Introduction of the New Algorithm

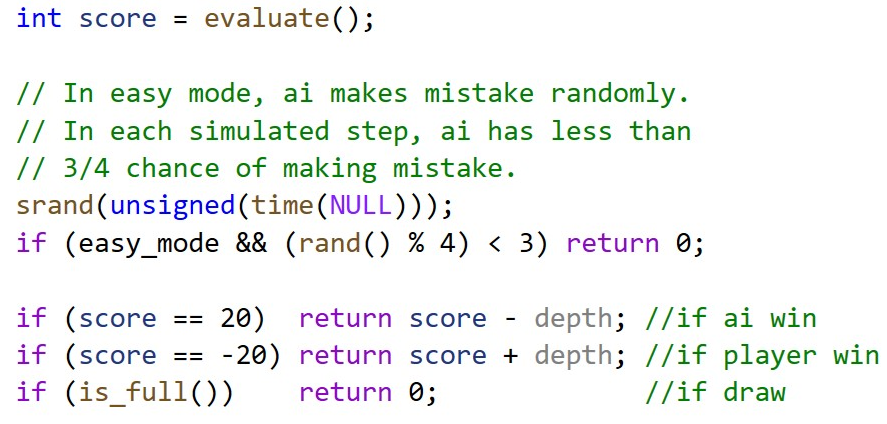
1. Old algorithm

The old algorithm that I used is the [Minimax algorithm](https://en.wikipedia.org/wiki/Minimax).

Three functions were used to implement it:

|  |  |
| --- | --- |
| int evaluate() | score each distribution of the chess pieces. |
| bool is\_full() | check whether the gameboard is full. |
| int minimax(int depth, bool isMax) | the implementation of the algorithm. |
| void ai\_move() | The implementation of the computer player |

1. New algorithm

A global boolean variable **easy\_mode** to check whether it’s the easy mode or utmost mode, and **two new lines** at the beginning of the function **minimax()** were added:

**🡨 The minimax function starts here.**

There are two possible mistakes:

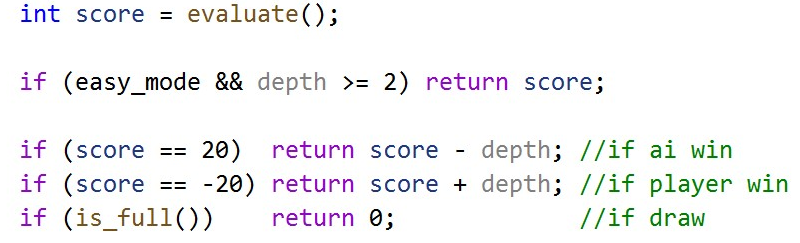
* The algorithm should return 20 or -20 but return 0 here
* The algorithm should search deeper but end here

For in each simulation, all possible steps will be considered, which magnifies the mistake-rate of the computer player. However, **return 0** here sometimes will also be the right result (//if draw), which will slightly decrease the mistake-rate.

In conclusion, the computer player in easy mode makes mistakes frequently, which allows child players to win from time to time.

1. Discussion

Another method to design the easy mode is to **constrain the depth of the minimax algorithm**. For example, the algorithm will return the current score whenever it reaches to the depth of 2.



But I think using rand() function makes the intelligence of the computer player unpredictable, which will **increase the playability of this mode**. What’s more, the mistake rate of the computer player can be adjusted freely and easily. That’s why I **abandoned this method**.