Maddox Scott

Benedict Quartey

Summer22 S02 Artificial Intelligence

5 August 2022

Minimax Optimization for Turn-based Games

In many activities where one player is attempting to win by ensuring that the other player loses, it if often unclear which set of actions will provide the highest probability of winning. However, by developing a search algorithm capable of evaluating every possible outcome of the activity, it would be clear which options a player should take to ensure their victory. For this assignment, a minimax search algorithm will be utilized to solve turn-based games such as tic-tac-toe and connect four.

These games were chosen as the medium for testing the search algorithm, as they are simple, quick games with relatively few options for a player to take each turn. By choosing a game that limits the number of actions each player can take and has a short duration for each match, the number of possible outcomes will be small enough to fully explore in a reasonable amount of time. Furthermore, any given state of a tic-tac-toe or connect four game is fully observable, as no information is hidden from the players or spectators. This allows the algorithm to be fully informed when making decisions, which would be difficult in other activities such as poker, where each agent’s knowledge is limited to just their hand. Lastly, there is no randomness or probability in these games, meaning that whatever action an agent takes is guaranteed to occur exactly as they foresaw.

The algorithm used to evaluate the most optimal move for the player implements a minimax search algorithm to evaluate each action. Minimax functions by examining all possible actions a player as well as the resulting state of the game after these actions. If this state is not a terminal state, where one player has either won, tied, or lost, then the algorithm will repeat, examining the successor states of this new state. Once all possible terminal states are reached, they are evaluated based on their outcome, being assigned a positive number if the player wins, a smaller value if the player ties, and a value of zero if the player loses. The minimax agent attempts to choose a set of actions which ensures the highest possible value for the player, always choosing the highest value. However, the other player is certainly not intending to lose, and might even be playing optimally as well. For this reason, whenever it is the other player’s turn in the minimax search tree, the lowest value available will be chosen. This limits the minimax agent to acting optimally under the assumption that the other player will do so as well. If the other player chooses to subvert this expectation by playing sub-optimally, then the minimax agent will be at an even greater advantage, meaning that it shouldn’t worry about this occurring.

The first iteration of the minimax agent utilized recursion to perform the looping process of searching for every terminal state. The process begins by calling the “maxValue” function on the current game state. If the current state is a terminal state, then the function will return the evaluation of the terminal state. If not, then for every action the other player can take from this state, the function of “minValue” is called on the successor state, as it is now the other agent’s turn to minimize player1’s score. This process is looped indefinitely until all game states are evaluated, which it will then return the action necessary to obtain the highest terminal value.

This first iteration worked as intended. Out of ten tic-tac-toe games, it beat me four times, tied six times, and never lost once. When the minimax agent plays against itself, both players are playing optimally, so they will always tie. The biggest flaw of the first iteration was the time the algorithm needs to compute and evaluate its turn. In both games, the game state with the most possible outcomes needed to be evaluated is the start state. In tic-tac-toe, the minimax algorithm takes 15.6 seconds to perform its evaluation, and in connect 4, a game with many more outcomes, the algorithm never finishes, always having more states to compute. This is not desirable so the next iteration should improve on time-efficiency.

The next iteration utilized alpha beta pruning to quicken evaluation speed. This agent performs a similar search process as the minimax agent, however, when it determines that a possible outcome branch will never be chosen, then the algorithm skips evaluating that branch. For example, the algorithm might discover that when it its player2’s turn, it will have two options available: one that leads to a terminal evaluation of zero, and another that could lead to an evaluation as great as +1. Since player2 is trying to minimize the terminal evaluation, it will never choose to go down the +1 path when a lowest possible outcome path is also available. Therefore, the algorithm prunes this branch from the search frontier, as it will never occur.

This improvement to the minimax algorithm greatly improved its evaluation time, without changing the actions that the algorithm will choose. For example, the first turn of the alpha beta algorithm in tic-tac-toe only takes 0.9 seconds now, which is more than 14.7 seconds of improvement per turn. No accuracy is lost either, as the alpha beta algorithm will still choose the same actions as the minimax algorithm. However, the alpha beta algorithm never reaches a conclusion in connect 4, despite its speed improvements.

To have the algorithm function for connect 4, a cutoff number must be implemented into the alpha beta algorithm. This number dictates how many actions ahead the algorithm will search before returning an action. Before, there was no cutoff and the algorithm explored every game state, but this is unfeasible in connect 4. The implementation is simple, as now the algorithm will check if each state is either a terminal state, or if the cutoff distance has been exceeded, which it will then provide an estimation of who is winning using a heuristic function.

With alpha beta cutoff implemented, the algorithm can now play connect four at a reasonable speed. The time necessary for the first turn of connect four per cutoff is displayed below. When the cutoff length exceeds 7, the algorithm will take too long to perform its evaluation.

Chart

Description automatically generated

Although the new cutoff variable improves the speed of calculations, it results in a loss in accuracy from the algorithm, as the algorithm is now action on best estimates rather than definitive knowledge. With a cutoff less than five, I can consistently beat the algorithm in tic-tac-toe, so cutoff must be reserved for games with many outcomes such as connect four. When playing games with fewer game states needed to be evaluated, then the alpha beta algorithm without cutoff would be most optimal.

The ethicality of using minimax to win in turn-based games must be considered. If one were to secretly use a minimax algorithm in a competitive setting, they would have an unfair advantage over their competitors, diminishing the value of skill in the game at hand. As such, if an algorithm such as this were to see widespread use, it must be obvious that it is being utilized, rather than being a person playing. In addition, the question comes into play whether these simple games needed to be solved. Tic-tac-toe and connect four are games with a demographic of primarily children. Their enjoyment of the games may not come from winning, therefore they would not be interested in whether they are performing optimally. If an unbeatable algorithm such as this were to be used against individuals who are simply playing for the enjoyment of playing, then the purpose of the algorithm would be lost.