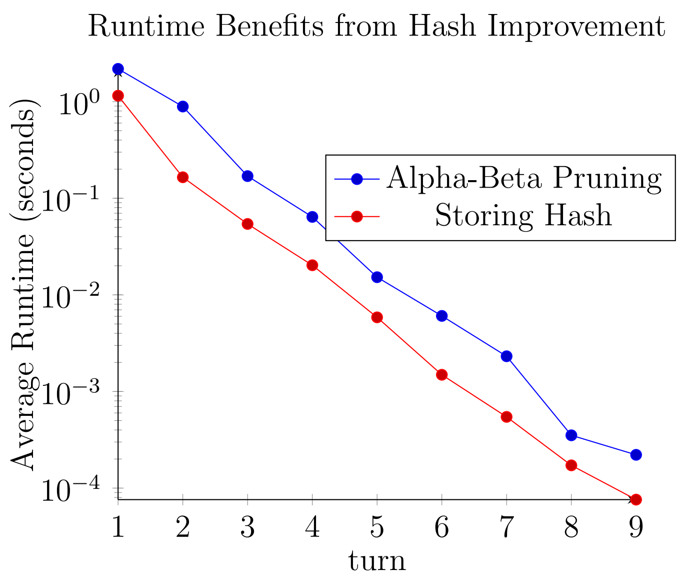
The following graph shows the result of applying pruning (without cache). This is most effective on the first move, where the pruned method visits only about 5% as many states as the original minimax algorithm.



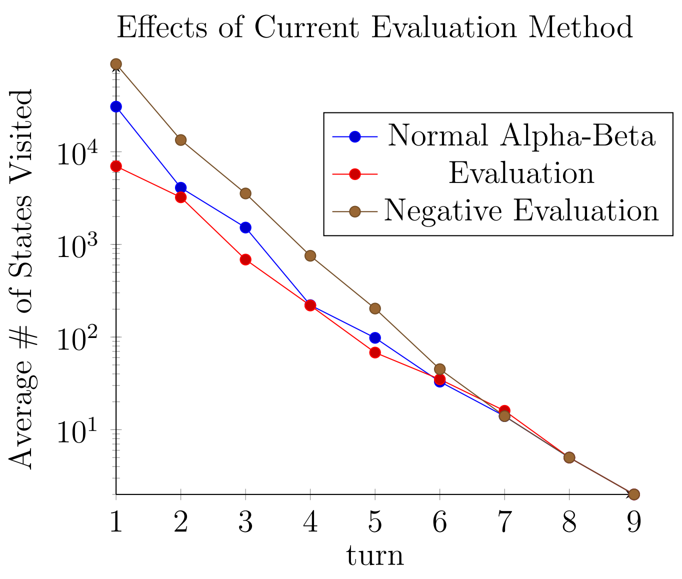
Caching did eliminate a few thousand states, but did not make a large difference due to the fact that not many states were cached. Note that there are still some uncached states visited after move 2, due to the fact that not all states were cached (those with pruned descendants were not).



The next graph highlights that the new hashing method (storing the hash rather than rehashing) has a consistent runtime improvment.



The following graph shows a comparision of states visited from a normal alpha-beta search, and two searches where moves are ordered by the current evaluation, in both a normal/positive and a reverse/negative fashion. All of these were done without a cache. There is a fairly consistent improvement caused by move ordering. Although the scale under-exaggerates it, move ordering visits just 25% as many states as the default order.



This evaluation results in the following board state, with the first move occuring in the middle cell (with the highest heuristic):

o	**x**	**x**
x	**x**	**o**
o	**o**	**x**

The reverse evaluation ends in this state, with the first move being in the middle of the top row (tied for lowest heuristic value):

o	**x**	**o**
x	**o**	**x**
x	**o**	**x**