Report

Introduction

In the field of artificial intelligence, zero-sum games are a category of games where one player's gain is exactly balanced by another player's loss. The sum of utility values for both players is 0. In this game, only one player can win after eating the another's all pieces, and another player will also lose as a result, which is in line with the characteristics of zero sum games. This report focuses on the development of an artificial intelligent agent for zero sum games: "infection", which is carried out by two players, red and blue, in turn, with the goal of eliminating all the pieces of the opponent to win. The information on the chessboard is known, and the opponent's next move can also be observed. So, a utility-based agent can be adopted to play this game. The report contains a brief overview of the game's rules and implementation, an explanation of why we choose the Minimax tree algorithm and the Alpha-Beta pruning techniques which are used in my project, a discussion of the utility function, an examination of optimization techniques, an evaluation of the agent's performance, and finally, a conclusion with future directions.

Game Rules and Implementation

Infexion is a complete information game played by two players, with the goal of controlling all pieces on a $7 * 7$ hexagonal chessboard. The position on the chessboard is represented using an axis coordinate system. There are two players in the game (red and blue) who take turns to operate. In the game, chess pieces can have 1 to 6 different weights (called POWER).

At the beginning of the game, the board is empty. The red side takes the lead, and then the two take turns doing one of the following:

SPAWN: Generate a chess piece with a weight of 1 on an empty grid, provided that the total weight of all grids on the chessboard is less than 49.

SPREAD: The player selects a controlled piece stack and spreads its weight of k pieces along a hexagonal direction onto adjacent k squares. The weight of each target grid is increased by 1. If the target grid originally belonged to an opponent, then that grid will be occupied. If the weight of a grid after dispersion exceeds 6, the grid becomes a space.

In this game, eating the opponent's pieces is in most cases more valuable than generating a new piece, because you can use the SPREAD operation to eat multiple power points in one turn, and the opponent will also lose multiple power points as a result, while SPAWN can only increase the number of your pieces by 1. Therefore, we cannot simply use obtaining more pieces as our game goal to guide us towards victory. Using the power difference between both sides as an evaluation function may be a good choice, as it will make the SPREAD action score higher than SPAWN, thus better guiding players to win

why we choose to use minimax tree algorithm：

- The Minimax search algorithm performs well in two person zero sum games because it can effectively evaluate and select the best strategy. Infexion games have the following characteristics, making the minimax search algorithm very applicable:
- Complete information game: Infexion is a complete information game, which means that at any time, both players can see the complete game status. In this environment, minimax search can predict the optimal strategy for each step by traversing possible game state trees.
- Two player zero sum game: In the Infexion game, one player's loss equals the other player's gain. This means that minimax search can minimize opponents' profits by maximizing its own profits, thereby finding the optimal strategy.
- Certainty: The actions and outcomes in Infexion games are deterministic, meaning that given a game state and the player's actions, the next game state is deterministic. This enables the minimax search algorithm to accurately predict game state trees.
- Limited search space: The Infexion game is played on a limited 7x7 chessboard, with an upper limit on the weight of each grid. This makes the search space limited and controllable, and the minimax search algorithm can find the optimal strategy in a reasonable time.

However, in the rule of Infexion game, a player only has a total turn time of 180 seconds and 250mb of memory space. The minimax search algorithm may face the problem of large search space and search time. Therefore, we can improve our algorithm by adopting alpha beta pruning and topk optimization. The former can prune low value actions that are not very helpful for decision-making, while the latter can limit the number of nodes we can expand each time to solve the problem of large search space and reduce the search time cost.

Minimax Algorithm and Alpha-Beta Pruning

The Minimax algorithm is a decision-making algorithm used in two-player, zero-sum games. It works by evaluating the game tree to determine the optimal move for the current player, assuming that the opponent is also playing optimally. In our implementation, we used the Alpha-Beta pruning technique to optimize the Minimax algorithm, which significantly reduces the number of nodes that need to be evaluated in the game tree.

Alpha-Beta pruning works by maintaining two values, alpha and beta, that represent the best possible score the current player can achieve and the worst possible score the opponent can achieve, respectively. During the search, if the current player's best possible score is greater than or equal to the opponent's worst possible score, the search is pruned, as it's unnecessary to explore further nodes. The default values for alpha and beta should be set to minus infinity and positive infinity, and the range of values should be continuously reduced as nodes update.

In our implementation, the Minimax algorithm with Alpha-Beta pruning is applied in the `alpha_beta_minimax_tree` function. The `alpha_beta_prune` function is recursively called to traverse the game tree, and the utility value of each game state is evaluated using the `new_evaluate_board` function. Each turn, we will get a list of action from the current board state, then, We use the total power of our chess pieces to subtract the total power of enemy chess pieces as our utility function, and we apply the resulting utility value to each corresponding action in the action list to get an action dictionary. Consider the utility value of the chessboard after executing this action. We use the values of the dictionary as the basis for sorting, sorting all feasible actions on the current chessboard from largest to smallest. Then in `alpha_beta_minimax_tree` function and `alpha_beta_prune` function, we using for loops to spread the nodes by apply these actions to the current board.

Advantages of using the Minimax algorithm with Alpha-Beta pruning include its ability to find the optimal move in a significantly reduced amount of time compared to a full-depth search:

- In Traditional Minimax algorithm:
  - Time complexity: For a complete game tree, the traditional Minimax algorithm has a time complexity of $O(b^d)$, where b is the branching factor (average number of sub nodes per node) and d is the depth of the tree (number of search layers). In Infexion games, the branching factor may be large, as each player can generate pieces on different empty squares or scatter existing pieces in different directions.
  - Space complexity: The space complexity of the Minimax algorithm is $O(b * d)$, as it requires storing each layer of nodes in the search tree. On each layer, there can be up to b child nodes, so a total of bd nodes need to be stored.
- In Minimax algorithm with Alpha Beta pruning:
  - Time complexity: Alpha Beta pruning reduces search time by pruning unnecessary parts of the search tree. In the worst-case scenario, the time complexity of Alpha Beta pruning remains $O(b^d)$, but in the best case scenario, its time complexity decreases to $O(b^(d/2))$. In practical applications, Alpha Beta pruning typically significantly reduces search time.
  - Space complexity: Compared with traditional Minimax algorithms, Minimax algorithms with Alpha Beta pruning have the same space complexity: $O(b * d)$. Although Alpha Beta pruning reduces search time, it still requires storing node information for each layer.

Therefore, Minimax algorithms with Alpha Beta pruning have advantages in terms of time complexity compared to traditional Minimax algorithms. Although their space complexities are the same, the alpha beta pruning can significantly reduce search time as it cut off abundant useless nodes, thereby finding better solutions in limited time.

Utility Function

The utility function, as implemented in the `new_evaluate_board` function, calculates the game state's value by subtracting the total power of the opponent's pieces from the total power of the current player's pieces.

$f = SelfTotalPower - OpponentTotalPower$

This simple heuristic encourages the agent to make moves that maximize its power while minimizing the opponent's power. The utility function plays a crucial role in guiding the agent's search through the game tree, helping it identify promising moves and avoid unfavorable ones. What's more, We have assigned higher utility values to actions that can lead to victory, in order to better guide agent in selecting actions that can end the game (in our code, positive infinity is used as the utility value for winning the game, and negative infinity is used as the value for losing the game)

Optimization Techniques

To further enhance the performance of the Minimax algorithm with Alpha-Beta pruning, we implemented two optimization techniques: Top-K pruning and transposition table.

Top-K pruning involves truncating the action list by only considering the top K actions, reducing the branching factor from b to k. This significantly decreases the time complexity of the algorithm, which becomes $O(k^d)$ instead of $O(b^d)$. Topk has played a very significant role in our search algorithm, which shocked us greatly!

Additionaly, Transposition tables use a hash value to represent the game state, allowing the agent to quickly look up previously evaluated game states and avoid redundant calculations. This should be able to speeds up the search process and reduces the overall computational cost.

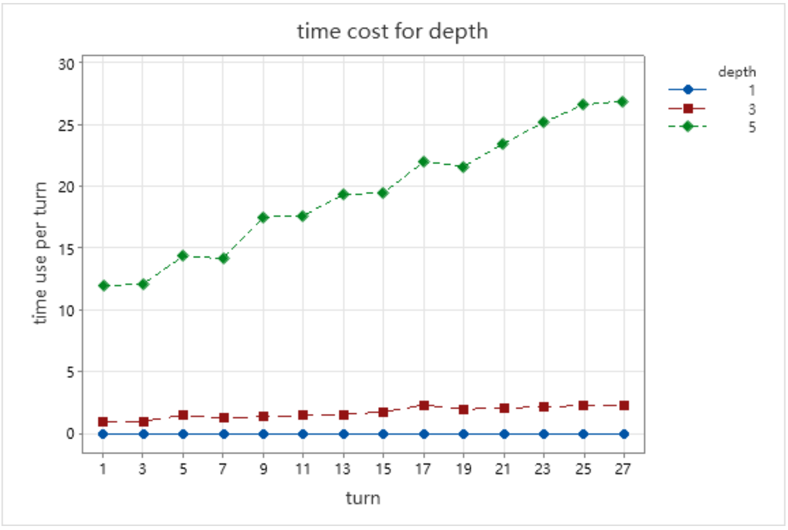
However, in our actual testing (in the old function: `minimax_tree_search`), we found that permutation tables did not play a significant role in accelerating search. This may be because for the game Infexion, the changes on the chessboard are relatively rich, with very few identical chessboards appearing. But with the increase of turns, the permutation table itself also needs to occupy memory, which led us to abandon using permutation tables to optimize search in the end.

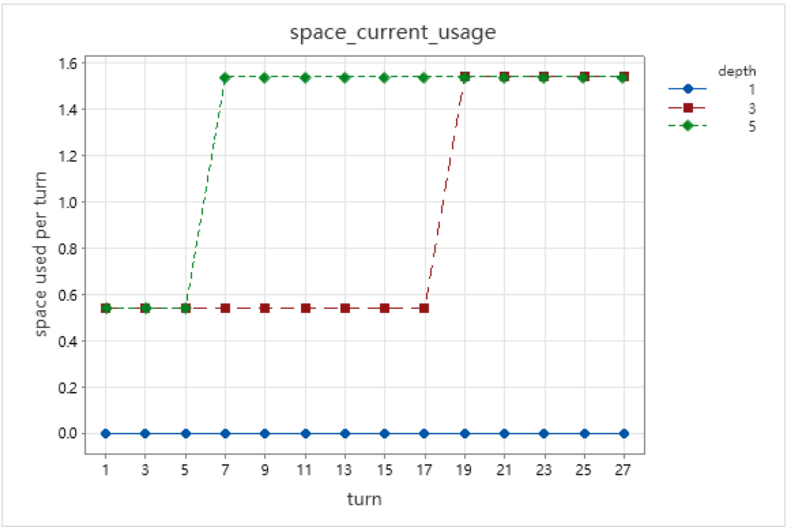
In the next section, we conducted quantitative testing on these two optimization techniques to observe their effectiveness

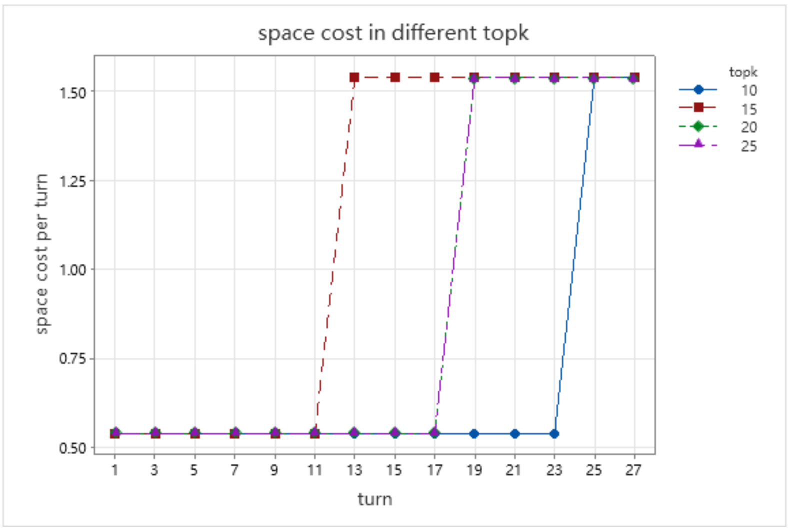
Evaluation and Analysis

We evaluate the performance of agents by comparing them with different algorithms and configurations. This agent did not perform well in the Monte Carlo Tree Search (MCTS) algorithm, even though MCTS was allowed to perform up to 1000 iterations. Continuing to increase the number of iterations of the Monte Carlo algorithm will consume a significant amount of memory and time, while the Minimax agent achieves comparable performance with lower time and space complexity, making it a more effective choice for this specific zero sum game. For example, the use of Top-K pruning effectively reduces the branching factor and significantly reduces the time complexity of the algorithm. In addition, the best case time complexity for Alpha Beta pruning with perfect sorting is $O(b^{(d/2)})$, which further improves the efficiency of the algorithm.

- Evaluation of the effectiveness of the current agent:
    - Firstly, we used a random action agent as our first opponent to test the algorithm correctness of the agent. After observing the rationality of each round of action (our agent can always make a step closer to victory), we verified the correctness of the algorithm.
    - Then, we changed the topk and minimax depth parameters to compete with each other, and by counting the winning rates of different combinations. We also arranged and combined different topks and minimax depths. By comparing the printing time and memory, we finally obtained a relatively reasonable threshold and applied the parameters to the final agent, Below is a graph which show the time usage and space usage for first 10 turns:







- Analysis of possible defects in the current agent:
    - The topk algorithm is a trade off of efficiency and performance, which may cause our agent to miss some actions that have low returns in the current round but significant returns in subsequent rounds when selecting the optimal solution. When these actions are trimmed by the topk due to the low returns they can bring in the current round, we may miss the optimal solution. Overall, although topk improves algorithm execution speed to some extent, it can make our agents short-sighted and lack foresight. In addition, selecting the first k actions requires time to sort the action dicts, it will cost $O(b \log b)$ to do that.
    - As mentioned above, alpha beta pruning cannot effectively reduce the number of nodes in certain situations, and its spatial complexity is the same compared to traditional minimax trees. In extreme cases, alpha beta pruning has almost no effect

Conclusion and Future Directions

In summary, this report discussed the development and evaluation of an AI agent for a zero-sum game using the Minimax algorithm with Alpha-Beta pruning and additional optimization techniques. The agent performed well against other algorithms, such as the MCTS, while demonstrating lower time and space complexity.
However, there are limitations to the current implementation, including the reliance on online search, which can be computationally expensive in terms of time and space complexity. Future work could explore the use of offline search techniques, such as reinforcement learning-based Decision Transform algorithms, which have been employed in systems like AlphaGo Zero. These algorithms can learn and improve over time, potentially leading to even better performance while maintaining lower computational costs.

Additionally, further refinement of the utility function and the incorporation of more advanced heuristics could enhance the agent's performance in more complex game scenarios. Overall, this project demonstrates the effectiveness of the Minimax algorithm with Alpha-Beta pruning in zero-sum games and highlights areas for future exploration and improvement.