

Design Manual

1. Introduction.....	2
1.1 Purpose of this document.....	2
1.2 Project background and goals.....	2
1.3 Definition.....	3
1.4 References.....	3
2. Design.....	3
2.1 Requirement overview.....	3
2.2 Schematic diagram of program structure.....	3
3. Program Description.....	4
3.1 Function.....	4
3.2 AI workflow diagram.....	7
3.3 Interface.....	11

1. Introduction

1.1 Purpose of This Document

This document provides game players and AI vs player chess software developers with a better understanding of the overall design of software development, algorithm design principles, and helps for secondary developers.

1.2 Project Background and Goals

The goal of this project is to develop an AI system that can automatically evaluate the current state of a chess game and provide the best move according to its analysis. The system will be presented through a graphical interface that displays the chessboard and the current state of the game as it progresses through human and AI moves.

It mainly has the following functions:

- Graphical interface display (showing window controls for user interaction, displaying the chessboard and chess pieces)
- AI evaluation settings (search depth, setting values for the relative strength of different chess pieces)
- Game player selection of game mode (both sides controlled by AI, one side controlled by AI and switchable, both sides manually controlled by players)
- Game player initiation and progression of gameplay
- AI automatic evaluation and completion of human-machine gameplay
- Software checks for compliance with game rules in move decisions
- Software detects game end conditions (checkmate, stalemate, etc., or if more than 50 moves have been played without a capture)
- Software timing move decisions.

1.3 Definition

- Min-max search: the minimax algorithm
- Pruning search: alpha-beta pruning
- C#: The project is coded in the C# programming language.
- WinForms: The graphical interface uses the Windows Forms API.
- Visual Studio 2017: The development environment, with a framework of .NET Framework 4.6.

1.4 References

Minmax Algorithm in game theory

<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>

Alpha-Beta Pruning

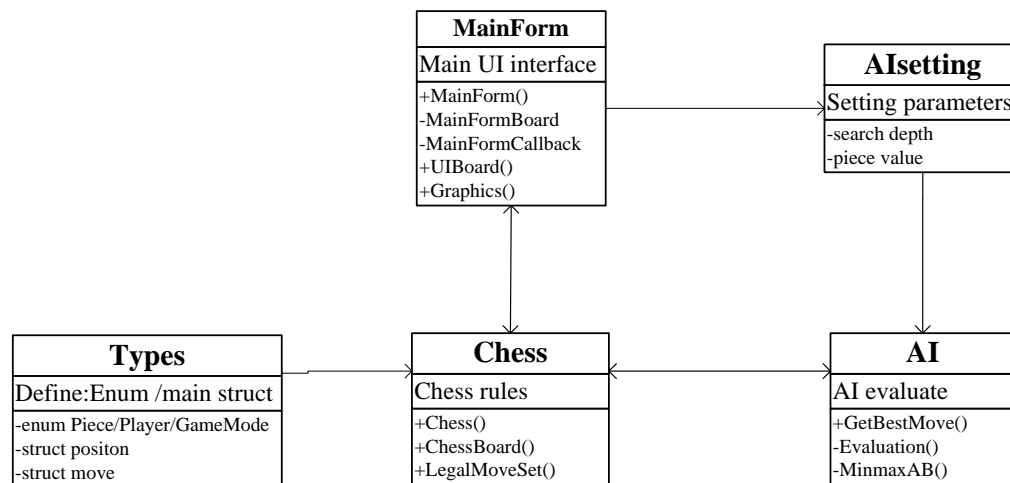
<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/>

2. Design

2.1 Requirements overview

1. Construct a model and related methods for chess movement rules based on the rules of international chess;
2. Implement AI that automatically responds to player moves to form a game;
3. Implement a graphical interface to display the game progress and board state for both players;
4. Provide prompts for capturing pieces or putting the opponent in check.
5. Record the game history log for both players.

2.2 Schematic diagram of program structure



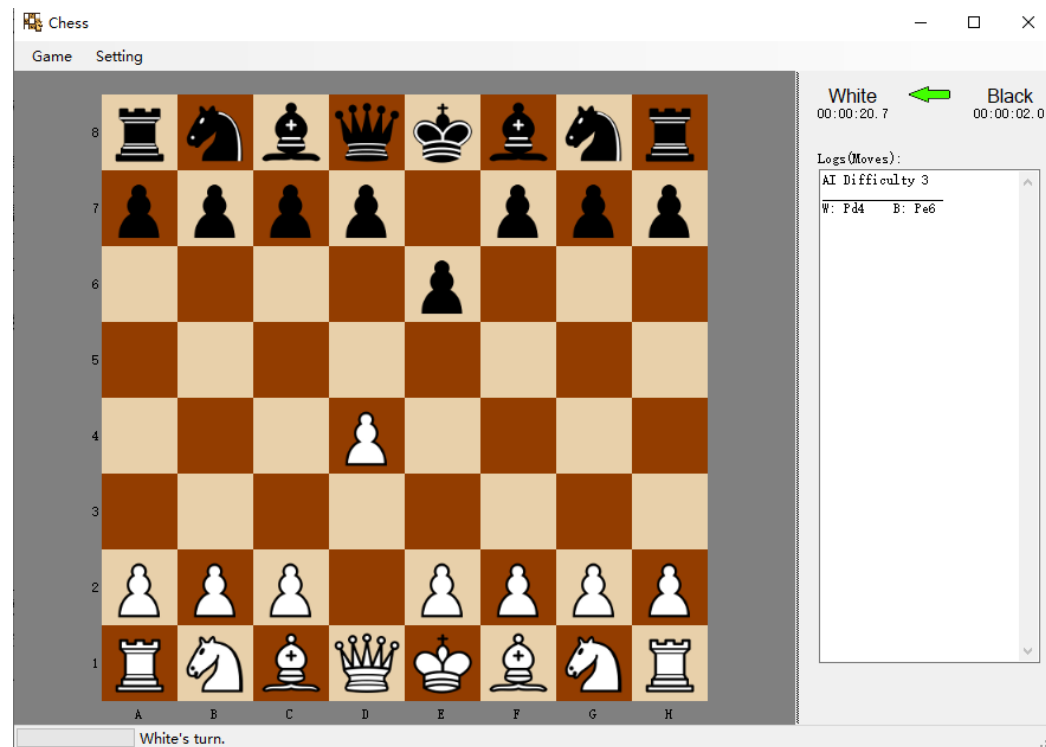
- Chess is a class that controls the logical operation of the game and forms the core of the software. It has two independent auxiliary classes: ChessBoard and LegalMoveSet. ChessBoard mainly records the current state of the chessboard in various forms. LegalMoveSets is a chess rule class used to determine whether a move is legal and whether the game is over. It can also form the action set of each chess piece under the current state.
- MainForm is the form that runs the game and serves as the UI. It has two shadow forms, MainFormBoard and MainFormCallbacks, to ensure smoother interface operation.
- AI is an AI that evaluates values and uses a minimax search with alpha-beta pruning to make the best move decisions.
- Types define various enumeration values and structures required by the system, such as chess piece enumeration, player enumeration, game mode enumeration, chess piece position structure, and chess piece movement model structure.
- AIsetting allows users to set the game difficulty (search depth) and chess piece strength values through the FrmSetting form interface to affect the AI evaluation results.

3. Program description

3.1 Functions

3.1.1 Graphical display of chessboard and chess pieces

Feature Description: The main interface displays an 8x8 square international chessboard in a graphical manner, with chess pieces displayed on the chessboard. When the participant or organizer starts or resets the game, the function of initializing the display of the chessboard should be achieved. After the game starts, the chessboard displays the game position and is constantly updated as both sides make moves.

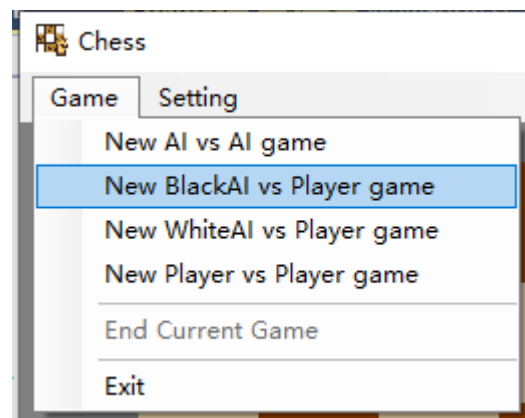


3.1.2 Playing games in different modes

The following modes can be used for playing games::

- A) AI vs AI;
- B) Black AI vs Player
- C) White AI vs Player
- D) Player vs Player

Where "Player" refers to a human player using the mouse to click on a chess piece and select its move location to complete a move, and "AI" refers to this software directly reading information from the AI to complete a move without user confirmation. Before starting a game, players should be able to create one of these modes from this software.



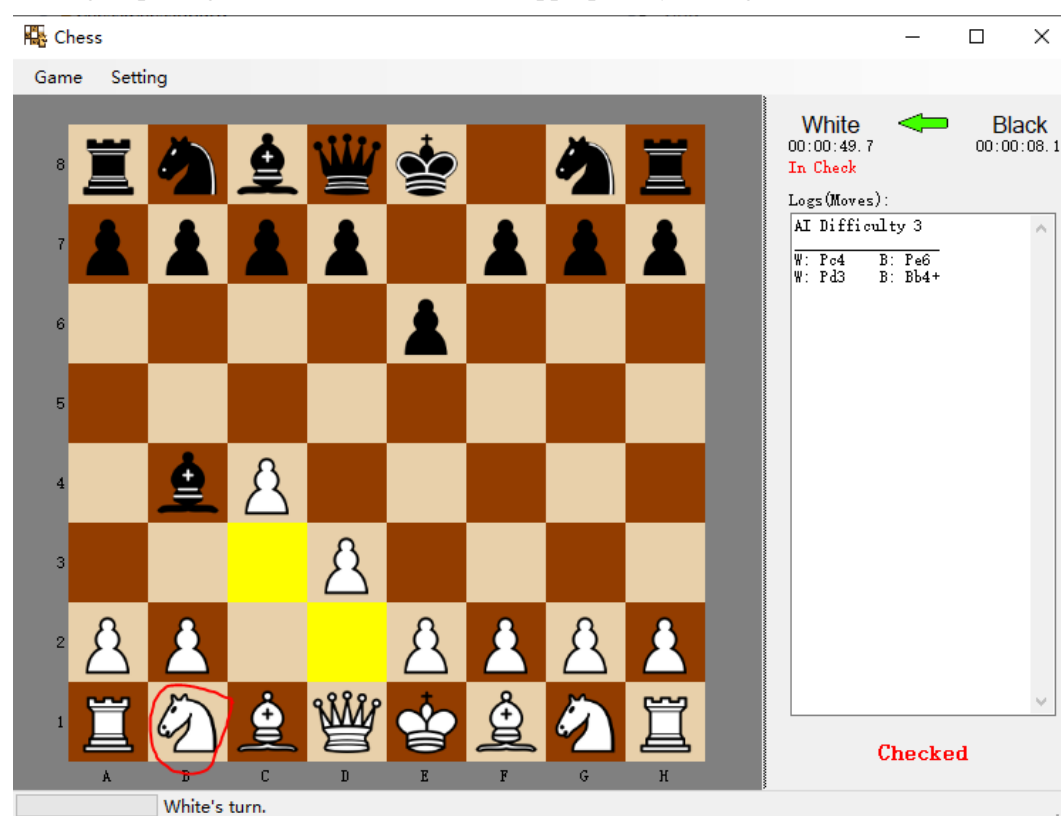
3.1.3 AI Setting

The AI can be configured with parameters such as difficulty level and piece values to

accommodate players of different skill levels.

- **Difficulty setting:** Difficulty is divided into beginner, easy, medium, difficult, very difficult levels, which correspond to the analysis depth of AI. Beginner, easy, medium, difficult, and very difficult correspond to analysis depth levels of 1 to 5, respectively. When the analysis depth is 3, the AI response time is about 5-6 seconds; when the analysis depth is 4, the AI response time is about 20 seconds; when the analysis depth is 5, the AI response time is about 2-5 minutes (or even longer depending on the board state).
- **Strength value:** The setting of strength values for each chess piece is to better evaluate the optimal move position and can be set from 1 to 999. After the settings are completed, they are only saved in the memory of the current software session and have not been saved to the local device. If you want to save them, you can handle it appropriately.

3.1.4 Human-computer interaction: handling user interactions, enabling automatic and manual gaming, and determining whether moves comply with the rules. When the user clicks on their piece, the system displays all possible move positions by highlighting the squares with different colors. All positions where the player can capture a piece are highlighted in red, while positions where the player can move without capturing are highlighted in yellow. The sound effects for moving, capturing, and check have also been appropriately distinguished.



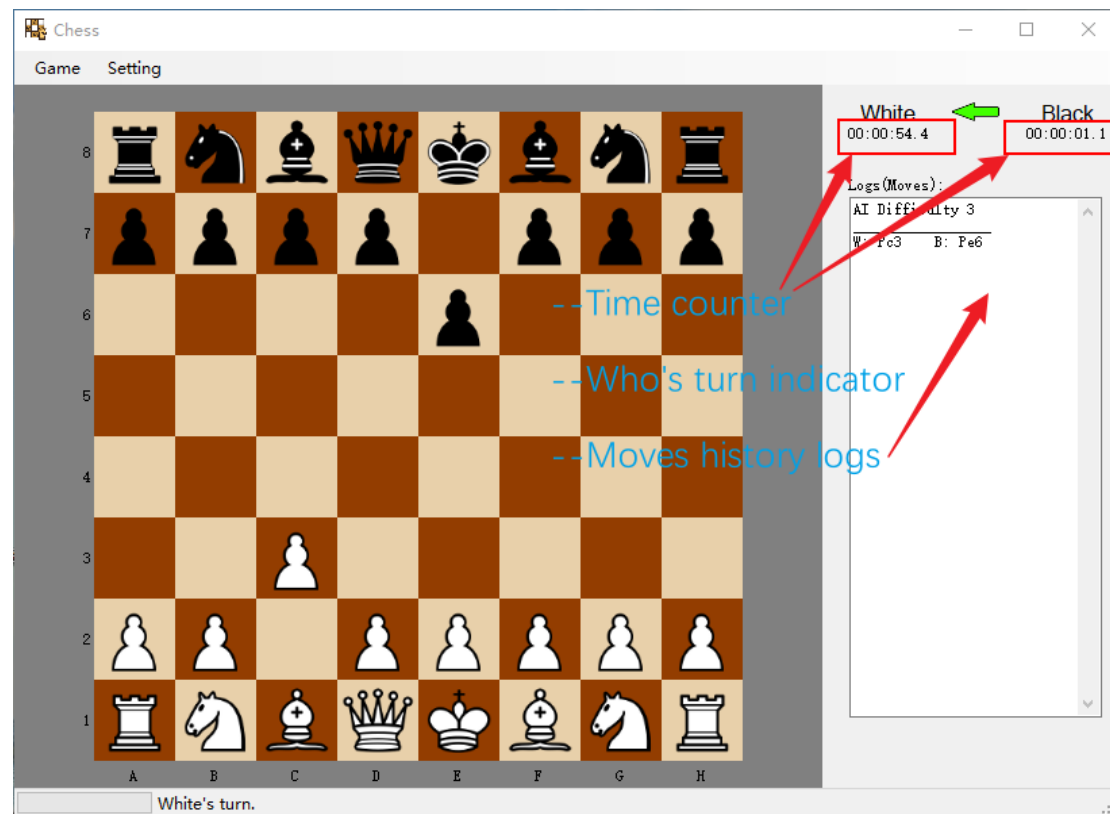
3.1.5 Game timing

To ensure that the AI's thinking time is controlled, the system separately tracks the time used by each player during their turn. When it is the black player's turn, the waiting time before making a move is accumulated and recorded as the time used by the black player. During the AI's thinking period, a progress bar in the lower-left

corner of the interface displays "AI thinking..." to indicate that the AI is computing its move.

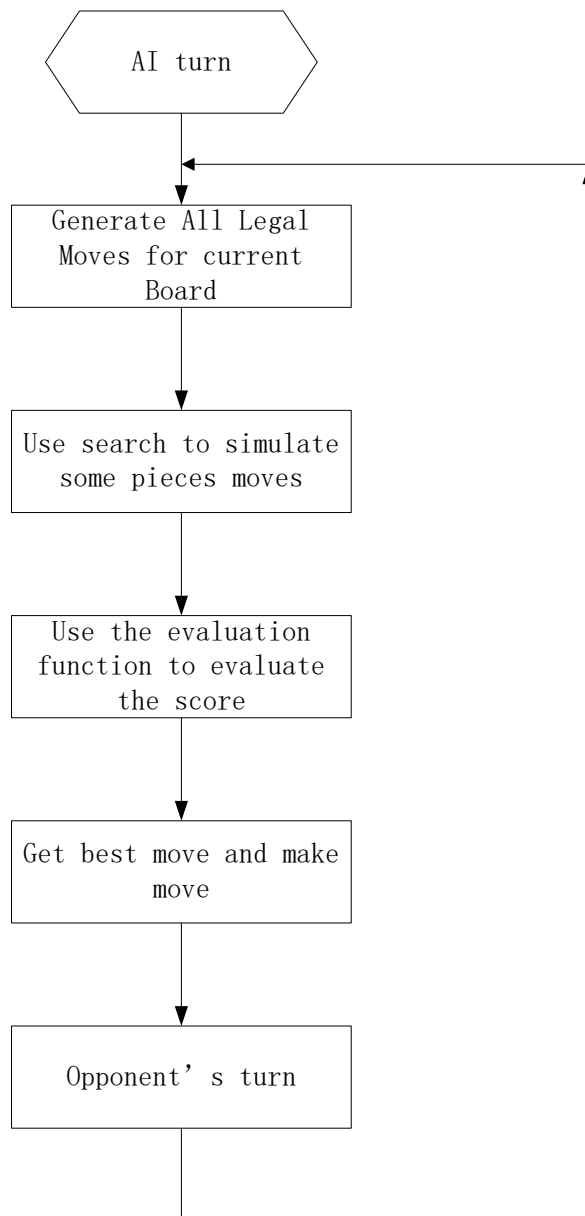
3.1.6 History record and current turn indicator:

- **Current turn:** The lower left corner will display whether it is White or Black's turn to play. At the same time, the green arrow in the upper right corner will point to White or Black accordingly.
- **Check:** When one player is in check, it will display "In Check" below the timer and "Checked" in the lower right corner.
- **History record:** The "Log (Moves)" box will record the abbreviated moves of both White and Black. The record starts with "W:" for White and "B:" for Black, followed by the abbreviation of the piece moved. P for Pawn, N for Knight, B for Bishop, R for Rook, Q for Queen, and K for King. Then, the destination coordinate is listed, such as "Pc4" for Pawn moving to c4. If the move puts the opponent's King in check, it is marked with a "+". If a piece is captured, it is marked with an "x".



3.2 AI Workflow Diagram:

According to the rules of chess, the basic workflow of determining AI is as follows:



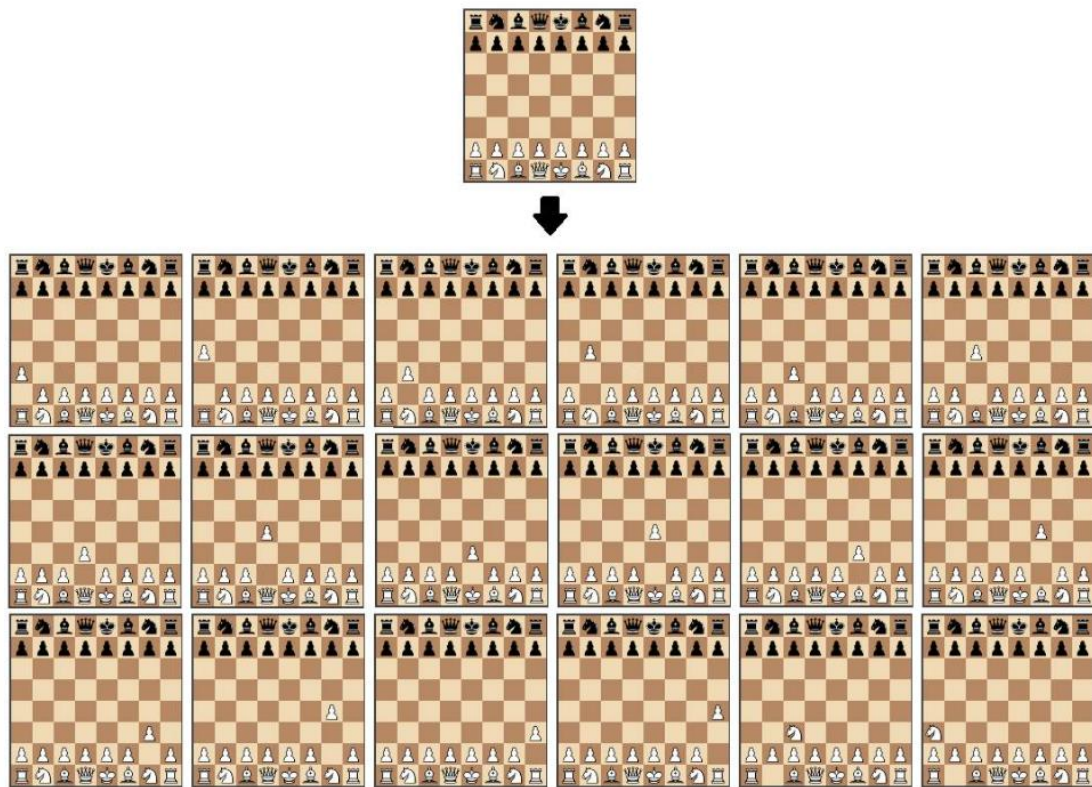
By utilizing the `LegalMoveSet.getPlayerMoves` function to obtain a dictionary of all legal moves for our side, the program conducts a search based on evaluation to ultimately determine the optimal move for our side. The search and evaluation functions work together to enhance the AI's "thinking" ability from two aspects::

- Evaluation function: directly impacts the AI's choices by evaluating the quality of the game state.
- Search method: accelerates the search speed and reduces the time required for deep searches, providing the potential to deepen the search depth and improve the AI's long-term and comprehensive "thinking" abilities.

STEP1: Iterate to get all possible walking methods.


In this step, the `LegalMoveSet.getPlayerMoves` library will be used to iterate through moves and calculate all possible legal moves for a specific chess

position.



STEP2: Simulated moves and position evaluation

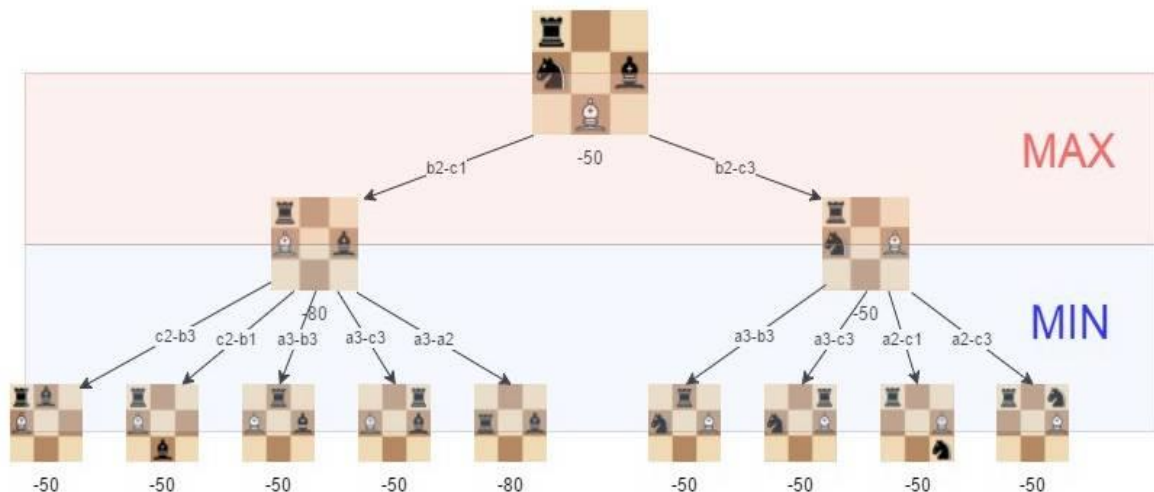
This section aims to determine which side has the advantage in a given chess position. The simplest method is to calculate the relative piece value of each chess piece on the board using a table. Through the evaluation function, we can obtain the move that maximizes the calculated evaluation value. However, in this case, all possible moves need to be calculated, which requires high computational power and

	10		-10
	30		-30
	30		-30
	50		-50
	90		-90
	900		-900

is time-consuming.

STEP3 **Search tree using the minimax method**

The algorithm constructs a search tree to choose the best move, using the minimax algorithm, which is widely used in decision theory, game theory, and statistics. Its basic principle is to minimize the loss under the worst possible scenario (i.e., the maximum possible loss). This algorithm searches all possible paths of the recursive tree based on a given depth and gives an evaluation at the "leaf" node. After that, the algorithm returns the minimum and maximum values from the child nodes to the parent node depending on whether it is the turn of the white or black player.



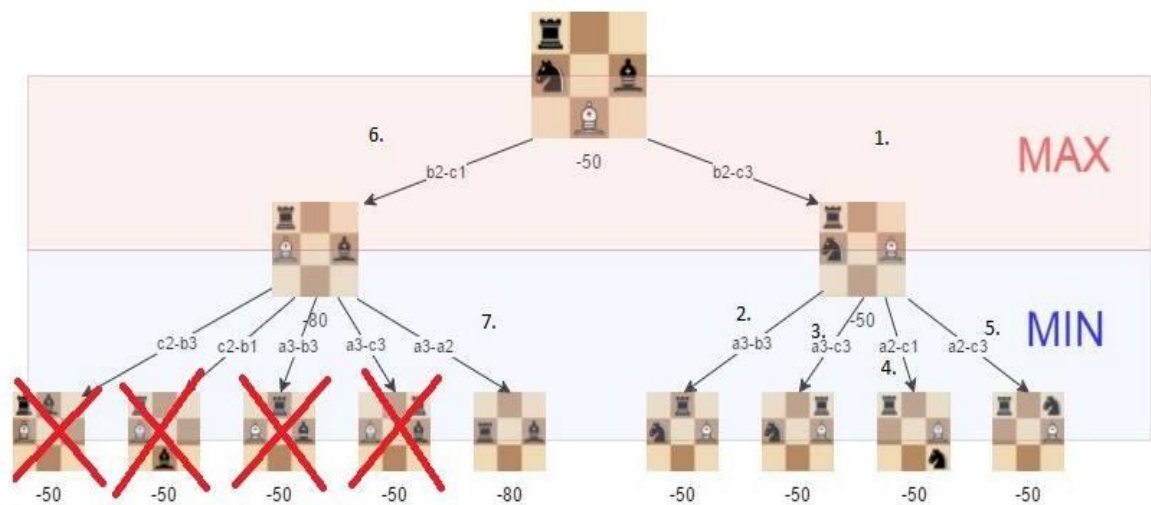
As shown in the above figure, the best move for white (our side) is b2-c3, as we can minimize the loss to -50.

The effectiveness of the Minimax algorithm relies heavily on the depth of search we implement, which we will continue to optimize.

STEP4 Alpha-beta pruning

Alpha-beta pruning is an optimization technique for the Minimax algorithm, which involves cutting off certain branches during the search process. This can save computational resources and increase the depth of the algorithm. The pruning algorithm abandons a branch if it discovers that the branch will lead to a worse outcome. This method does not affect the Minimax algorithm but can speed up the algorithm. Of course, if the optimal path can be found at the beginning, the alpha-beta algorithm will be more

effective.



STEP5 Improving the evaluation function

To optimize the function, we added a factor that evaluates the position of the chess pieces.

For example, a knight in the middle of the board is better than one in the corner (because it has more moves and is more active).

3.3 interface

API document.chm

AI.GetBestMove Method

Main interface: get best move after evaluated by fuction Minmax Search and Pruning Search(Alpha-Beta-cut)

Namespace: Chess
Assembly: Chess (in Chess.exe) Version: 1.0.0.0 (1.0.0.0)

Syntax

```
C# VB C++ F#
public static move_t GetBestMove(
    ChessBoard board,
    Player turn
)
```

Parameters

board
Type: Chess.ChessBoard
current ChessBoard state

turn
Type: Chess.Player
"Who's side are we viewing from.">

Return Value
Type: move_t
move_1

See Also

Reference
AI Class
Chess Namespace

4. AI search optimization

4.1 MINMAX and NEGAMAX

PVS (Principal Variation Search) is a variant of Alpha-Beta algorithm. The PVS algorithm is also commonly known as the NegaScout search algorithm. The ordinary minimax algorithm appears more complex, as one side attempts to take the maximum value while the other side tries to take the minimum value. Therefore, we need to check which side is going to take the maximum or minimum value, in order to perform different actions. In 1975, Knuth and Moore proposed the Negamax method, which eliminated the difference between the two sides and was concise and elegant. Using the Negamax method, both sides of the game take the maximum value. The core of the Negamax algorithm is that the value of the parent node is the negation of the maximum value of the values of its child nodes. To ensure the correct operation of this algorithm, there is an additional point to consider. For example, in chess, the evaluation function must be sensitive to which player is making the move. That is, if a positive evaluation value is returned for a position where the red player makes a move, then a negative evaluation value should be returned for a position where the black player makes a move. Note the negative sign involved.

In terms of principles, the MINMAX and NEGAMAX algorithms can be equivalent. However, their search efficiency is different. From the third level onwards, PVS search clearly surpasses the speed of MINMAX + Alpha-Beta search. Of course, the number of evaluated leaf nodes in the same depth search is also less than that of Alpha-Beta search. When the search depth is 5, the speed of PVS search reaches 250% of that of MINMAX + Alpha-Beta search.

Note: In this program, the PVS algorithm is indeed much faster than MINMAX, but the game-playing effect seems to be slightly different, with MINMAX appearing more stable. Therefore, in the program, search depth of up to 4 layers still use MINMAX search, while those exceeding 5 layers use NEGAMAX.

4.2 Transposition Table

During the minimax search process, the goal of improving the search algorithm is to eliminate as many unnecessary (redundant) branches from the search process as possible, in order to reduce computational costs by searching fewer branches. In the several search algorithms mentioned earlier, we have seen that redundant branches/nodes can be pruned using Alpha-Beta pruning, but nodes that have already been searched can be exempted from further search.

4.3 History Heuristic

The efficiency of Alpha-Beta pruning in AI search is almost entirely dependent on the ordering of nodes. In an ideal state of node ordering, the number of nodes that Alpha-Beta search needs to traverse is only about twice the square root of the number of nodes required by the minimax algorithm. That is to say, for a minimax tree that needs to traverse 106 nodes to obtain results, Alpha-Beta search in an ideal state only needs to traverse about 2,000 nodes to obtain results. However, when the ordering of nodes is the least ideal, the number of nodes that Alpha-Beta search needs to traverse is the same as that required by the minimax algorithm. Therefore, how to adjust the ordering of the moves to be expanded is the key to improving search efficiency.

One feasible approach is to adjust the order of nodes to be searched based on the partial results that have already been searched. Typically, when a good move is found for a position, there are often similar positions among its successor nodes, such as positions where only some unimportant chess pieces are different, and these similar positions are often good as well. These similar positions can be identified through some more complex judgments, and searched first, thereby improving pruning efficiency.

During the search, whenever a good move is found, an increment is added to the historical score corresponding to that move. A move that has been searched and confirmed to be good multiple times will have a higher historical record. When searching intermediate nodes, the moves are ordered based on their historical scores to obtain a better ordering. This method is much easier than sorting nodes based on chess knowledge. As the historical score table changes with the search, the ordering of nodes will also dynamically change accordingly.

Note: This method has been added, but its effectiveness is not as expected.

4.4 Killer Moves

The Killer heuristic method records the moves that cause the most pruning for each level. When searching at the same depth again, if the killer move is a legal move for the current position, it can be searched first. Since searching good nodes first has a high pruning efficiency, it is important to have a good ordering of nodes to be searched. The computation time spent on adjusting the node ordering is often worthwhile. When a transposition table is available, the best candidate branch can be selected based on the content of the transposition table.

Note: After debugging, this method did not work and often resulted in stupid moves, so it was commented out and not used in the program.

4.5 Quiescence Search/Capture Search

When the search reaches the maximum depth, it stops and returns a static evaluation. However, this approach has a problem: when the search terminates, the poor-quality positions that lurk afterwards cannot be recognized by the evaluation function. As a result, the program often falls into traps set by the opponent. Search capture is to continue searching all capture moves after reaching the maximum depth, evaluate the resulting positions, and sort them, making pruning more effective.

Note: After testing, this method only increases search time without improving efficiency, so it was commented out and not used.