---------AI project 3 Gomoko----------

By

Jinxi Zhao (jz2540)

Programming Language and Version: Java 1.6

Development Environment: Eclipse Java IDE v3.2 on Windows 8

------Evaluation function-------

My evaluation is based on the overall trend of a specific board condition. For every open spot on the board, my agent plugs in a black stone and a white stone respectively, and evaluate scores for them (one score is positive, one is negative), based on searching results from 8 directions.

```
For example:
    ......x....                .......x....
    .....x.....                ......x.....
    ..oo?......       ->       ...oox......
    ...........                ............
    ...........                ............
    ...........                ............
```

If a black stone is plugged in, and forms a three-stone chain (let's assuming it's a normal Gomoko with 5-stone-chain to win), give a score of a three-stone chain; meanwhile, it detects a 2-stone chain of white, give a negative store for white, add the positive score and negative score separately (with two variables). My agent do this "sum up" for every open spot, and finally add them all into the two variable.

Now the agent check if this is black's turn or white's. If it is, agent's turn (let's say black), we know that resulting in this condition is good for my agent. Because my agent can put a stone on it, and form a three-stone chain, while the white only has a two-stone dead chain.

But if it is now the white's turn, we the board is going to be like this, obviously, it is also good to the white. That's why my evaluation function needs to check whether it is agent's turn or the opponent's. If it is agent's turn, return a positive value, so that it can be choose; otherwise, return a negative, so that it won't be considered by Max-value function.

Also, if the agent detects that this board will form a winning/losing move, it weights it more by multiplying it by a large number, to make sure that it is the max-value/min-value for all, so that the agent will definitely select it if it is its turn; otherwise, avoid the moves that lead to this situation.

For the quickness, since I have defined a range, and only search open spots within it, it is very quick to execute the evaluation. I have tested it, most of time, it takes less than 1ms execute the whole evaluation function.

So it looks like a good evaluation function for me either in perspective of accuracy or efficiency.

----Test reports----

Battle Mode:

Well I have to say either I am so bad at playing this, or my agent is doing not as bad as I think. For five times, I won three times. My agent does make some pointless moves(at least to me) in early game, I guess it is because the the weights for different chains are not balanced well, so that the score doesn't accurately reflect the real fact.

AI vs. Random:

My agent always wins, usually within 10 moves. The same problem, in early game, the score is not so accurate. It tends to put a stone next to the randomly-put stone, rather than form its own chain. But once it sees a two or three chains, it can continue to make winning chain.

AI vs. AI:

The one who moves first, always win, no exception. I watched them play move by move. Generally, they played moves that is good to themselves, but they were not playing the best moves sometimes. For example, it tends to block opponent's chain rather than make its own chain.

------------Otheres-------------

I have implemented a iterative alpha-beta pruning algorithm, which iterates depth limit from 3 to 10. Why I set it to 3 to 10, it is because even you set time limit to 1 second, it can easily iterates to 5, just in case, I set a lower value of 3. For the 10 can larger, I don't see much difference on moves, so I conclude that it may be just waste of time for my agent to search move. When I tested it only laptop, even I allow 60 seconds, it usually output a move within 30 seconds. So I guess this is acceptable.

When time is up, and my algorithm will drop anything it get from the iteration. For example, with 6 depth iteration, it got a best move [5 6]. When it increase depth limit to 7 and is running, time's up, then return the best result from 6-depth limit iteration.

For the score evaluation, since scores for better type should have more score so that to make sure it can be chosen. In my algorithm, I use factorial to calculate scores, say total number of type is 11 with winning chain M being 5, then the best type is type 1, which worth 11!, while the 11th type worth only 1.