Competitive Path Planning Via Minimax Algorithm and Alpha-Beta Pruning

Discrete Fall 2025
Irene Hong
Ivy Mahncke
Vivian Mak

Introduction to Minimax and Alpha-Beta Pruning

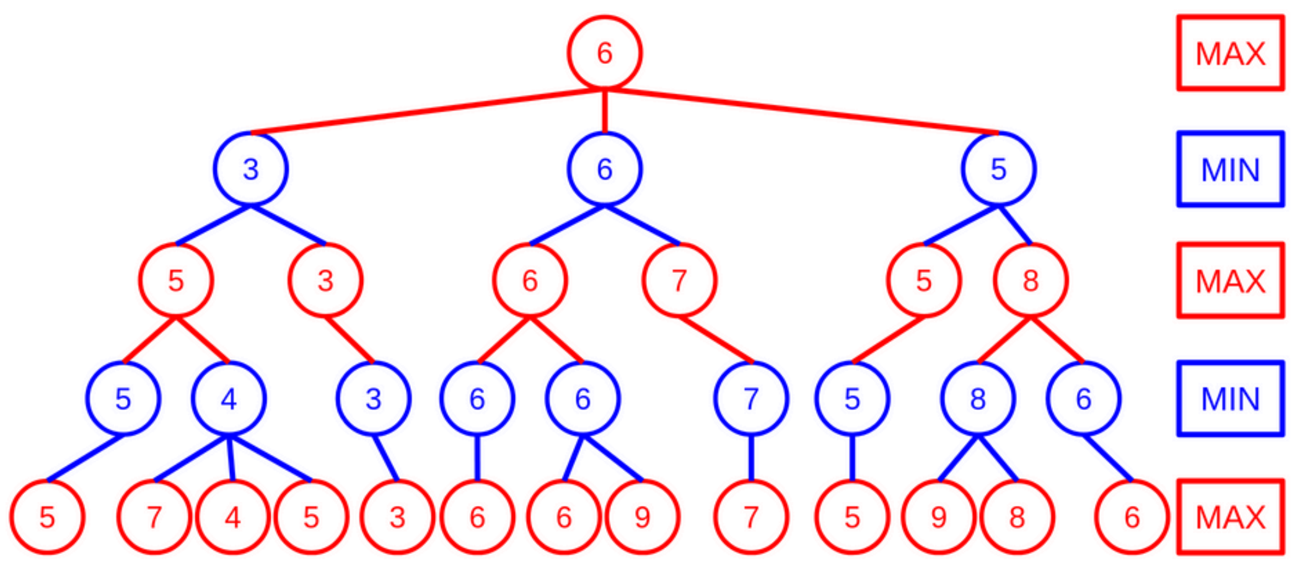
Competitive games like chess, checkers, and Go are difficult to play perfectly because the actions of your opponent are unknown. While keeping track of all possible future game states could help you decide which of your moves is best, it's nearly impossible to do so.

Can we use discrete math to guess what an opponent will do and use that information to optimize our own scores?

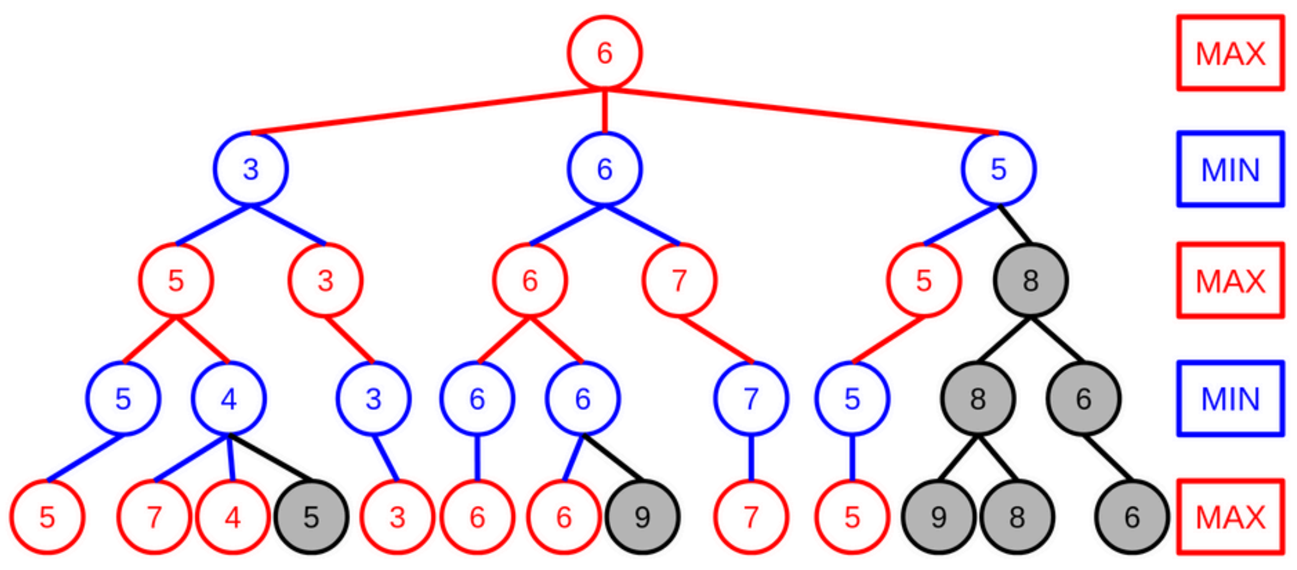
Minimax is a recursive decision-making algorithm that selects whatever move maximizes the current player's score, assuming that the opposing player will attempt to minimize that score. On each turn, the algorithm predicts future moves up to a specified "look-ahead" depth, making it capable of long-term planning. With each additional depth, the number of possible game states increases exponentially, making infinite depth exploration computationally impossible.

Alpha-beta pruning optimizes Minimax by skipping branches that will never actually occur. By always choosing the minimizing action, your opponent will force one of your options to cause a worse game state than an option you've already found. That option's branches can be safely pruned.

Minimax On Maximizer's Turn

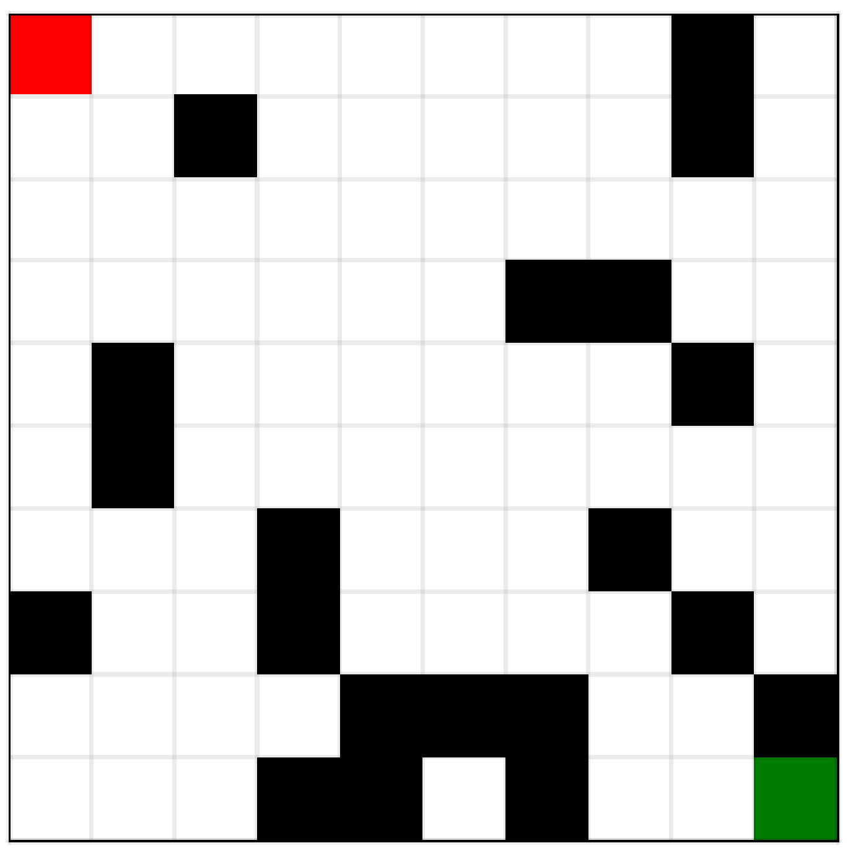


With Alpha-Beta Pruning



Methodology

Example Environment



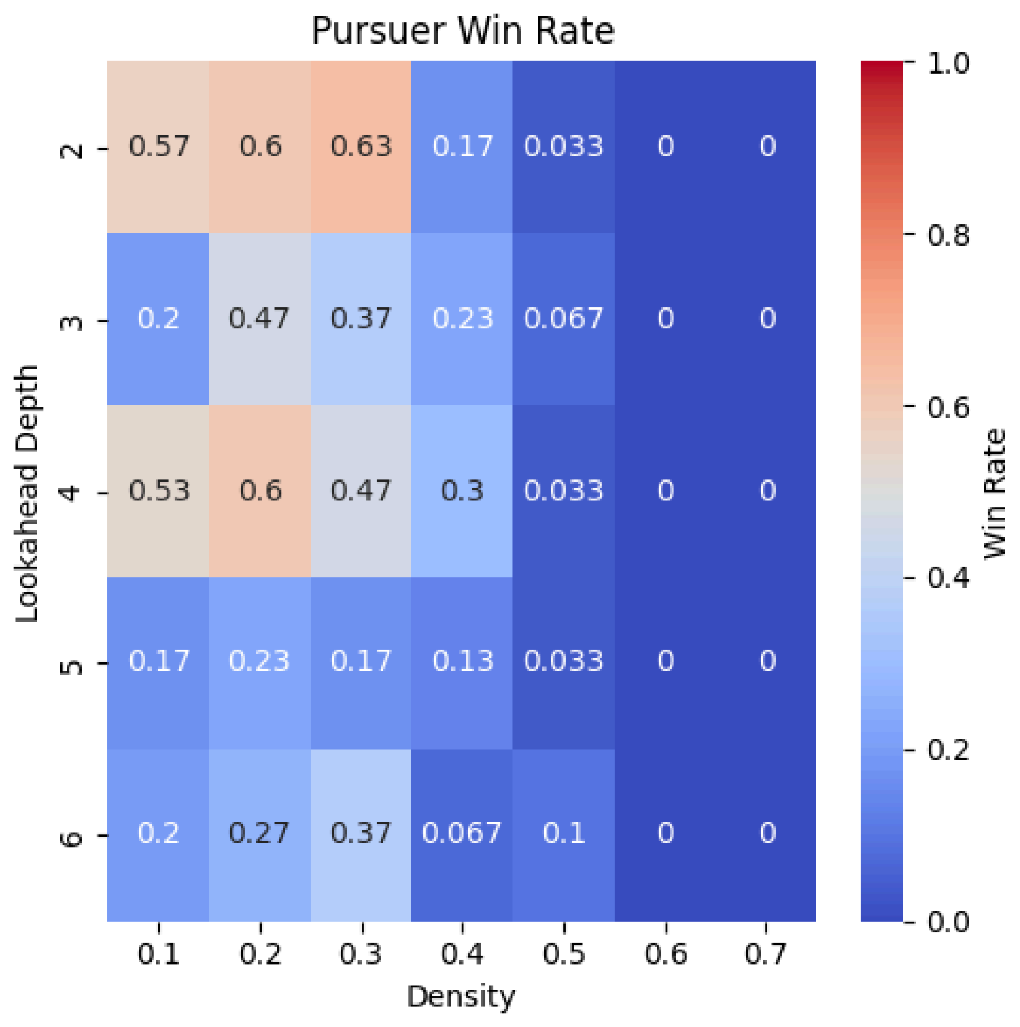
⬛ Obstacle 🟩 Evader
⬜ Non-Occupied 🟥 Pursuant

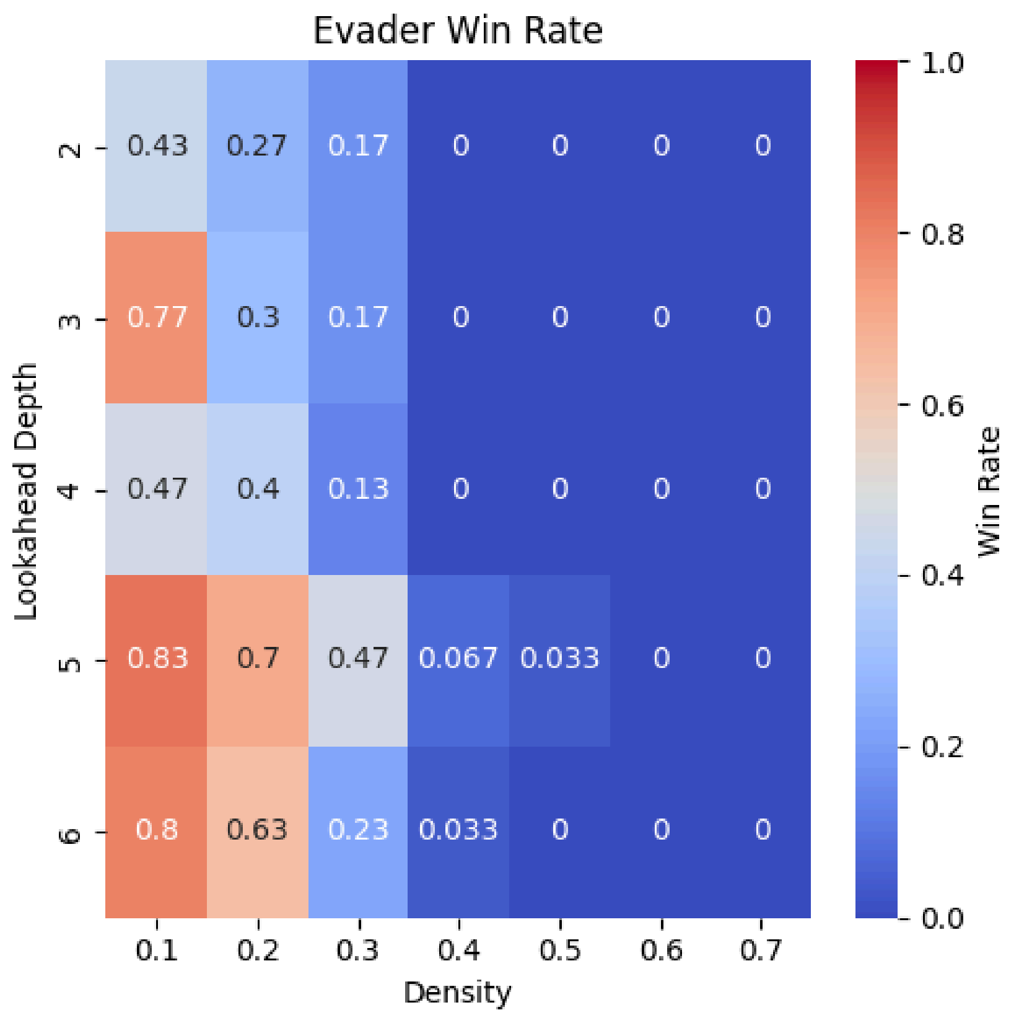
We designed a game in which two robotic agents chase or evade the other in an obstacle-dense world. In this game, the heuristic function for the Minimax algorithm is the distance between the two agents. The pursuant wins by stepping adjacent to the evader, while the evader wins by surviving for 25 consecutive turns.

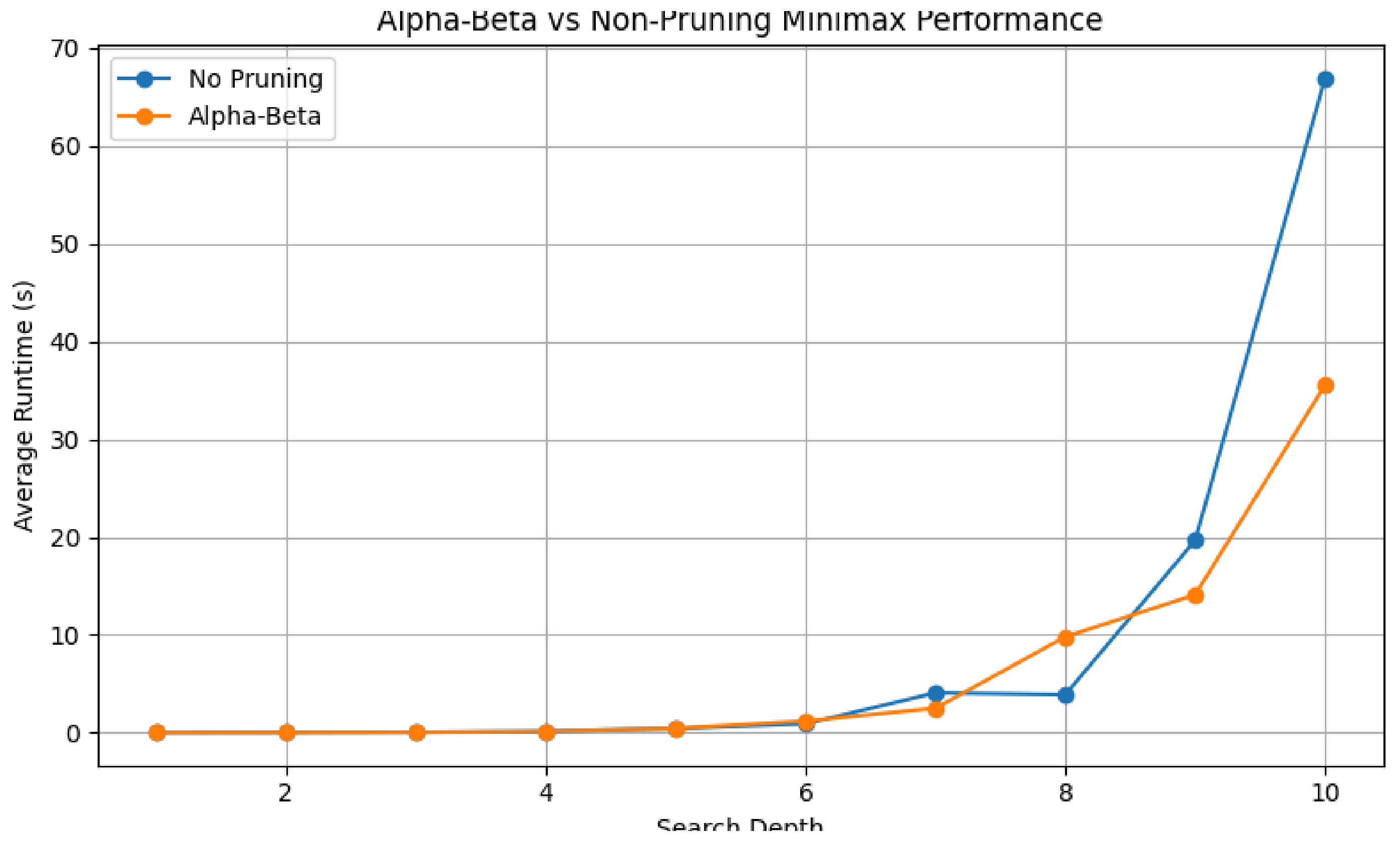
Before each agent acts, it generates a game tree representing future game states up to a certain look-ahead depth. Then, the agent calls the Minimax algorithm with AB pruning to determine which of its possible actions will get it closer to (minimize) or farther from (maximize) the other agent.

```
function minimax_ab(node, depth, α, β, maximizingPlayer)
  if depth == 0 then
     return node heuristic value
  if maximizingPlayer then
     value := −∞
     for each child of node do
        value := max(value, minimax_ab(child, depth − 1, α, β, FALSE))
        if value ≥ β then
           break (* β cutoff *)
        α := max(α, value)
     return value
  else
     value := +∞
     for each child of node do
        value := min(value, minimax_ab(child, depth − 1, α, β, TRUE))
        if value ≤ α then
           break (* α cutoff *)
        β := min(β, value)
     return value
```

Results







These two graphs show the relationship between the look ahead depth and the density, after running 100 games, and the correlation to the percentage of times the pursuer or evader has won. For higher densities, neither agents win due to the intraversable environments, resulting in a tie. However, as the environment is populated with less obstacles, the look ahead depths factor into the game outcome. The pursuer tends to win more with a smaller search depth, whereas the evader wins more often with a larger search depth. With a search depth of 4, the game outcomes tends to be more evenly split.

The average runtime for the minimax algorithm with and without alpha-beta pruning was captured over 10 different games at 10 look-ahead depths. Initially, both algorithms perform nearly identically, with alpha-beta pruning even having higher run times at certain points. This is because at lower depths, updating the alpha/beta parameters and deciding which branches to prune takes longer than looking through every existing node. However, at larger depths, the decrease in runtime is significant, since the game tree grows exponentially.

Conclusion

In this two agent game, the minimax algorithm makes the optimal move by anticipating and preparing for the opponent's counter-moves. However, the depth search calls for intensive computational power as the choices grow exponentially. The way to alleviate this is by using alpha-beta-pruning where a decision branch cuts off once a parent node has already computed a best branch based on shortest-path heuristics. Performing a test to see the percentage of win rates between the pursuer and evader yielded results that show that with lower densities, a higher search depth is advantageous for the evader, as opposed to a lower search depth for the pursuer.

Future Work

- **Corner Moves.** Increasing the amount of choices by including corners as valid moves. With this, there would be a more noticeable difference with the effects of pruning
- **Additional Heuristics.** Adding additional heuristics to each agent, this would present additional goals and create a more interesting decision tree. This would also enforce constraints like having agents avoid corners or walls.
- **Concurrency.** Lastly, turning this game into simultaneous agent decision-making since it would be more application to real-time situation.