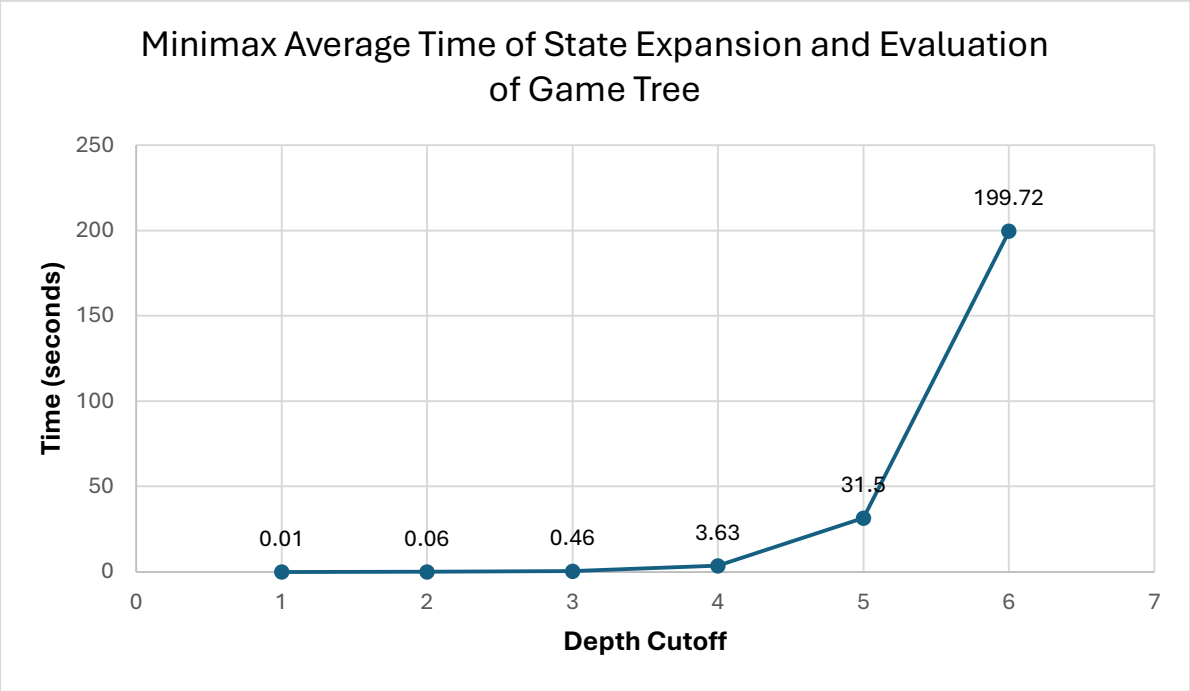
Assignment 1 Report

Jeremy Piperni

260990011

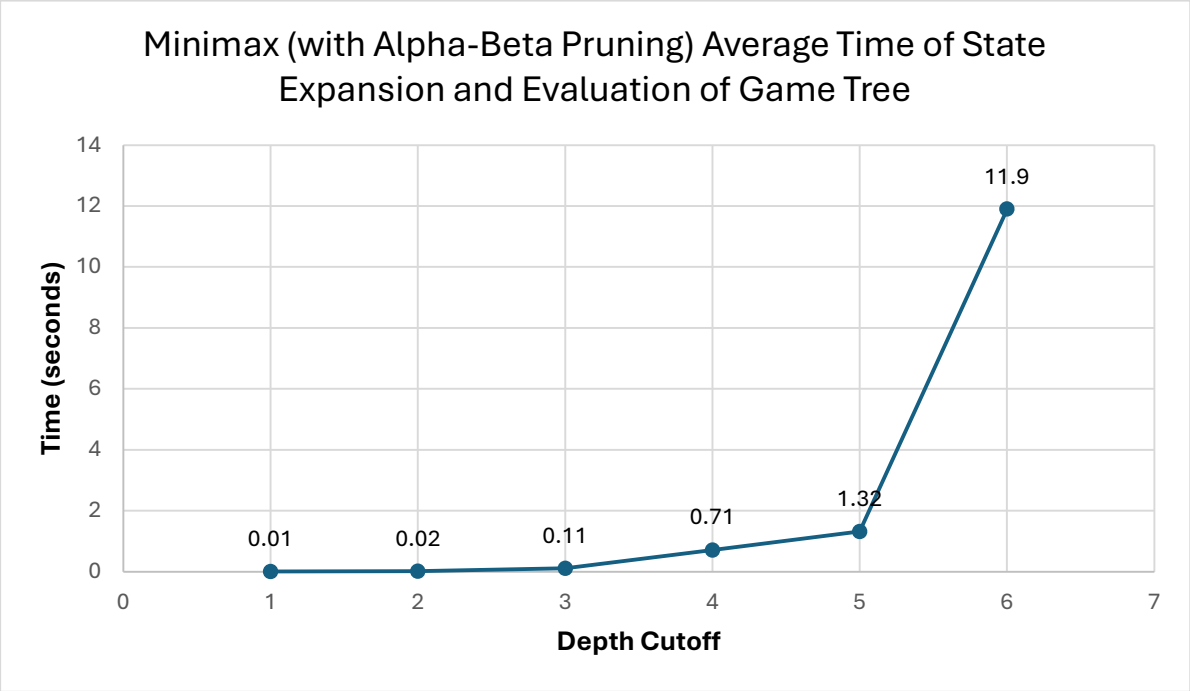
Please note, the **numpy** package was used for this assignment. All times were calculated on the Trottier Linux machines (ssh).

Part 1



The minimax algorithm performs very fast for the state space expansion of depth cutoffs of up to 3. It starts slowing down at a depth cutoffs of 4 and 5. A depth cutoff of 6 is extremely slow, taking over 3 minutes to calculate a move. Alpha-Beta Pruning drastically improves the performance of the algorithm. The state space expansion for the first 3 depth cutoffs is still extremely fast but we see the major increase of performance at a depth

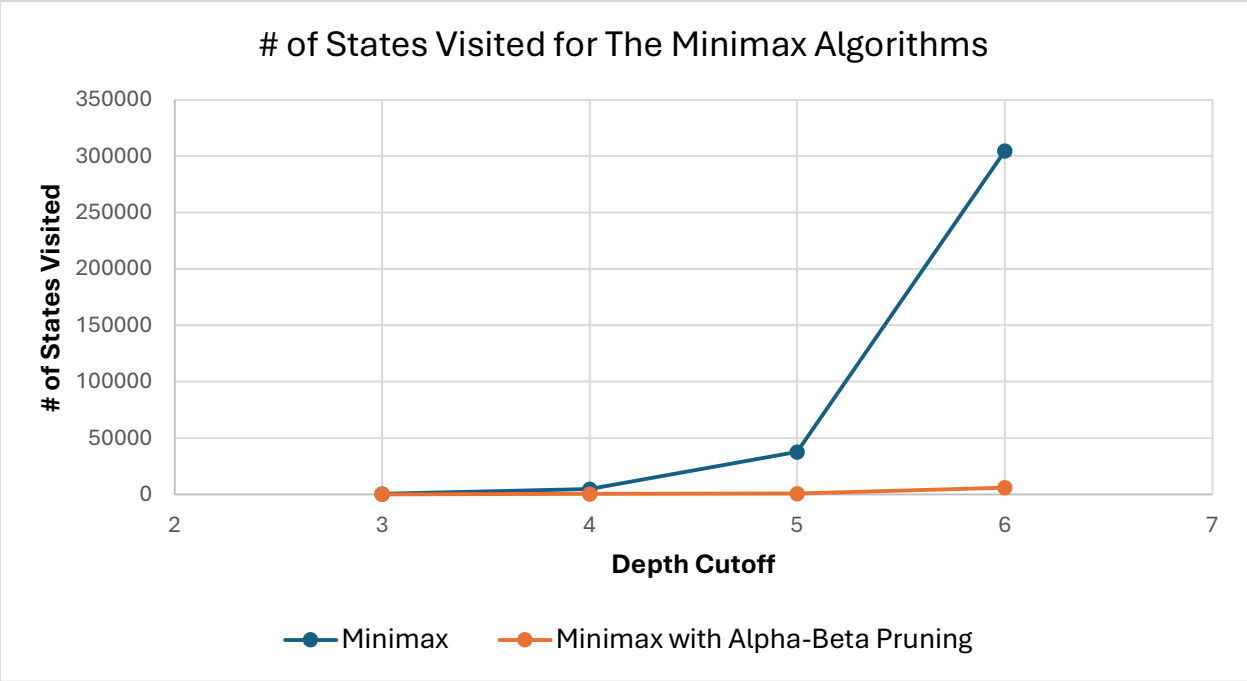
cutoff of 4. With Alpha-Beta Pruning, a state expansion with depth cutoff of 4 is 5x faster, with depth cutoff of 5 is 23x faster, and with depth cutoff of 6 is 16x faster.



The following table demonstrates the # of states visited by the algorithm for different depth cutoffs when starting from Turn 1 as “White”. Results for both the minimax algorithm and minimax with alpha-beta pruning are shown.

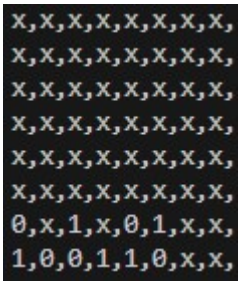
Depth Cutoff	Minimax	Minimax with Alpha-Beta Pruning
3	584	94
4	4680	371
5	37744	837
6	304560	6051

Implementing Alpha-Beta Pruning visits much less states than Minimax, which is not surprising after seeing how fast it runs compared to plain minimax. The following is a graphical representation of the table above.



Changing the order of which states get generated in our algorithm will not result in visiting more or less states for the simple minimax algorithm. That is because by definition and implementation simple minimax will visit every state possible even if it finds the best terminal node possible. For minimax with alpha-beta pruning, changing the order of which states get generated can result in a different number of states being visited. If we order the state expansion starting with moves that we think will be the best, more pruning will occur when we start getting close to terminal nodes. Currently my algorithm starts the state expansion with Drop moves starting from column 1 to 8, then it finishes it off by checking if any Slide moves are possible. I believe that Slide moves might be better than Drop moves,

so I'll change the implementation by having the state expansion start with Slide moves. Here is the result for the number of states visited after this change; the playing board below is the starting board for this state expansion test.



Depth Cutoff	Minimax (Drops First)	Minimax (Slides First)	Minimax Alpha-Beta Pruning (Drops First)	Minimax Alpha-Beta Pruning (Slides First)
3	835	835	210	242
4	7485	7485	493	632
5	57334	57334	3504	4185
6	603747	603747	5440	7303

As can be seen, the ordering for simple minimax does not change the # of states visited while it does for minimax with alpha-beta pruning. Note that my earlier hypothesis of Slide moves being better seems to be false, as fewer states were visited when Drop moves were ordered first.

Part 2

After considerable testing, I've made changes to the program to increase the chances of winning. I've changed the order of possible moves for the state expansion. Although my testing in the previous section showed that having drop moves first led to fewer states being visited, I believe that this is only true for the early game when slide moves aren't as valuable; So, I've decided to order Slide moves first. I also changed the Drop Order to the following sequence: col = [4,5,3,6,2,7,1,8]. The reason behind this is that playing the middle of the board is more beneficial as it opens up the possibility for more runs. For the heuristic function, I've only tweaked it slightly to the following

Heuristic Function when playing White (w: white):

H(n) = (3 * w runs of three) + (w runs of horizontal twos) + (w runs of twos)

Heuristic Function when playing Black (w: white, b: black):

H(n) = (3 * b runs of three) + (b runs of horizontal twos) + (b runs of twos) – (3 * w runs of three)

This is an improvement over the original heuristic function as it gives more value to runs of three than runs of two. It also adds value to horizontal runs of two as they can be utilized for sliding and can block the other players from sliding. More notably, we aren't keeping track of the other players' runs of twos or threes when playing white. The reasoning for this is that we are already going to block runs if we know it'll lead to a win for the other player because of our minimax algorithm. White can also always win if it plays perfectly, so we should focus on offense. For black we must focus more on defense and try to pounce on white if

they make a mistake. So we have an extra term in the heuristic function to try to minimize white from creating runs of threes.

We know that the heuristic function will not affect the numbers of states visited for plain minimax, as all states are always going to be visited. The table below compares the number of states visited for our old heuristic vs our new heuristic function for different depth cutoffs.

Depth Cutoff	Old Heuristic Function Avg States Visited (First 10 moves)	New Heuristic Function Avg States Visited (First 10 moves)
3	120	119
4	670	786
5	2879	2279
6	9170	9390

As we can see, only at a depth cutoff of 5, does our new heuristic function led to fewer states visited than our old heuristic function. But we can see the change of heuristic function didn't drastically change the number of states visited. Even with an extremely simple heuristic function, the depth cutoff will always be a greater factor in the memory used by our algorithm. The important rule that we have to remember is that our moves must be completed in under 10 seconds to be valid. So as long as our moves follow this rule, then visiting slightly more states does not matter if we can provide a better heuristic

function (leads to more wins or faster wins). The following is a table for how many turns it took for white to win (old vs new heuristic, alpha-beta pruning)

Depth Cutoff	Moves required for white to win (old heuristic)	Moves required for white to win (new heuristic)
3	11	7
4	Black wins	7
5	8	6
6	Black wins	9

The old heuristic would lead to losses for white on depth cutoffs of 4 and 6, when theoretically white should never lose. Lastly, we will compare the logs of a game with the old vs. new heuristic function for a depth cutoff of 4.

Old Heuristic Function Log

White Move 1

D 4

Black Move 1

D 4

White Move 2

D 4

Black Move 2

D 4

White Move 3

D 4

Black Move 3

D 4

White Move 4

D 4

Black Move 4

D 4

White Move 5

D 6

Black Move 5

D 7

White Move 6

D 8

Black Move 6

D 5

White Move 7

D 1

Black Move 7

D 3

White Move 8

D 6

Black Move 8

D 2

White Move 9

D 3

Black Move 9

L 3 8

White Move 10

D 1

Black Move 10

D 5

White Move 11

D 6

Black Move 11

D 6

White Move 12

D 2

Black Move 12

D 2

White Move 13

D 6

Black Move 13

D 6

White Move 14

D 6

Black Move 14

D 6

White Move 15

D 2

Black Move 15

D 2

White Move 16

D 2

Black Move 16

D 2

White Move 17

D 2

Black Move 17

D 1

White Move 18

D 1

Black Move 18

D 1

White Move 19

D 1

Black Move 19

D 1

White Move 20

D 1

Black Move 20

D 8

White Move 21

D 7

Black Move 21

D 7

White Move 22

D 5

Black Move 22

D 5

White Move 23

R 6 7

Black Move 23

R 4 5

White Move 24

D 5

Black Move 24

L 6 5

White Move 25

L 6 4

Black wins!

0: white, 1: black

```
0,0,x,1,x,1,x,x,
1,1,x,1,x,1,x,x,
0,0,x,0,x,1,x,x,
1,1,x,0,0,1,x,x,
0,0,x,1,1,1,0,x,
1,1,1,0,0,1,1,x,
0,0,0,1,1,0,0,0,
1,1,0,0,1,0,1,0,
```

New Heuristic Function Log

White Move 1

D 4

Black Move 1

D 4

White Move 2

D 4

Black Move 2

D 4

White Move 3

D 4

Black Move 3

D 4

White Move 4

D 4

Black Move 4

D 4

White Move 5

D 5

Black Move 5

D 3

White Move 6

D 6

Black Move 6

D 5

White Move 7

D 6

Black Move 7

R 4 7

White wins!

0: white, 1: black

```
x,x,x,1,x,x,x,x,
x,x,x,1,x,x,x,x,
x,x,x,0,x,x,x,x,
x,x,x,1,x,x,x,x,
x,x,x,0,x,x,x,x,
x,x,x,1,x,x,x,x,
x,x,x,0,1,1,x,x,
x,x,1,0,0,0,0,x,
```

For the tournament, a depth cutoff of 5 will be used as it provides the best performance as well as following the rule that every move must take less than 10 seconds to compute.