An implementation of the game Gomoku using Minimax algorithm with quiescent search

Vy Ung

Minerva Schools at KGI

1/ Problem Definition:

For this small project, I programmed an AI opponent for the game Gomoku in Python. The rule is quite simple - it is an extended version of Tic-Tac-Toe . Playing on a board of typical size 15x15, whoever got 5 consecutive symbols in a row horizontally, vertically or diagonally first will win.
It was believed that whoever goes first will have an extremely high chance of winning the game, but results from human playing cannot justify this claim as we are of course not rational all the time and do not have a perfect strategy to ensure winning. However, this claim could be better supported or weakened by a simulation using AI pot or something similar. Therefore, I believe AI approaches are a good fit to solve this problem. First, we can devise a super AI player that can beat human, and then we let the AI players play with each other to get more insights into the pattern of the game.
This is an interesting problem to solve because first it has a large search space (similar but simpler than chess) which can take up expensive computational resources. However, because the effectiveness of the AI for this game not only relies on search algorithm but also on the "domain-specific strategic knowledge" (Allis, van den Herik & Huntjens, 2007), we can devise a powerful heuristic function that can evaluate the importance of a move on the board and from there we may be able to confidently cut the search depth.

2/ Solution Specification:

My first attempt was to simply use Minimax Algorithm together with Alpha-Beta pruning, but it took so long for the algorithm to run even with a small-sized board (10). Therefore, I decided to look for different methods to speed up the process.

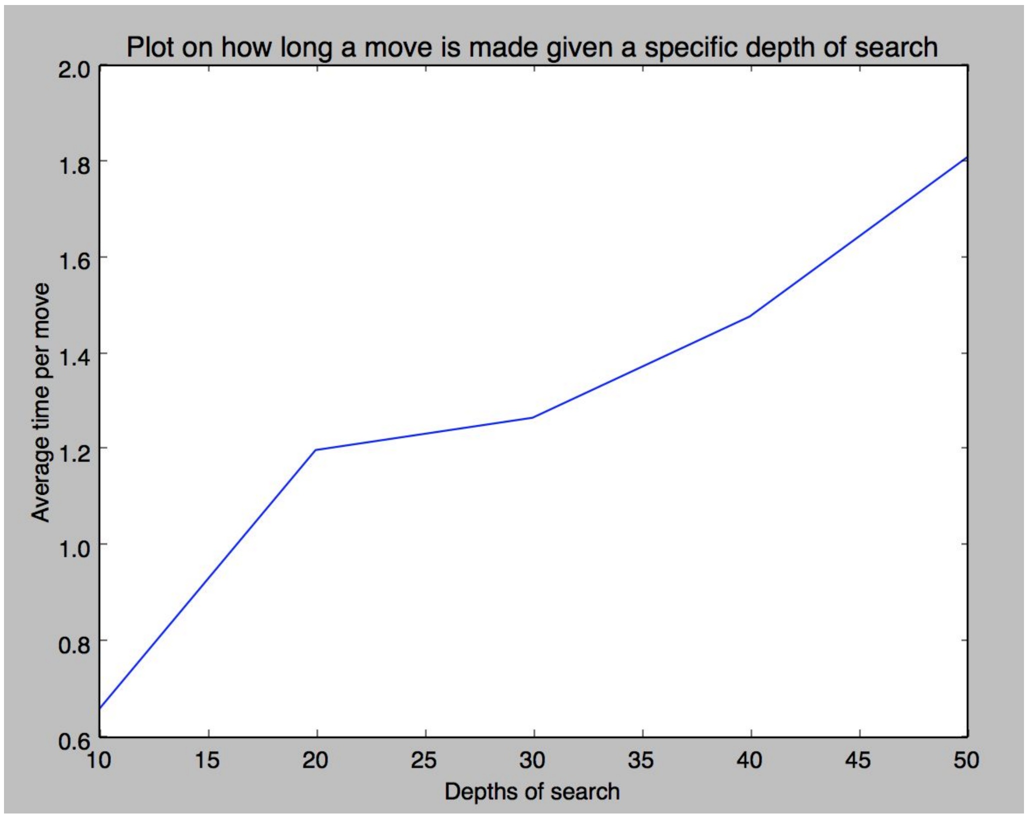
From online research and synthesis, I finally devised a quite stronger evaluation function to give score to a specific position given the current state of the board. It does so by evaluating the number of ways that that position can be used to get to winning state. However, because the best move for one player is also the best move for the other player (to avoid the best move of the one player) in this zero-sum game, the importance of a specific position could be evaluated as the sum of its attack power and its defense power for a given player.

Now when we already got the evaluation function, search could be performed. However, for a board with size 15, for example, the state space would be enormous $(15 \times 15)^3 = 11390625$ and the algorithm may get lost in evaluating poor scenarios while there would be some obvious winning moves available. This "horizon effect" is known to be resolved by quiescent search, in which the algorithm abandon most of the "bad-looking" moves and search all the promising moves to a great depth. Those bad moves are considered as quiet move, hence the name "quiescent search". More specifically, I implemented an n-best-selective search which only looks for the best n moves at each node. (A., Abdel, Gadallah & El-Deeb, 2014). In addition, I also integrate a cutoff for search to speed up the process. As you can see in the function to find next move, there are two parameters that can be specified: the number of best moves that we will perform further search on, and the depth of our search. More details of the algorithm are included in the code comment.

3/ Analysis of Solution:

As expected, when we increase the depth of search, the average time it takes to make a move increases. I did not run multiple trials for one specific value of depth search because as it is programmed, there is no randomness involved so the game will be the same with the same input.

I did not have a chance to run for larger depth to deduce the complexity, but this plot is quite sufficient in showing the trend.



One interesting result is that when I let two AIs play with each other, it actually occurs that the one moves second would win. It was on a board of 10x10 size, with max_consecutive value of 5, with a search depth of 30 and the number of best moves chosen from each step is 10. This goes against the hypothesis as mentioned in the problem description that who goes first will always win the game. Also, by observing such a game, I found out that in the beginning, the white player (who goes second) may seem to make "non-sense" moves, but they end up leading to a winning move. This proves that the evaluation function together with quiescent search are pretty effective in looking ahead of the game and go for the most promising path. As you can see in Appendix, the states that were colored red were the leading states into winning, and it is not quite obvious to get there (as least for me).

REFERENCES

A., A., Abdel, M., Gadallah, M., & El-Deeb, H. (2014). A Comparative Study of Game Tree Searching Methods. International Journal Of Advanced Computer Science And Applications, 5(5). http://dx.doi.org/10.14569/ijacsa.2014.050510

Allis, L., van den Herik, H., & Huntjens, M. (2007). Go-Moku and Threat-Space Search.

haslam22/gomoku. (2017). GitHub. Retrieved 19 April 2018, from https://github.com/haslam22/gomoku

Minimax for Gomoku (Connect Five). Blog.theofekfoundation.org. Retrieved 19 April 2018, from https://blog.theofekfoundation.org/artificial-intelligence/2015/12/11/minimax-for-gomoku-connect-five/

Nero144/Gomoku. (2014). GitHub. Retrieved 18 April 2018, from https://github.com/Nero144/Gomoku

tdang33/Gomoku-Five-in-a-row-. (2015). GitHub. Retrieved 19 April 2018, from https://github.com/tdang33/Gomoku-Five-in-a-row-