5002 Final Project individual report

ZHOU Yang

1. Understanding of MCT

The basic principle of MCT is to simulate the subsequent situations of a certain situation in large quantities, and use the statistical results as the estimated score of the situation to provide a basis for AI's decision-making at the previous level in the situation. Taking this Gomuku project as an example, we have two ways to use MCT: one is to traverse until a win or loss is determined, and return the value (win or lose) to the root node for statistical processing. The other is to traverse until the specified depth, then give the situation a score, and return this score to the root node for statistical processing. In actual operation, due to the judgment time limit, we adopted the method of specifying the depth, but expanded the number of simulations.

2. MCT application in our project

The way we evaluate the situation is: win +10, draw +1, lose -10. When deciding the next step, we traverse all possible positions, perform an MCT search for each position, and choose the point with the highest score to place the piece. However, a big problem here is the search time. The solution given by our group is divided into two parts:

First, optimize the storage method of chess game information. The usual idea is to store chess game information as a two-dimensional array, but this means that we often need to traverse the array when judging the situation. Therefore, our group's chess game storage method is to use a four-dimensional array (credit to our groupmate ZHANG Jinqian), and each chess piece contains a record of its current coordinates (two-dimensional) and the placement of adjacent positions (one-dimensional) in different directions (one-dimensional). When judging the situation, we only need to determine whether the four-dimensional array of the position of the chess piece generates win-loss information each time we place the chess piece, without traversing the positions around it, which greatly increases the running speed.

The second is to narrow the search range (credit to our groupmate LI Bowen). We found that not every point on the chessboard is worth searching, so we use the four-dimensional matrix information of each point (mainly the last two dimensions that record the placement of nearby positions in four directions) to evaluate the search value of the point, and only search for points with a value higher than (highest value + average value)/2. Details of this part can be found in Readme and our program.

3. Other approaches to improve the performance

In addition to the above two improvements, we also tried to combine other methods with MCT to improve the performance of AI. One way is to introduce reinforcement learning (credit to our groupmate LIU Liangjie), in order to obtain the evaluation value of each position by weighting the four-dimensional array, and use this to decide whether to perform MCT search at that position. There is no doubt that if this method is successful, it will optimize our MCT search range, but because this part requires a lot of time to train, we ultimately cannot get better results, so this part was not added in the final submitted program.

Another way is the minmax algorithm. The minmax algorithm is a common algorithm in chess games. Theoretically, if we use the minmax algorithm to expand to the leaf node to generate the win or loss, we can get the optimal strategy. Our initial idea was to narrow the search range through the recommended search points in the previous step, that is, each move generates a recommended range, traverses this recommended range to generate the next step, and finally perform MCT search at the leaf node to generate the score. However, even if the search range has been greatly narrowed, the time required is still exponentially increasing as the number of steps increases (the implicit assumption here is that the size of the search range is stable. And from the method we used to generate the search range, this assumption is reasonable). In order to reduce the time, we used the pruning algorithm, which is also an algorithm that often appears with minmax. However, in practice, the pruning algorithm does not help us much. We thought about sorting the search range to maximize the effect of pruning, but we did not put it into practice in the end. Because in actual testing, we found that at a given depth, the results obtained by spending time doing minmax did not seem to be as good as spending time doing a larger number of MCTs. We believe that if minmax is to perform well, it needs a larger search depth. However, the amount of computation required will far exceed the limit, and even pruning will not help much. Therefore, we did not include minmax in the final result.

In summary, we finally determined that pure MCT is the most perfect answer we can give. This is actually close to verifying an implicit assumption of using the MCT strategy in Gobang. Previously, I thought that many scenarios in MCT might not be achieved (because our MCT is a random placement, that is, in theory there is a possibility of connecting five pieces without any hindrance, but this is impossible to achieve in actual battles because the opponent will definitely prevent it), so I doubt whether there is a high positive correlation between high MCT scores and the correctness of placement. But this time the result made me realize that when the number of simulations is sufficient, a piece that can produce more winning situations actually tends to play a greater role in the current situation.

4. Peer Evaluation

In this project, I am very grateful for the efforts of each group member and enjoy the time spent discussing with everyone. I give the highest score to each groupmate.