****

**Project Report: AI Games of Gomoku**

**Course Code:** GE2340 Artificial Intelligence

**Team Members:** Chan Tai To - 54791401

LI Haotian - 54780576

WANG Fangran - 54777001

**Date Submitted:** 02/12/2019

**Table of Contents**

[Introduction 3](#_Toc26209427)

[Objective 4](#_Toc26209428)

[Matches Result 5](#_Toc26209429)

[Future Improvements 6](#_Toc26209430)

[Reduce Decision Time 6](#_Toc26209431)

[Facilitate Online Matches 6](#_Toc26209432)

[References 7](#_Toc26209433)

# Introduction

The Gomoku also called *Five in a Row*, is a two-player abstract strategy board game conventionally played on a Go board. During the game, the players alternate turns to place black or white stones and the game requires an unbroken horizontal, vertical or diagonal row of stones to win.

The initial AI players of Gomoku or other board games like Go use classical search algorithms to choose their moves. Searching a large number of possibilities in the game tree could cost a considerable amount of computing time, and the process is improved by alpha-beta pruning to decrease nodes (Knuth & Moore, 1975). The pruning algorithm would stop evaluation when a move is found worse than previously checked ones. A remarkable result using alpha-beta pruning was that Deep Blue defeated then-world human chess champion in 1997 (Silver et al., 2018). Preceding AlphaGo Zero and Alpha Zero, computer programs were augmented to better evaluate positions by improving the search algorithms with heuristics and domain-specific adaptations. Alpha-beta pruning is still used as the workhorse of the predominant computer chess engine Stockfish today (Tian et al., 2019).

Despite the comparatively powerful applications of these programs using artificial intelligence, their capacity is highly restricted by human experts’ data. A tremendous breakthrough is the AlphaGo Zero developed by DeepMind that is trained solely by self-play reinforcement learning to achieve superhuman performance in Go without any human data’s supervision accept the rules of the game (Silver et al., 2017). The principle of reinforcement learning is that if a strategy leads to a favourable reward, the tendency to use this strategy will be strengthened. The algorithms are further generalized to become AlphaZero and accommodate a broader range of board games (Silver et al., 2018). The full concepts and implementation of AlphaZero are published in Silver et al. (2018) and the essential ideas are extracted here.

AlphaZero uses a deep neural network as the workhorse , parameterised by . The neural network takes the board state *s* as an input and outputs a probability vector for possible moves and a scalar value *v* estimating the anticipated outcome from the perspective of the current player. Without the support of the human database, AlphaZero learns to move probabilities and estimation values from self-play.

The search progress is enhanced by Monte Carlo Tree Search (MCTS) algorithm instead of alpha-beta pruning. Starting from an empty search tree, a series of simulated games of self-play are expanded from root sate to a leaf state. At each state, MCTS prioritizes a move not previously frequently explored and high unitality value according to the neural network’s current policy, either proportionally or greedily. A common heuristic used in the process is to find the move that maximizes the upper confidence bound applied to the search tree (Tian et al., 2019). The function consists of the expected reward for taking a specific move from state *s*, the number of times taking the move from state *s* across simulations and the probability vector . If the move leads to a new state, add the new state to the tree and initialize **p** and v from the neutral network. At the end of the game, the terminal position is scored according to the rules. After a few simulations, the neutral network can provide a better policy noted as a vector representing the probability distribution over moves. The updated parameters are used in the subsequent self-play games.

# Objective

This project aims to implement the existing methods of AlphaZero algorithm for Gomoku and test the performances by letting it play online matches against the AI adopted by the website. The opponent AI’s method is unknown but speculated as an alpha-beta searching algorithm.

We used two online Gomoku match websites to evaluate to performance of AlphaZero. The access to the website:

* <https://dkmgames.com/Gomoku/gomoku.htm>
* <http://gomoku.yjyao.com/>

The URL of our project in GitHub: <https://github.com/BigtoC/GE2340-AI-Gomoku>.

# Methodology

## AlphaGo

As the predecessor of AlphaZero, AlphaGo was still a lengend in the field of deep learning. AlphaGo is consisted of these main components: Policy Network, Value Network, Fast Rollout and MCTS (Monte Carlo Tree Search). Policy Network predict the next move in a game. The Value Network gives out the probability of winning for both player. Fast rollout is the same use as traditional strategic algorithm. MCTS is one of the most important algorithm to generate, predict and prune the game tree.

A close up of a map

Description automatically generated

Picture 1. Learning flow

## AlphaZero

Compared to AlphaGo, the biggest difference is that AlphaZero use no human chess pattern for training. AlphaZero possess only one single neural network for whole learning process. There are three section during the learning process: Self-play, Retrain Network, Evaluate Network. Initially the system perform self-play and collecting data from these games. Each game state and the probability of each move in the game will be stored according to MCTS. Then it will randomly choose 2048 samples from the collected data then perfrom thousands itereation of Retrain Network.

A close up of a map

Description automatically generated

Picture 2. Comparison

## Model Structure

The model we use in this project is basically a simpler version (<https://github.com/initial-h/AlphaZero_Gomoku_MPI>) of the original AlphaZero. For the network structure. The current model has 19 residual blocks, which may leads to higher accuracy. The trade-off will be the computing-time. During online matching, we surprisingly find that it may need to update all 19 residual blocks then can place its move which leads to around 5 seconds per move.

* As for the filters in Convolutional layers is shown as follows.

A screenshot of a cell phone

Description automatically generated

Picture 3. Network Structure

As for all the feature planes, there exist a handy API for setting parameters for selecting features in training. Another difference between this model and the original model is that the author added Dirichlet noises in every node instead of the root node only. The weights between prior probability and noises are still (0.75/0.25) for current model. Though the author suggests that need to increase the value to 8:2 or even 9:1 since the noises are added more. In this way it can be balanced off.

The parameters in the whole model is shown in the following table:

A screenshot of a cell phone

Description automatically generated

Picture 4. Parameter table

## Model Training

Initially, our group tends to train our own model with the above structure. It tooks around 16 hours for a laptop to run but only 80 iterations has been done. Therefore the results is very poor. After the failed attempt, we implemented the trained model provided by the author for a 15\*15 game board. This model was trained and performed 100,000 games and it took 800 hours. This is also a very hardware-consuming job. The configuration is 2 CPU and 2 1080ti GPU.

## Matching with Traditional AI

By traditional AI, we selected a few online Gomoku scripts and flash games. These AI are implemented in strategic programming such as min-max search etc. and the move is fixed if you provide the same input. Since AlphaZero (in our project) uses ResNet it records previous moves and noises which makes its hard to predict and more accurate (but slow at the meantime).

As for Gomoku games with GUI, we tried using crawlers in python to geet its move and pass that information into our model. Unfortunately our model runs very slow due to 19 residual blocksto generates a move which may leads to error for crawler. We also tried to copy the scripts and abandon the GUI and play in terminal. But due to time limit and technical issue we performed manual update on both platforms.

# Code Explanation

* Our project is available on Github: <https://github.com/BigtoC/GE2340-AI-Gomoku>.
* File Structure:

A screenshot of a cell phone

Description automatically generated

* OnlineMatch folder includes all the script and web crawlers and cv implementation for auto-matching with internet AI.
* AlphaZero folder includes all training algorithm, game board and Command Line Interface (CLI). The GUI for this model is not compatible well with other platform, thus we abandoned the GUI for this model.
* Model\_15\_15\_5 contains the well trained model. This model was evolved from a 11\*11 game board model since training from zero will be much more time-consuming.
* The following code snippet is used for printing evaluating matrix after Alphazero make a move:

A screenshot of a social media post

Description automatically generated

* This class from mcts\_alphazero.py is a implementation of MCTS which is used for generating the game tree.

A screenshot of a social media post

Description automatically generated

# Matches Result

The matches with online Gomoku are quite good, AlphaZero won in most of the time.

The record of matches results show in below:

Online Gomoku VS AlphaZero

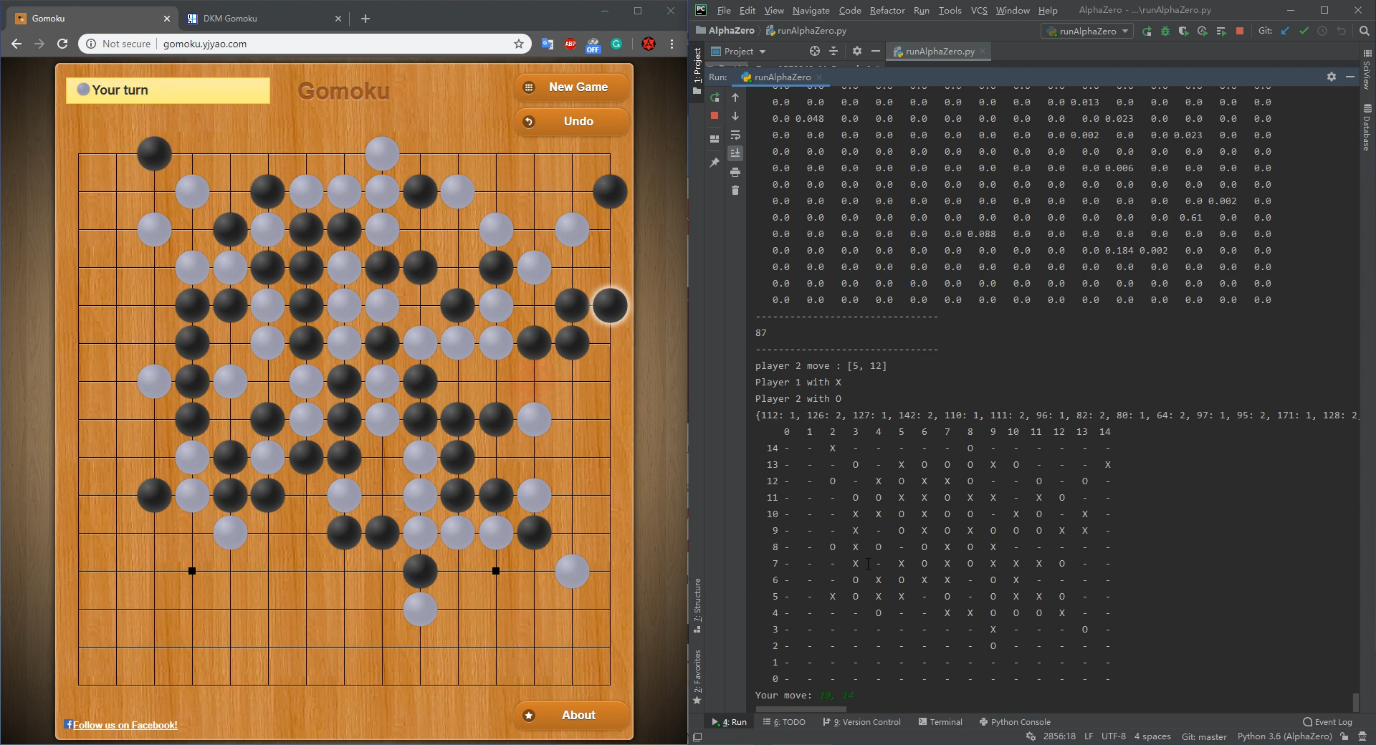
<http://gomoku.yjyao.com/>

1. Round one: 1 : 0
2. Round two: 2 : 0
3. Round three: 2 : 1
4. Round four: 2 : 2
5. Round five: 2 : 3

<https://dkmgames.com/Gomoku/gomoku.htm>

1. Round one: 0 : 1
2. Round two: 0 : 2
3. Round three: 0 : 3
4. Round four: 0 : 4
5. Round five: 0 : 5

The results are also available on YouTube: <https://www.youtube.com/playlist?list=PL4nBMCNLJMG7MNGBUwLIJyixv4ZU-GYID>



During the matches, we found that AlphaZero must go through a large operation to calculate the strategy, that means it took a lot of time for thinking, even the chessboard show that the opponent already had four chess in a line.

There is a fun fact that we observe. AlphaZero plays many unusual moves that we made us hard to understand. But in the end, we found that it did play a smart move and help it win the match. Maybe this is the advantage when AI play chess matches. They do not limit by human’s rules yet develop their own brilliant strategies.

# Future Improvements

## Reduce Decision Time

Like we mentioned before, even some obvious movements like the opponent or itself already got four chess in a line, we can hardcode this rule for AlphaZero to reduce the decision time and calculation resource. A checker can be developed to check if there are four chess in one line, and this checker should run before AlphaZero active the strategy-making progress.

Also, a computer with a better GPU is helpful for speed up the calculation progress.

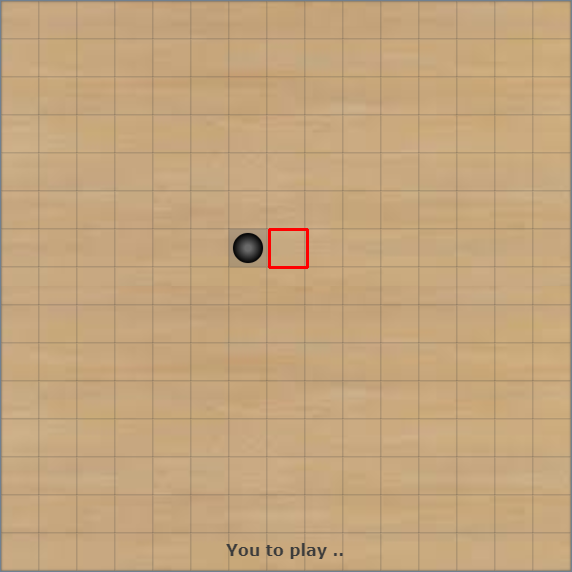
## Facilitate Online Matches

An automatics program is developing. The program is developed in Python, and use Pyppeteer for auto chess playing via browsers. Pyppeteer is an unofficial Python port of puppeteer JavaScript (headless) chrome/chromium browser automation library, and most things that you can do manually in the browser can be done by Pyppeteer.

Because most of the Gomoku match websites use Javascript to implement the decision making, that means the calculations are done locally, no network traffic can be monitored, so we can use OpenCV to detect the movements made by opponents (the detection result is shown in the below picture), and send the chess placement data to AlphaZero. After AlphaZero got the movement of the opponent, it starts the calculation, and send back the data. Once the other program got data from AlphaZero, in the help of Pyppeteer, it clicks the corresponding location in the website.

But the communication between AlphaZero and Pyppeteer is still developing, but the functions of auto visit the match websites, start a match and place chess are finished developing. As you can see in the demo video, we had to input the locations of opponents’ movements manually.

If we can finished the online matching program, we can facilitate the training in different platforms, let AlphaZero learn as much as different strategies, and help it improve quickly.



# References

[1] Knuth, D. E. and Moore, R. W (1975). An analysis of alpha-beta pruning. *Artificial intelligence*, 6, 293–326

[2] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, D., Graepel, T. and Hassabis, D (2017). Mastering the game of go without human knowledge. *Nature*: 550, 354-359

[3] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*: 362, 1140–1144.

[4] Tian, Y., Ma J., Gong Q., Sengupta S., Chen Z., Pinkerton J. and Zitnick C. L (2019). *Proceedings of the 36th* *International Conference on Machine Learning*, Long Beach, California, PMLR 97.

[5] *https://medium.com/applied-data-science/alphago-zero-explained-in-one-diagram-365f5abf67e0*

**TOBE SITED WELL!!!!!!!!!!!!!!!!**

<https://www.nature.com/articles/nature24270.epdf?author_access_token=VJXbVjaSHxFoctQQ4p2k4tRgN0jAjWel9jnR3ZoTv0PVW4gB86EEpGqTRDtpIz-2rmo8-KG06gqVobU5NSCFeHILHcVFUeMsbvwS-lxjqQGg98faovwjxeTUgZAUMnRQ>