SWEN30024 Artificial Intelligence: Project Part B Report

Random_Group_17

Hoang Minh Huy Luu

Jude Offord

Our Approach:

For our game playing agent we have used an adversarial search method, specifically, a minimax algorithm with A-B pruning. To facilitate the use of this algorithm, we have created the findBestMove function. Theoretically, this function can evaluate all the future outcomes of all possible moves by using minimax(), returning the move that maximizes the value of the board for the agent while minimizing it for the opponent.

The minimax() function itself will check whether the current move is better than the best move by recursively analyzing all the possible ways the game can go, considering the potential moves of both the agent and the opponent, and returning the move that leads to the board state with the highest value. The value of a given board state is determined by the evaluate() function.

The evaluate() function makes use of Dijkstra's algorithm. It looks at each of the "end" nodes (the edges of the board, depending on the player's colour) and determines the shortest available distance from one side of the board to the other, considering a node matching the player's colour to have distance of zero. This allows us to take any given board state and determine how close a player is to winning. Incorporating the adversarial nature of our approach, we compare the scores of both players, enabling us to select the board in which the agent is closest to winning and the opponent is the furthest from winning.

As previously mentioned, we have also implemented Alpha-beta pruning. This is an optimization technique for the minimax algorithm that allows us to reduce the runtime of the program. Since evaluating every single possible future outcome can be very time consuming (particularly with larger board sizes), it is necessary to apply this pruning for the program to complete in reasonable time. If there exists a better move available, the algorithm will cut off the branches in the game tree which no longer need to be searched. The algorithm takes 2 more parameters: alpha and beta: alpha is the best value that the maximizer can guarantee at that level or above. Beta is the best value that the minimizer currently can guarantee at the level or above. In the initial call, the value of alpha is -infinity and value of beta is +infinity. The depth parameter determines how many moves into the future the minimax function will consider.

Additional Strategies:

The aforementioned minimax implementation is our main strategy for determining the best possible move in any given situation, however we have implemented some additional strategies in order to maximize the overall performance of the agent.

Within the findBestMove function, we have introduced some secondary conditions to help the agent select optimal moves not considered by the initial scoring algorithm. For example, it ensures that any move that would result in a win will always have the highest score value and thus always be selected. Also, if the board is in a state where the opponent can create a diamond and capture the agent's nodes, blocking this capture is heavily prioritized. Captures can put the agent in a very disadvantageous situation very suddenly, so we made sure to minimize how often they occur.

The findBestMove function does not incorporate additional value to the agent itself making captures, as we felt this could lead to problems with the agent becoming

distracted from its main goal of connecting the edges of the board. Instead, we have implemented a strategy where the agent will only prioritize captures if it is currently losing. A losing situation can be very difficult to come back from, especially when facing another intelligent agent. Thus, if the agent is losing, it will begin to prioritize captures in an attempt to make a comeback.

One final small strategy we implemented involves the "steal" rule. Simply put, if the agent is assigned to the blue colour (going second), it will always opt to steal red's move. We can mostly expect that an intelligent opponent will always pick an advantageous or neutral first placement. However, even if they were to select an unfavourable starting point, we found in our testing that the advantage of going first (which blue gets to guarantee through the steal action) is far too critical to ignore.

Overall Effectiveness and Limitations:

To test the effectiveness of the algorithm, we created a few other agents to compete against it. A random agent that selects its placements randomly and a rush agent that instantly takes the first available placement. Against both agents our adversarial search method in combination with our extra strategies is able to consistently win every time on all board sizes. When testing our agent against itself, we saw that both players very effectively try to move towards their goal while also attempting to hinder the progress of the other. In almost all cases the player that goes first will win, in the case of our agent this would be the blue player making use of the "steal" rule. However, on a few smaller board sizes certain captures became available that allowed red to comeback from its usually losing situation. This shows our adversarial agent to be quite intelligent, even capable of overcoming the first-mover advantage in some scenarios.

Although our agent can effectively run most of the time, there are still some limitations. Our evaluation function works well when using a low cutoff depth for

pruning, but a higher depth would bring far better performance. However, if the agent is initialized with a high depth, the program will exceed the time limit of n^2 (where n is the board size) due to the higher branching factor. On large boards with high depth the program's run time became too long to consider. Because of this, we had to make decisions sacrificing better evaluation for shorter run time, in order to keep a reasonable balance.

Conclusion:

Overall, we found our final version of the Minimax with Alpha-beta algorithm to be quite an efficient agent for playing Cachex. The process of developing this strategy was based on our observations from multiple tests from our earlier versions and against different kinds of bot agents. Although this algorithm is not perfect in some cases, it manages to maintain a good balance of all the factors that need to be considered: space complexity, time complexity and effectiveness.