Chess Evaluation Functions

The AI mainly relies on the "search" algorithm to find the optimal move. Every time a leaf node is searched, we need to use an evaluation function to quantify the advantages of each move for us, so as to choose among many possible moves. A move that is optimal for oneself, so the evaluation function directly determines the strategy of the AI playing, which is the key to determining the intelligence of the AI.

Through the study of the rules of chess and common tactics, we mainly added three aspects: the evaluation of sub-power, the evaluation of mobility, and the defiance factor in the project.

Basic evaluation functions

1. Material

A term determined by the sum of piece values of each side. The material value is the most influential part of the evaluation. Usually it is the sum of a constant value for each piece present, which allows positional features of the position, worth less than a single pawn, to be evaluated without requiring fractions but a fixed point score.

2. Piece square table

A simple way to assign values to specific pieces on specific squares. A table is created for each piece of each color, and values assigned to each square. This scheme is fast, since the evaluation term from the piece square tables can be incrementally updated as moves are made and unmade in the search tree. Because of that speed, piece-square tables are of great help when conducting lazy evaluation.

3. Mobility

Mobility reflects the number and importance of all possible moves of all pieces of a party. The calculation method is to count the number of feasible moves of each piece of the party to be evaluated, multiply the value of its moves, and then accumulate the calculated values of all pieces.

One of the hallmarks of checkmate is that there are no logical moves. Therefore, we add mobility evaluation, so that when choosing moves, the AI is inclined to move plans that can make more moves . However, considering trying to limit the opponent's mobility at all costs would have the program conduct a series of aimless generals while destroying its own lines of defense. Therefore, it is necessary to strictly control its proportion in the evaluation system.

4. Center control

A chess strategy to control or occupy the center and extended center of the board, which gains space and allows pieces fast access to most of the board areas, and more movement and possibilities for attack and defense.

5. King Safety

Good evaluation of the king's safety is one of the most challenging tasks in writing an evaluation function, but also the most rewarding. The subjects are placed here in order of approximate (implementation) difficulty. If this page grows, it might be worthwhile to create a sub-page for each term.

6. Sub-force assessment

The evaluation of the value of the piece force refers to considering the type and position of each piece on one side of the chessboard, determining its evaluation value according to the importance of the type of piece and the pros and cons of its position, and then calculating the evaluation value of all the pieces on the chessboard. accumulate directly. This kind of method only considers the attributes of each piece itself, and does not care about the situation around the piece, and has the largest weight in the entire evaluation system .

The value of fixed piece force refers to the existence value of the piece itself, reflecting the comprehensive piece force of a piece of chess. Set a special value to the king, infinity, to show its inescapable importance. The piece position value reflects the attack ability of the piece at a specific position on the board. Applying different reward or penalty values for the same piece in different positions on the board can make the game move towards a favorable position.

The processing of the piece force evaluation part of this AI is mainly in the keyboardmujahid.h. In order to save the resources for calculating the evaluation value, we store the current score in the dictionary board_info, and the evaluation of the chess surface after subsequent moves can be performed directly on the basis of the previous chess surface. Add-subtract operations , Effectively improve the speed of evaluation

Our evaluation functions:

We are using 3 evaluation functions for the different 3 states as We need to make a system that interacts with the user as a human. We are using the following evaluation function for this purpose.

1. **Starting evaluation function**

   We are using this function for the first 3 moves that are based on the **Material method**. We are just adding the values of the pieces that are present on the board and conclude based on the final sum of the board values.

```
int startEvaluation(gameState g) {
    int score = 0;
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
        {

            score += g.Board.board[i][j] * 3;
        }
    return score;
}
```

2. **Middle state evaluation function**

Using this function when we have more than 3 moves and less than 7 it works based on mobility we are setting the **mobility** of each move in it and adding that mobility in a variable at the end of this process this function will return the sum of all the mobility that are present in the form of pieces

```
int midEvaluation(gameState g) {
    int score = 0;
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
        {
            if (g.Board.board[i][j] == 4 || g.Board.board[i][j] == -4) {// cod
                score += 5;
            }
            else if (g.Board.board[i][j] == 2 || g.Board.board[i][j] == -2) {/
                score += 3;
            }
            else if (g.Board.board[i][j] == 3 || g.Board.board[i][j] == -3) {/
                score += 3;
            }
            else if (g.Board.board[i][j] == 5 || g.Board.board[i][j] == -5) {/
                score += 9;
            }
            else if (g.Board.board[i][j] == 6 || g.Board.board[i][j] == -6) {/
                score += 100;
            }
            else if (g.Board.board[i][j] == 1 || g.Board.board[i][j] == -1) {/
                score += 1;
            }

        }
    return score;
}
```

3. **Advance state function**

When we have more than seven moves. It decides the weight of each move with the **king's attack strategies**. if our king under-thread try to protect it, Plus it also uses **Mobility strategies**.

```
int advanceEvaluation(gameState g) {
    int score = midEvaluation(g);
    if (g.getPlayer() != playerColor) {
        if (g.kingUnderThreat(g.getPlayer())) { // if just king underThread
            score += 50;
        }
        if (g.Actions.getActionCount() == 0 && g.kingUnderThreat(g.getPlayer())) {
            score += 100;
        }
    }
    return score;
}
```