**Gomoku**

**Ver 1.0**

09016401

顾婷瑄

Botzone id: Gutingxuan

1. **Old versions review**

This is my very first version of Gomoku, and I am excited to introduce it to you.

1. **Literatures I read**

[1] 王志水. 基于搜索算法的人工智能在五子棋博弈中的应用研究[D].山东：中国石油大学, 2006

[2] Blog: 五子棋基本棋型@我是老邱

五子棋估值算法@maxuewei2

1. **Current idea of design**
2. **Essence**

A function based on the evaluation of each available position.

1. **Origin of the idea**

Myself

1. **Expectation**

The function can comprehensively take every potential board type into account and then make the best move.

1. **Implementation** 
   1. A function that collect all the positions that has neighbors in the distance of 2 so that it can save searching time.
   2. Two evaluation functions which will separately evaluate the attacking potential and the defense potential of each available position.
   3. A function that can evaluate the chess from eight directions.
   4. A function which will choose the best position to make the next move.
2. **Advantages of the design**

It is easy to implement and it is convenient to edit the current known board type so that the computer can give out a more accurate score.

1. **Weakness**

The computer can only evaluate the current board and give the score. It can not evaluate the potential moves in future turns and give out a better solution.

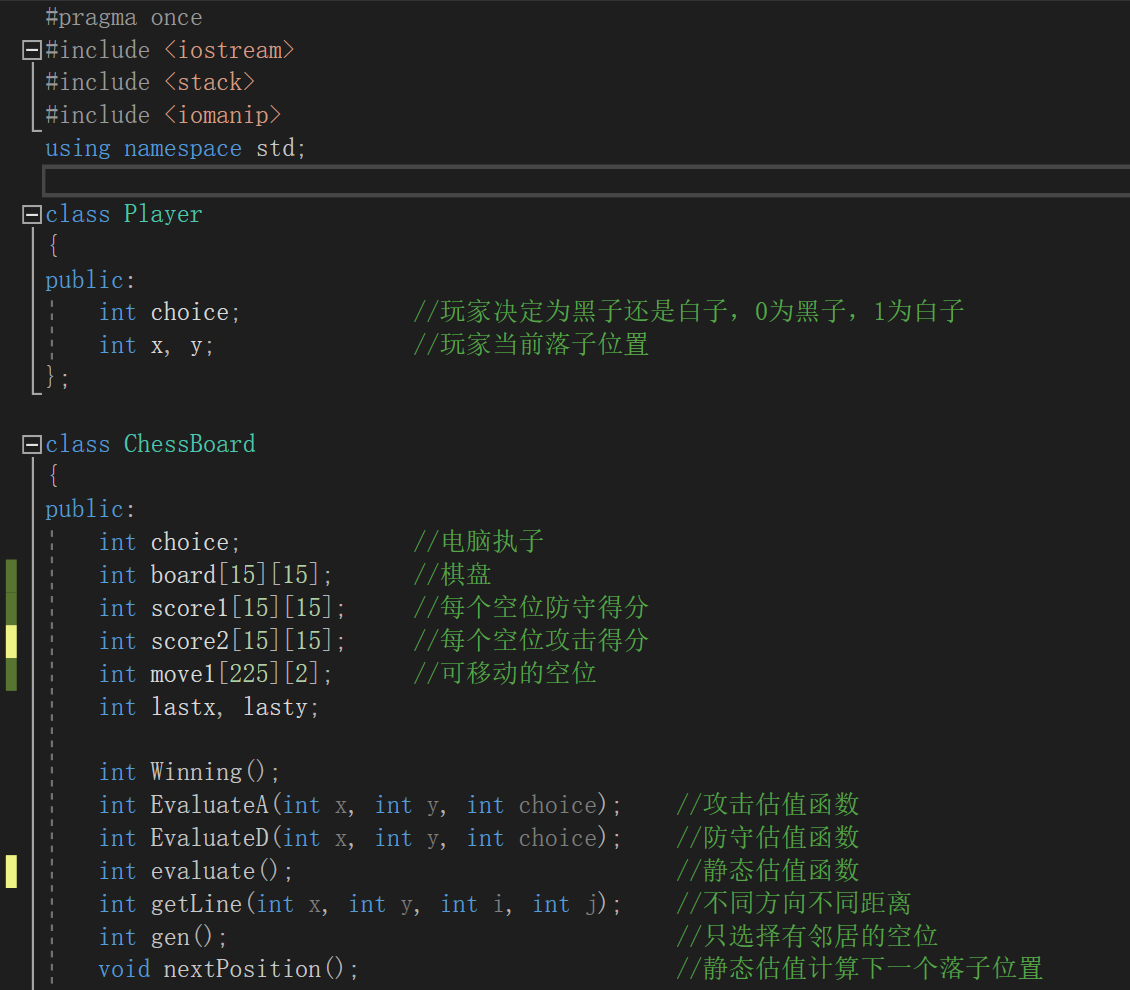
1. **Improvement directions**

I will implement a search algorithm which can foresee future moves and thus give out a better solution.

1. **Evaluation of the performance**

It is still pretty weak now since even I can beat it. However, after being posted on the Botzone, sometimes it can actually beat other Gomoku AI players. But since this is just the very first version, I am not ready to put it in the ranking list because the result might hurt feelings so there will not be any rank or score now. I am still glad it can successfully detective the winning move of the player and then prevent in advance and the accident victories really give me a thrill.

1. **Code structure and detailed implementations**
2. **The code structure**



1. **The score table**

Originally, the table goes:

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0**

**0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0**

**0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0**

**0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 0**

**0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1, 0**

**0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 4, 3, 2, 1, 0**

**0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0**

**0, 1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 0**

**0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0**

**0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 4, 3, 2, 1, 0**

**0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1, 0**

**0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 0**

**0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0**

**0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0**

Different position is given different initial score so that if the player goes black and makes the first move, then the computer will choose the position that near the center to make the move instead of making the move randomly from the first row.

/\*

\*: 当前空位置

0：其他空位置

1：当前player

2：对手

\*/

Different board type is given different scores:

Attack Defense

活四 01111\* 300000 300000

冲四A 21111\* 250000 200000

冲四B 1\*111 250000 200000

冲四C 11\*11 250000 200000

活三A-1 0111\*0 60000 50000

活三B-1 01\*110 60000 50000

活三B-2 \*10110 40000 30000

活三B-3 01011\* 40000 30000

眠三A-1 2111\*0 3000 2000

眠三A-2 21110\* 1500 500

眠三B-1 211\*10 3000 2000

眠三B-2 21101\* 1800 800

眠三C-1 21\*110 3000 2000

眠三C-2 21011\* 1800 800

眠三D-1 211\*01 1600 600

眠三D-2 2110\*1 1600 600

眠三E 21\*101 1550 550

活二A-1 0\*0110 1650 650

活二A-2 0\*110 4000 3000

活二B-1 0\*1010 1650 650

活二B-2 01\*10 4000 3000

活二C-1 01\*010 1650 650

活二C-2 \*10010 1650 650

眠二A-1 211\*00 1150 150

眠二A-2 2110\*0 1150 150

眠二A-3 21100\* 1150 150

眠二B-1 21\*100 1250 250

眠二B-2 2101\*0 1250 250

眠二B-3 21010\* 1250 250

眠二C-1 21\*010 1200 200

眠二C-2 210\*10 1200 200

眠二C-3 21001\* 1200 200

眠二D-1 21\*001 1100 100

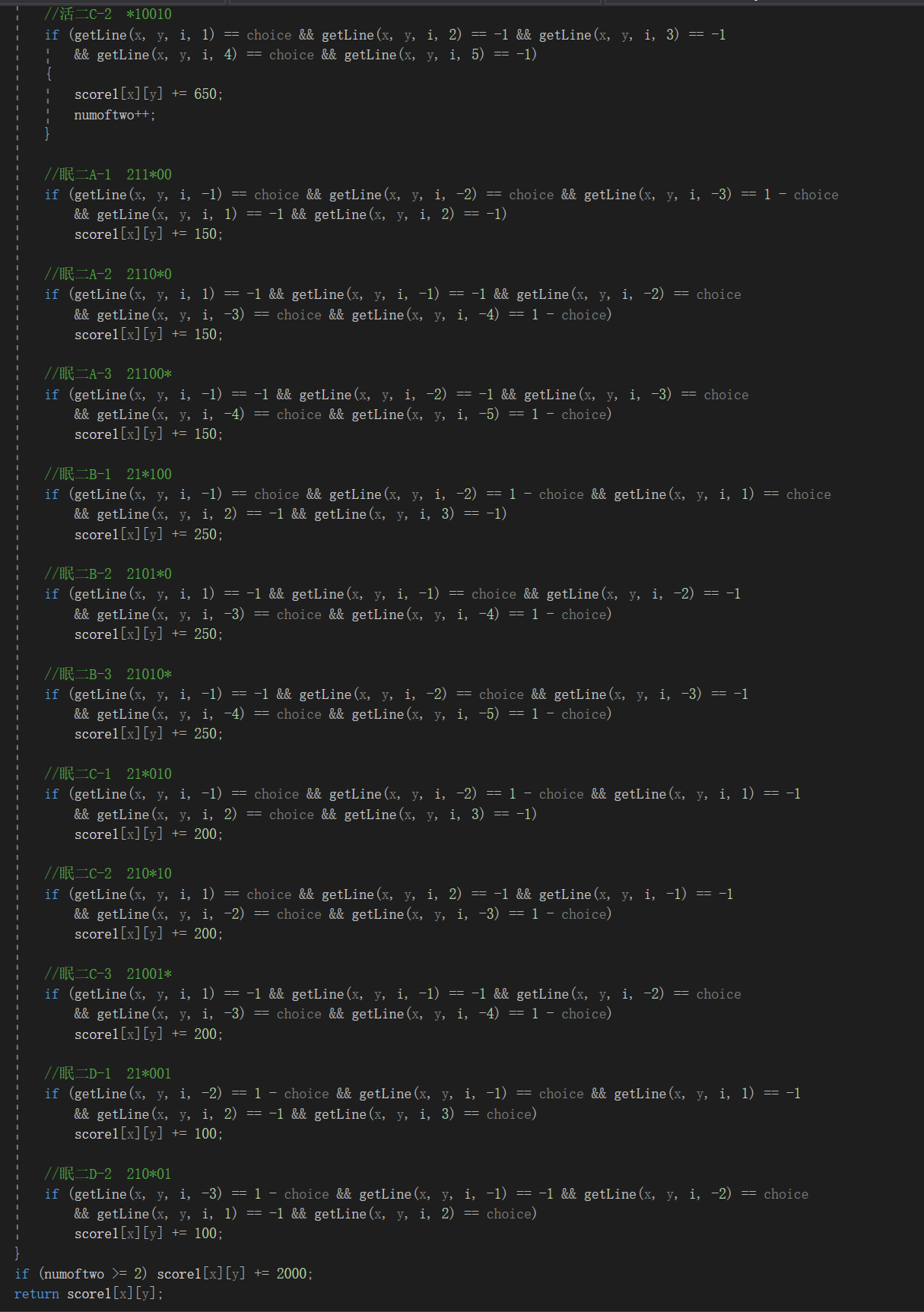
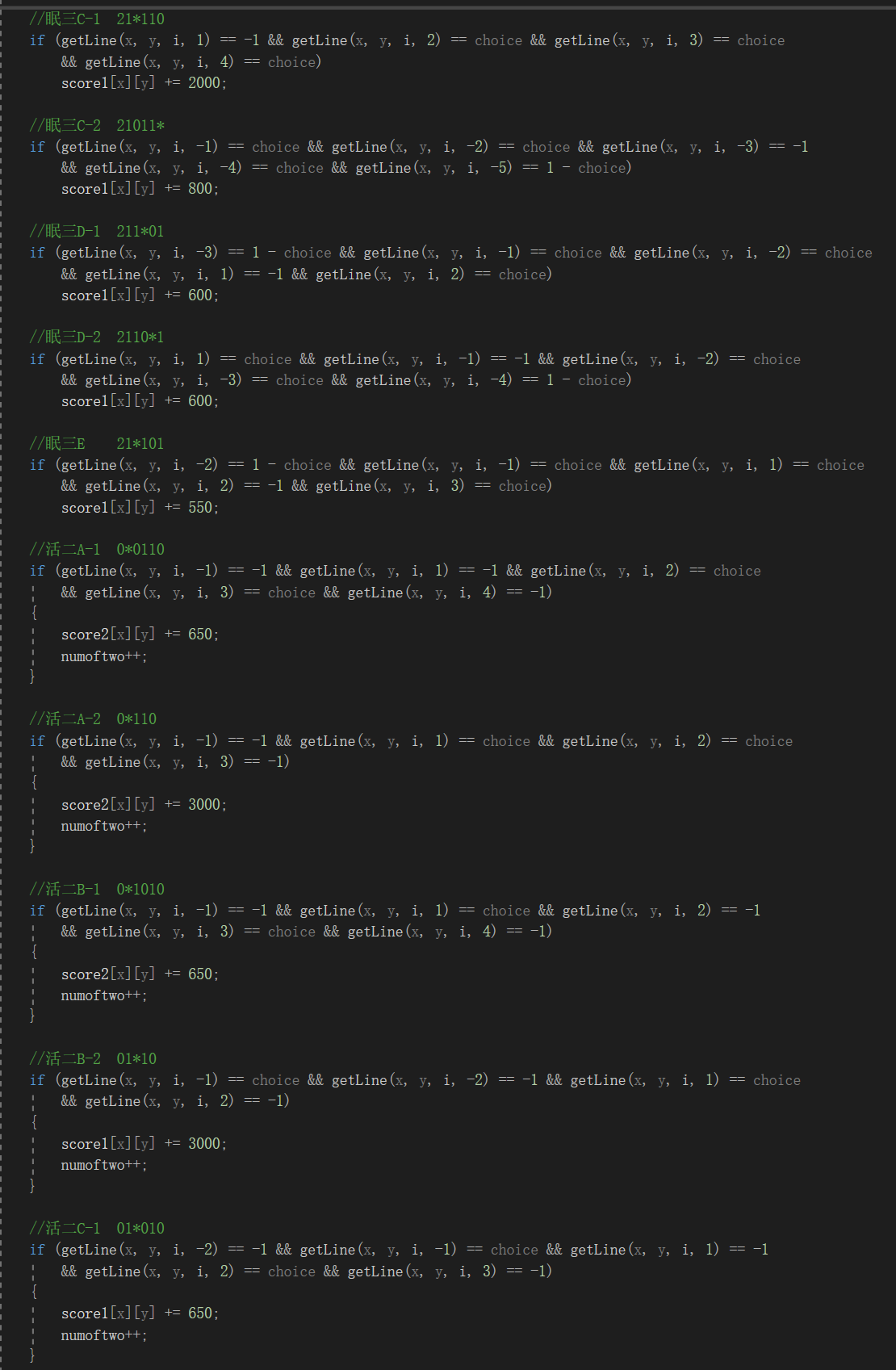
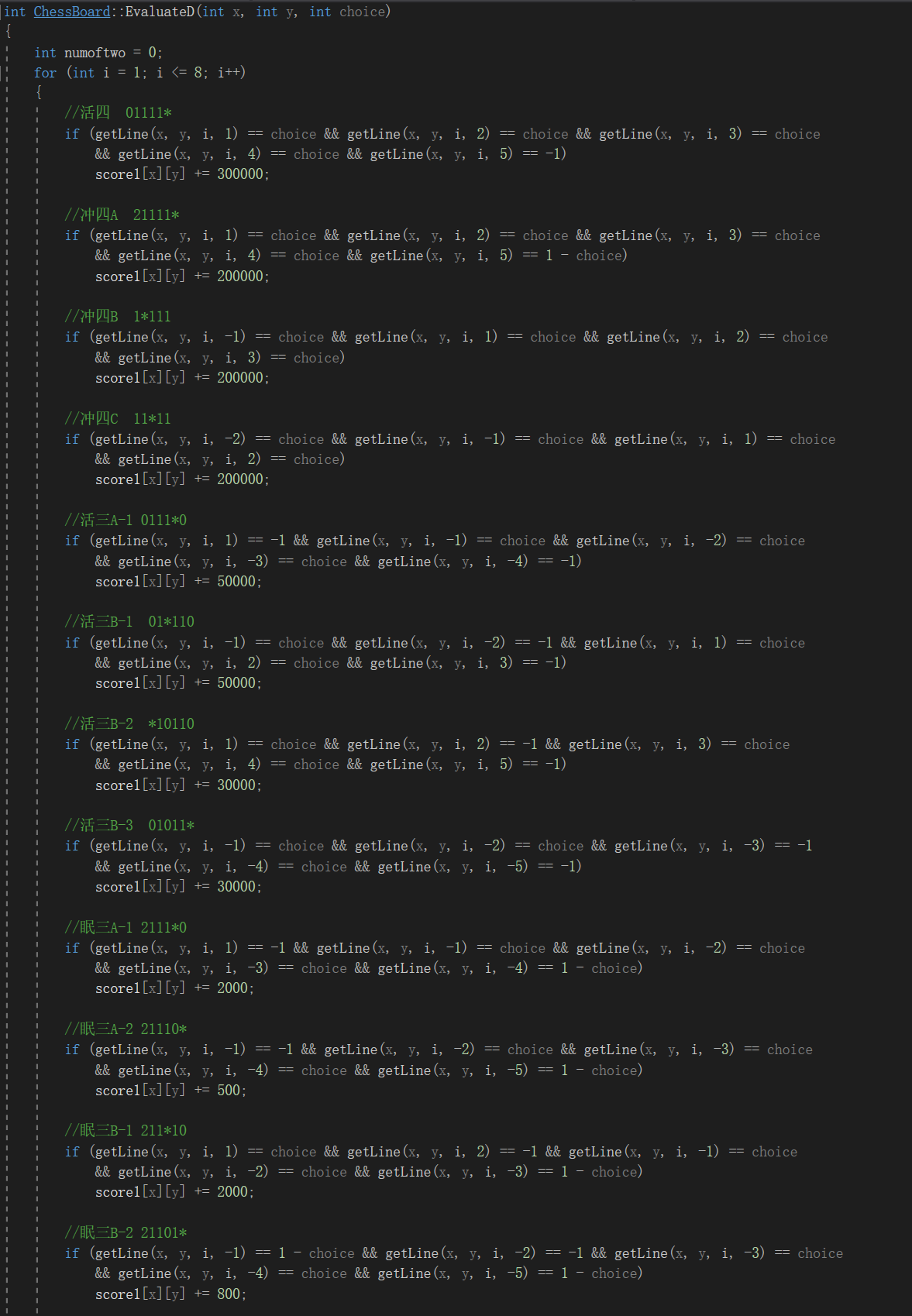
眠二D-2 210\*01 1100 100

Numoftwo>=2 3000 2000

Different board type scores can help computer choose the best position to make the move. Attack score is the one that computer can gain if it places the stone on a position and the defense score is the one that player can gain if he places the stone on a position. Therefore, the computer will have to choose whether to make a move to attack or to defense in case the player will make a move to gain a higher score. And I choose to make the computer a more competitive one so the attack score will be a little bit higher than defense score in the same board type case. And after playing a few games, I found that the number of alive-two board types sometimes means a lot so I give extra points to the position that meets at least two alive-two board types.

1. **How to evaluate the score of each position**

Based on the score table and the search from eight directions that whether this available position is in any valuable board type.



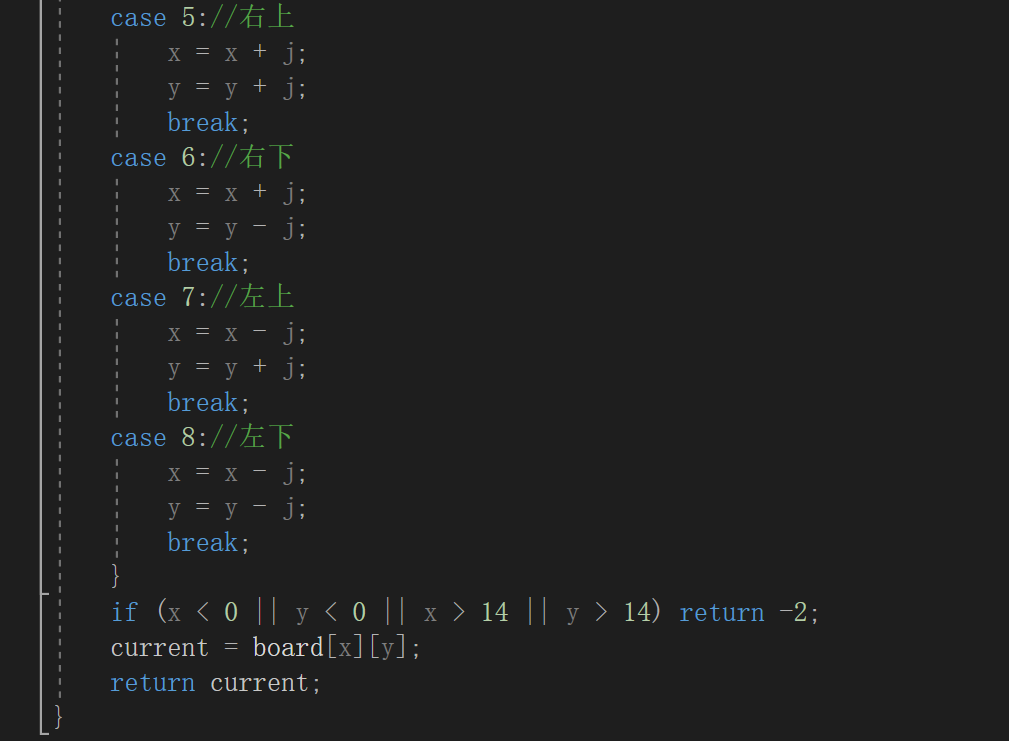
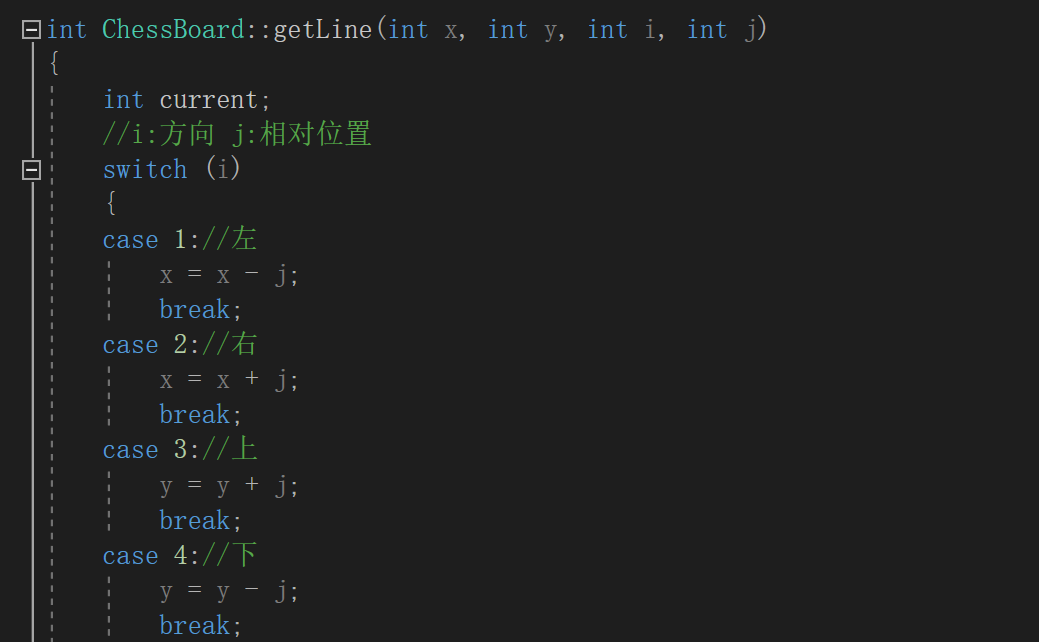
1. **How to evaluate 8 directions**

As we all know, there are four kinds of rows in a Gomoku game so if we need to value a position, we need to value it in eight different directions. The getLine() function can help us get the coordinate in eight directions in a easy way and we can choose the distance we want.

// x , y means the current position

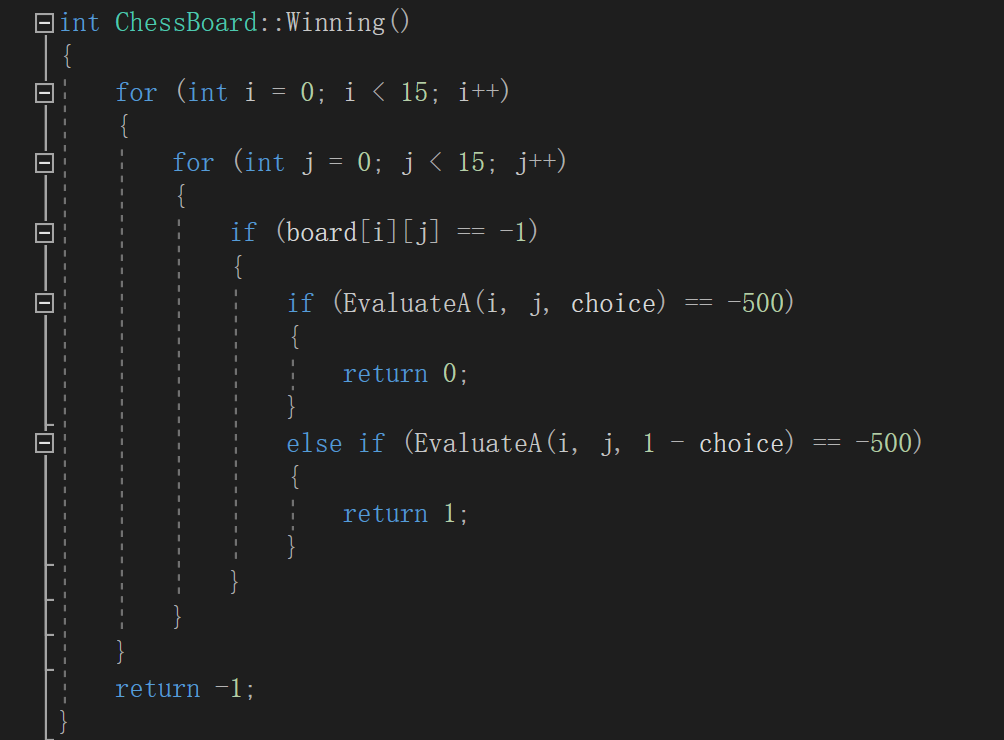
// i is the direction

// j is the relative distance



1. **The winning function**

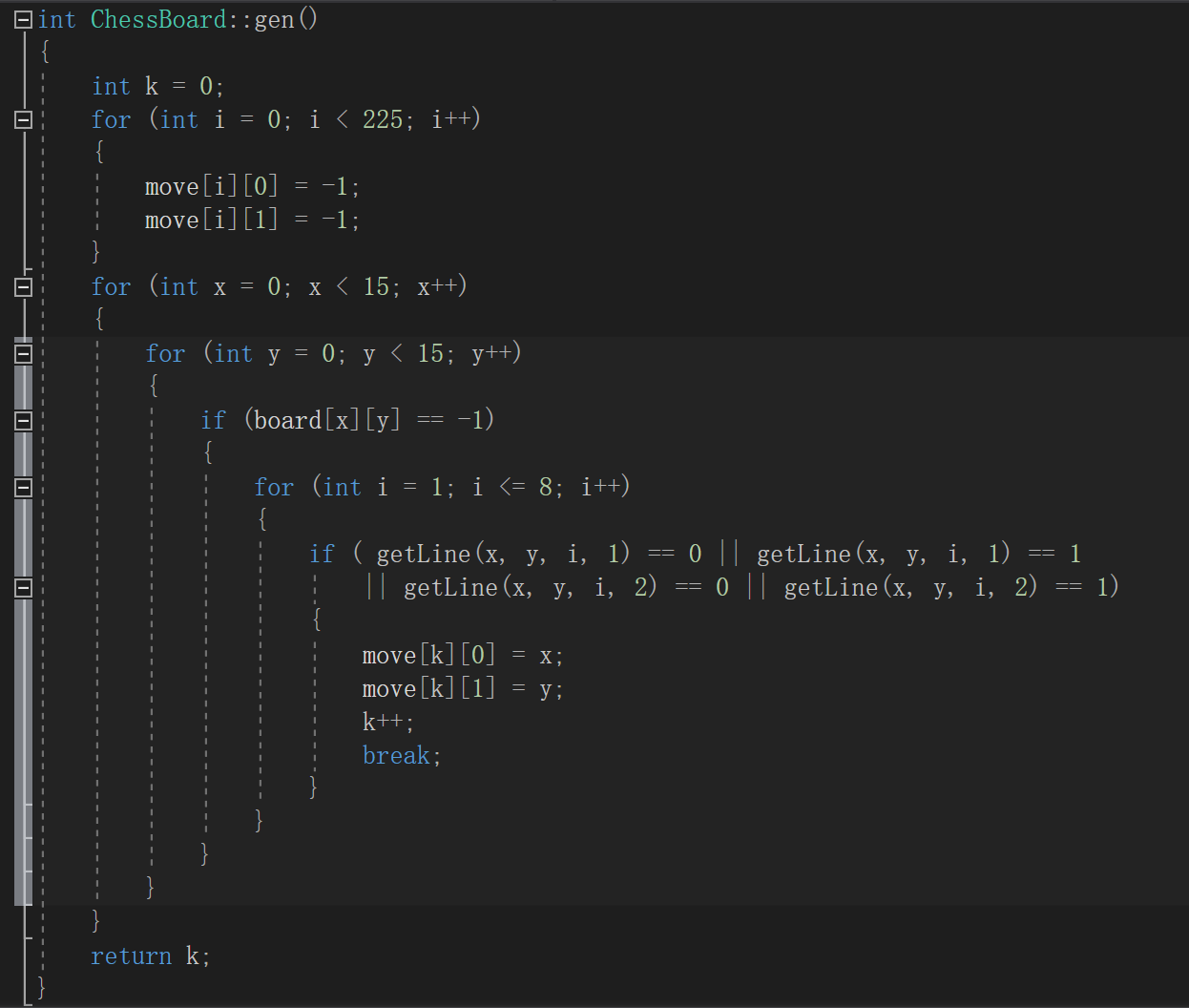
If the evaluation process detective a five-in-a-row situation, the score of that position will be -500. And if the winning function detective a score of -500, then the game will come to an end. This function will not be used when the program is put on the Botzone.



1. **The function that collect all the available position that has neighbors in the distance of 2**

Based on experience, it is usually a waste of time to evaluate the positions that have no neighbors nearby. Therefore, I decided to evaluate only those positions that has neighbors in the distance of two in order to save the time of searching. Basically it just uses the getLine() function to search whether there is a black or white stone in the distance of two.

However, when applying this function on the Botzone, there seems to be a problem in the later period of the game, so I decided to use it in the future search algorithm instead of the current version.



1. **The function that makes the next move**

This is the main function in this version of design that makes the next move. It evaluates the attack potential and the defense potential of each position and picks out the position which has the max attack score and the position that has the max defense potential. Compare these two scores and choose the bigger one to make the move. If two positions have the same attack score, choose the one that has a bigger defense score in it to make the move.

