

# Mastering the game of Connect 4 through self-play

Julian Wandhoven  
fgz

June 13, 2022

## **Abstract**

Alpha Zero is an AI algorithm, that is capable of learning to play zero sum stated multiplayer games. These types of games include Go, Chess, Phi Sho and so forth. This is done by training a neural network and from data generated by a Monte Carlo Tree Search. This document also explains how neural networks work and a short explanation of the infrastructure around the AI to allow for playing on remote devices. [?][?]

# Contents

|          |                                 |           |
|----------|---------------------------------|-----------|
| <b>1</b> | <b>Methods</b>                  | <b>7</b>  |
| 1.1      | Reinforcement Learning          | 7         |
| 1.2      | Game                            | 9         |
| 1.2.1    | Game Board                      | 10        |
| 1.2.2    | Actions                         | 10        |
| 1.3      | MCTS                            | 11        |
| 1.3.1    | Structure                       | 11        |
| 1.3.2    | Quick Overview                  | 12        |
| 1.3.3    | Evaluation Basis                | 13        |
| 1.3.4    | Leaf Selection                  | 14        |
| 1.3.5    | Node Evaluation and Expansion   | 14        |
| 1.3.6    | Backfill                        | 15        |
| 1.4      | Neural Network                  | 15        |
| 1.4.1    | Introduction to Neural Networks | 15        |
| 1.4.1.1  | Fully Connected Layer           | 15        |
| 1.4.1.2  | Convolutional Layer             | 16        |
| 1.4.1.3  | Activation Function             | 17        |
| 1.4.1.4  | Training                        | 21        |
| 1.4.2    | Network used by AlphaZero       | 22        |
| 1.4.2.1  | Neural Network input            | 22        |
| 1.4.2.2  | Neural Network Architecture     | 23        |
| 1.4.2.3  | Training                        | 25        |
| 1.5      | Data generation                 | 25        |
| 1.5.1    | Action selection                | 25        |
| 1.5.2    | Memory                          | 26        |
| 1.5.2.1  | Memory update                   | 26        |
| 1.5.2.2  | Model Training                  | 26        |
| 1.6      | Model evaluation                | 26        |
| <b>2</b> | <b>Evaluation</b>               | <b>27</b> |
| 2.1      | Elo-rating                      | 27        |
| 2.1.1    | Relativity of the Elo-rating    | 29        |
| 2.2      | Elo results                     | 29        |

|          |                                    |           |
|----------|------------------------------------|-----------|
| <b>3</b> | <b>Servers and Clients</b>         | <b>31</b> |
| 3.1      | The Web Server . . . . .           | 31        |
| 3.2      | The Data Server . . . . .          | 31        |
| 3.3      | The AI Server . . . . .            | 33        |
| 3.3.1    | State Transmission . . . . .       | 34        |
| 3.3.2    | Action Selection . . . . .         | 35        |
| 3.3.3    | Action Transmission . . . . .      | 35        |
| 3.4      | Clients . . . . .                  | 35        |
| 3.4.1    | Desktop Client . . . . .           | 35        |
| 3.4.2    | iOS Client . . . . .               | 36        |
| <b>4</b> | <b>Further definitions</b>         | <b>36</b> |
| 4.1      | Set Exclusion . . . . .            | 36        |
| 4.2      | Hadamard product . . . . .         | 36        |
| 4.3      | Inner Product . . . . .            | 37        |
| 4.3.1    | Matrices . . . . .                 | 37        |
| 4.3.2    | $n$ -dimensional Tensors . . . . . | 37        |
| 4.4      | Submatrix . . . . .                | 37        |
| 4.5      | Vectorization . . . . .            | 38        |
| 4.5.1    | Tensors . . . . .                  | 38        |
| 4.6      | Vector Concatination . . . . .     | 39        |
| <b>5</b> | <b>Code</b>                        | <b>40</b> |
| 5.1      | AlphaZeroPytorch . . . . .         | 40        |
| 5.1.1    | AlphaZeroPytorch.h . . . . .       | 40        |
| 5.1.2    | AlphaZeroPytorch.cpp . . . . .     | 41        |
| 5.1.3    | cmake.sh . . . . .                 | 43        |
| 5.1.4    | CMakeLists.txt . . . . .           | 44        |
| 5.1.5    | convertToJceFormat.cpp . . . . .   | 48        |
| 5.1.6    | doEloRaiting.cpp . . . . .         | 49        |
| 5.1.7    | makeFiles.hpp . . . . .            | 51        |
| 5.1.8    | Replay.cpp . . . . .               | 52        |
| 5.1.9    | runServer.cpp . . . . .            | 53        |
| 5.1.10   | showLoss.cpp . . . . .             | 54        |
| 5.1.11   | test.cpp . . . . .                 | 55        |
| 5.1.12   | include . . . . .                  | 56        |
| 5.1.12.1 | config.hpp . . . . .               | 56        |
| 5.1.12.2 | config.cpp . . . . .               | 59        |

|             |                       |     |
|-------------|-----------------------|-----|
| 5.1.12.3    | io.hpp                | 60  |
| 5.1.12.4    | log.hpp               | 64  |
| 5.1.12.5    | log.cpp               | 70  |
| 5.1.12.6    | timer.hpp             | 73  |
| 5.1.12.7    | ai                    | 74  |
| 5.1.12.7.1  | agent.hpp             | 74  |
| 5.1.12.7.2  | agent.cpp             | 80  |
| 5.1.12.7.3  | MCTS.hpp              | 82  |
| 5.1.12.7.4  | MCTS.cpp              | 87  |
| 5.1.12.7.5  | memory.hpp            | 89  |
| 5.1.12.7.6  | memory.cpp            | 94  |
| 5.1.12.7.7  | model.hpp             | 96  |
| 5.1.12.7.8  | modelSynchronizer.hpp | 113 |
| 5.1.12.7.9  | modelWorker.hpp       | 119 |
| 5.1.12.7.10 | modelWorker.cpp       | 120 |
| 5.1.12.7.11 | playGame.hpp          | 121 |
| 5.1.12.7.12 | playGame.cpp          | 123 |
| 5.1.12.7.13 | utils.hpp             | 130 |
| 5.1.12.8    | game                  | 132 |
| 5.1.12.8.1  | game.hpp              | 132 |
| 5.1.12.8.2  | game.cpp              | 139 |
| 5.1.12.9    | jce                   | 147 |
| 5.1.12.9.1  | load.hpp              | 147 |
| 5.1.12.9.2  | save.hpp              | 153 |
| 5.1.12.9.3  | string.hpp            | 158 |
| 5.1.12.9.4  | vector.hpp            | 159 |
| 5.1.12.10   | server                | 160 |
| 5.1.12.10.1 | eloClient.hpp         | 160 |
| 5.1.12.10.2 | server.hpp            | 163 |
| 5.1.12.10.3 | server.cpp            | 165 |
| 5.1.12.11   | test                  | 169 |
| 5.1.12.11.1 | testSuit.hpp          | 169 |
| 5.1.12.11.2 | testSuit.cpp          | 171 |
| 5.1.12.11.3 | testUtils.hpp         | 176 |
| 5.2         | Clients               | 177 |
| 5.2.1       | ConsoleClient         | 177 |
| 5.2.1.1     | ConsoleClient.h       | 177 |
| 5.2.1.2     | ConsoleClient.cpp     | 178 |

|           |                        |            |
|-----------|------------------------|------------|
| 5.2.1.3   | include                | 179        |
| 5.2.1.3.1 | agent.hpp              | 179        |
| 5.2.1.3.2 | game.hpp               | 183        |
| 5.2.1.3.3 | modifications.hpp      | 189        |
| 5.2.1.4   | scr                    | 190        |
| 5.2.1.4.1 | game.cpp               | 190        |
| 5.2.2     | iosClient              | 198        |
| 5.2.2.1   | caller.py              | 198        |
| 5.2.2.2   | connect4IOS.py         | 199        |
| 5.2.3     | pyClient               | 208        |
| 5.2.3.1   | Client.py              | 208        |
| 5.2.3.2   | game.py                | 211        |
| 5.2.3.3   | gameSaver.py           | 215        |
| 5.2.3.4   | GUI.py                 | 218        |
| 5.2.3.5   | main.py                | 224        |
| 5.2.3.6   | test.py                | 226        |
| 5.2.3.7   | winStates.json         | 227        |
| 5.3       | elo                    | 230        |
| 5.3.1     | agent.py               | 230        |
| 5.3.2     | renderElo.py           | 232        |
| 5.3.3     | score.py               | 233        |
| 5.3.4     | server.py              | 234        |
| 5.4       | game                   | 238        |
| 5.4.1     | connect4               | 238        |
| 5.4.1.1   | config.hpp             | 238        |
| 5.4.1.2   | game.hpp               | 241        |
| 5.4.1.3   | game.cpp               | 248        |
| 5.4.2     | othello                | 256        |
| 5.4.2.1   | config.hpp             | 256        |
| 5.4.2.2   | game.hpp               | 259        |
| 5.4.2.3   | game.cpp               | 264        |
| 5.5       | CMakeLists.txt         | 274        |
| 5.6       | README.md              | 275        |
| 5.7       | serch.sh               | 276        |
| <b>6</b>  | <b>Demos</b>           | <b>277</b> |
| 6.1       | Matura-AlphaZero-demos | 277        |
| 6.1.1     | CMakeLists.txt         | 277        |

|       |                                 |     |
|-------|---------------------------------|-----|
| 6.1.2 | main.cpp . . . . .              | 280 |
| 6.1.3 | README.md . . . . .             | 284 |
| 6.1.4 | supervised.cpp . . . . .        | 285 |
| 6.1.5 | include . . . . .               | 289 |
|       | 6.1.5.1 point.hpp . . . . .     | 289 |
|       | 6.1.5.2 utils.hpp . . . . .     | 290 |
| 6.1.6 | src . . . . .                   | 291 |
|       | 6.1.6.1 point.cpp . . . . .     | 291 |
| 6.1.7 | supervised . . . . .            | 294 |
|       | 6.1.7.1 include . . . . .       | 294 |
|       | 6.1.7.1.1 config.hpp . . . . .  | 294 |
|       | 6.1.7.1.2 model.hpp . . . . .   | 295 |
|       | 6.1.7.2 src . . . . .           | 296 |
|       | 6.1.7.2.1 model.cpp . . . . .   | 296 |
| 6.1.8 | unsupervised . . . . .          | 299 |
|       | 6.1.8.1 include . . . . .       | 299 |
|       | 6.1.8.1.1 cluster.hpp . . . . . | 299 |
|       | 6.1.8.1.2 config.hpp . . . . .  | 300 |
|       | 6.1.8.1.3 group.hpp . . . . .   | 301 |
|       | 6.1.8.2 src . . . . .           | 302 |
|       | 6.1.8.2.1 cluster.cpp . . . . . | 302 |
|       | 6.1.8.2.2 group.cpp . . . . .   | 304 |

Alpha Zero is an algorithm published in 2018 by Google Deepmind as the generalization of AlphaGo Zero, an algorithm that learned to play the game of Go using only the rules of the game. In the generalized version, the same principles were applied to Chess and Shogi. Unlike previous algorithms such as StockFish or Elmo that use hand-crafted evaluation functions along with alpha-beta searches over a large search space, Alpha Zero uses no human knowledge. Rather, it generates all information through self-play. It has been shown to achieve superhuman performance in Chess, Shogi and Go. In this project the AI will be trained to play connect4. The entire algorithm is implemented in C++ to increase efficiency during the Monte Carlo Tree Search (MCTS). Furthermore, to allow for better performance when playing, all computations are handled via a server. A secondary server has also been added to allow for easy re-routing of the main server and handle things like elo-ratings (see section 2.1 on page 27) for all agents. This server only handles routing and data storage while the other one handles the AI. Additionally, I have added a short introduction on how neural networks work and how they are trained in section 1.4 on page 15.

## 1 Methods

The Alpha Zero algorithm is a reinforcement learning algorithm using two major parts: a) a *Monte Carlo tree search* (MCTS) that is guided by b) the *neural network* to improve performance. The agent (computer player) runs a certain amount of simulation games using its MCTS and neural network. At each step, the MCTS evaluates the most promising next states as given by the neural network's estimation. The MCTS, by simulating games starting from the current state, will improve the neural network's prediction for that state. At the end of each game, the winner is determined and used to update the neural network's estimation of who would win a game starting from a certain state.

### 1.1 Reinforcement Learning

When training neural networks, there are three major possible situations: Supervised learning, unsupervised learning, and reinforcement learning. The first uses predetermined data with known in- and outputs the network is trained to predict. An example of supervised learning is the recognition

of handwriting as the data is defined by humans. This method consists of creating a large database of examples, and the neural network is then trained to predict a given output for all examples.

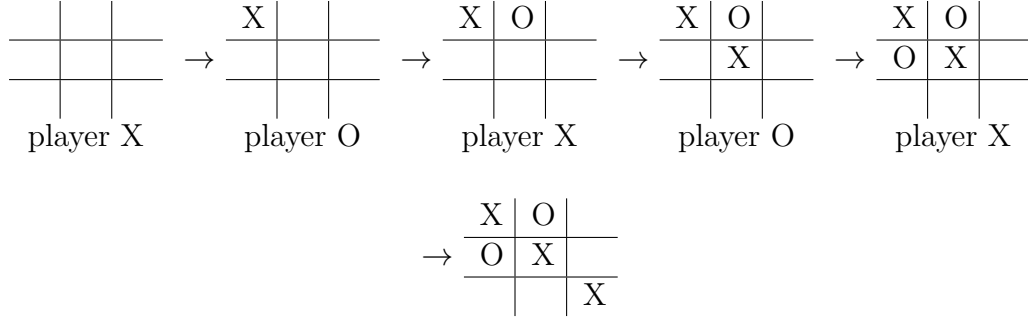
Unsupervised learning or self-organization is used when there is no previous available data and the neural network has to create those classifications itself. An example of unsupervised learning is Vector quantization. The algorithm sorts points in  $n$ -dimensional space into a predetermined amount of groups. Every group is defined by its centroid point. Training happens by selecting a sample point at random, and moving the closest centroid point towards the sample point by a fraction of the distance between them. The sample point is selected from the input data [?]. An example of both supervised and unsupervised learning can be seen in the demo<sup>1</sup>.

These two methods represent the extreme ends of the spectrum. Reinforcement learning on the other hand can be thought of as an intermediate form. It uses a predetermined environment which gives positive, neutral and negative feedback. The neural network is then discouraged from taking actions leading to negative feedback and encouraged to take actions leading to positive feedback. The feedback is determined by the environment the agent learns to interact with. In this case, losing a game would be bad and result in negative feedback whereas winning a game leads to positive feedback. Ties lead to neutral feedback. The agent's learning is set up in such a way, that it is encouraged to take actions leading to positive feedback and discouraged from taking actions that lead to negative feedback. However actions, can lead to a loss that only occurs many game steps in the future. A common approach to solve this problem is to have the feedback propagate backwards to previous actions. In Alpha Zero, this is handled by the memory (see section 1.5.2 on page 26). When the game reaches an end state and a winner is determined, the feedback is propagated backwards up the game. If the player won, the feedback is positive. If he lost, it is negative. More specifically, if a player takes an action  $a_s$  at a state  $s$ , that leads to a win, the reward for that state is defined as  $R(s, a_s) = 1$ . On the other hand, if the action leads to a loss, the reward will be  $R(s, a_s) = -1$ . If the game ends in a tie, the reward is  $R(s, a_s) = 0$ . Let  $g$  be the set of states traversed in any given game. Every agent  $p$  will try to maximize  $\sum_{s \in g \cap \mathbb{P}} R(s, a_s)$ . Where  $\mathbb{P}$  is the set of all state player  $p$  can traverse. Let's look at a tic tac toe example of the following game:

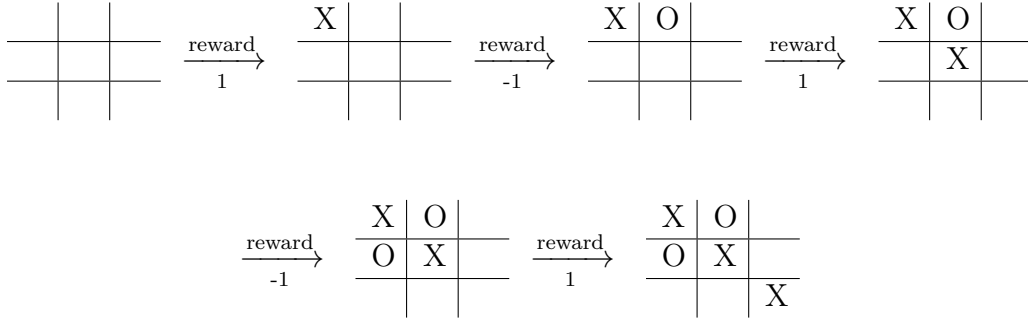
---

<sup>1</sup>demo is at <https://github.com/JulianWww/Matura-AlphaZero-demos>





Since player  $X$  won the game, the reward for every state  $s \in g \cap \mathbb{X}$  is  $R(s, a_s) = 1$  and the reward for every state  $s \in g \cap \mathbb{O}$  is  $R(s, a_s) = -1$ . Where  $\mathbb{X}$  is the set of all state with player  $X$  at turn and  $\mathbb{O}$  for player  $O$ . The reward for the entire game is:



The important thing to keep in mind is that reinforcement learning algorithms encourage actions that lead to a positive feedback and discourage actions that lead to a negative feedback.

## 1.2 Game

The game is the environment, that is used to train the AI. The game consists of constant unchanging game states. Every game state consists of a game board and a player. An end game state is a state at which the game is done. This means that one player won or the game ended in a tie. For connect4 this means four stones in a line or a full game board. Let  $\mathbb{G}$  be the set of all legal game states. Let  $\mathbb{G}_{done}$  be the set of all game states for which the game is done.

### 1.2.1 Game Board

Board games consist of placing stones of different types on a board with a certain amount of fields. Many games, like Go, Chess and Connect4, arrange their fields in a rectangular pattern. These games have two distinct stones. We can represent these game boards as stack of binary layers. Every layer is associated with one kind of stone. Each layer contains a one, where the board has a stone of the appropriate type and zeros everywhere else. For instance, the following tic tac toe game board can be represented by the following binary plane stack.

$$\begin{array}{|c|c|c|} \hline & X & O \\ \hline O & X & \\ \hline O & & X \\ \hline \end{array} \rightarrow \left[ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \right]$$

Internally, the game board is represented by a flat vector. The conversion from a game state  $s \in \mathbb{G}$  to vector is defined as  $vec(T_s(s))$ . Where  $T_s(s) : \mathbb{G} \rightarrow \mathbb{R}^{\dots}$  is the board's 3-dimensional board tensor. The  $vec$  function is defined in section 4.5.1 on page 38. This operation for the tic tac toe board from before would look like this:

$$vec \left( T_s \left( \begin{array}{|c|c|c|} \hline & X & O \\ \hline O & X & \\ \hline O & & X \\ \hline \end{array} \right) \right) = [010010001001100100]^T$$

### 1.2.2 Actions

Actions are numbers used to identify changes to the game. Every game has a set of all possible actions  $\mathbb{A}_{possible} \subset \mathbb{N}_0$ . This set represents the sum of actions a certain player can take at any valid game state. In connect4 there is no need to distinguish between colors as the action represents where to place the next stone. The color is determined by which player is at turn when the action is taken. In connect4, the set of all possible actions for the current player is  $\mathbb{A}_{possible} = [0, 41]$ . There is no need to have actions for the player, that is not at turn as these will never be taken. Every number is associated with a position on the game board. The mapping  $a$  to game fields is the following:

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  |
| 7  | 8  | 9  | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| 35 | 36 | 37 | 38 | 39 | 40 | 41 |

Let  $\mathbb{A}(s)$  be the set of all legal actions for a given state  $s \in \mathbb{G}$ . For all states  $s_{done} \in \mathbb{G}_{done}$  the set of all legal actions  $\mathbb{A}(s_{done})$  is the empty set. The function  $\mathcal{A} : \mathbb{G} \times \mathbb{A} \rightarrow \mathbb{G}$  is used to get from one game state to another by taking an action. Where  $\mathbb{A}$  is the set of all legal actions the chosen game state. If we were to map action to position for tick tack toe as follows and that the game board is the following:

|   |   |   |   |   |  |
|---|---|---|---|---|--|
| 0 | 1 | 2 | X | O |  |
| 3 | 4 | 5 |   |   |  |
| 6 | 7 | 8 |   |   |  |

State  $s$

In this example player  $X$  is allowed to place a stone in any empty field  $\mathbb{A}(s) = \{2, 3, 4, 5, 6, 7, 8\}$ . Therefore  $\mathcal{A}(s, a)$  is valid if  $a \in \mathbb{A}(s)$  and otherwise invalid. Therefore  $\mathcal{A}(s, 0)$  and  $\mathcal{A}(s, 1)$  are invalid while  $\mathcal{A}(s, 2), \dots, \mathcal{A}(s, 8)$  are valid.

### 1.3 MCTS

A Monte Carlo tree search (MCTS) is a tree search algorithm that can be used to find sequences of actions leading to a desirable outcome. This is done by procedurally generating a directed graph of possible successor states to the current state or root state.

#### 1.3.1 Structure

This graph consists of nodes that are used to simulate possible sequences of states starting from a root state and edges that connect nodes. The set of all possible nodes  $\mathbb{M}_{possible} = \{\mathcal{N}(s) \mid s \in \mathbb{G}\}$ , where  $\mathcal{N} : \mathbb{G} \rightarrow \mathbb{M}_{possible}$  is a bijective function that maps a game state to a node.  $\mathcal{S} : \mathbb{M}_{possible} \rightarrow \mathbb{G} = \mathcal{N}^{-1}$  is the inverse of  $\mathcal{N}$ . The set of all allowed actions for a certain

node  $n \in \mathbb{M}_{possible}$  is  $\mathbb{A}(n) = \mathbb{A}(\mathcal{S}(n))$ . The set of all nodes in an MCTS at any given time is  $\mathbb{M} \subseteq \mathbb{M}_{possible}$ . Every node  $n \in \mathbb{M}$  has a set of edges  $\mathbb{E}(n)$  that connect it to other nodes, the set of all possible edges is  $\mathbb{E}_{possible}$ .  $\mathcal{E} : \mathbb{M} \times \mathbb{A}_{possible} \rightarrow \mathbb{E}_{possible}$  is a bijective function used to map nodes and actions to edges. Furthermore for  $\mathcal{E}(n, a)$  to be valid  $a$  must be an element of  $\mathbb{A}(n)$ . The function  $\mathcal{N}_{to} : \mathbb{E} \rightarrow \mathbb{M}$  maps an edge to the node it is pointing to while  $\mathcal{N}_{from} : \mathbb{E} \rightarrow \mathbb{M}$  is used to find the node an edge is pointing from. There are, however, two different kinds of nodes, that can be distinguished by the set of their edges. The first are the expanded nodes  $\mathbb{M}_{expanded} \subseteq \mathbb{M}$ . For expanded nodes  $n \in \mathbb{M}_{expanded}$  the set of edges  $\mathbb{E}(n) = \{\mathcal{E}(n, a) \mid a \in \mathbb{A}(n)\} \neq \emptyset$ . The second category are the leaf nodes  $\mathbb{M}_{leaf} \subseteq \mathbb{M}$ . Leaf nodes are unexpanded nodes. This means that they do not yet have any connections to other nodes. Thus  $\mathbb{E}(n) = \emptyset$  for all  $n \in \mathbb{M}_{leaf}$ .

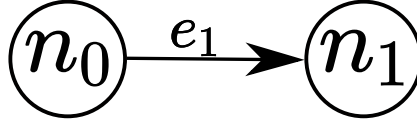


Figure 1: A vary simple MCST example

Consider the example MCST in fig 1 on page 12. The set  $\mathbb{M}$  of every node in the MCTS is  $\mathbb{M} = \{n_1, n_2\}$ . Node  $n_0$  has one edge  $e_1$  pointing to  $n_1$ . Therefor  $\mathbb{E}(n_0) = \{e_1\}$  and because  $n_1$  has no edges leaving it,  $\mathbb{E}(n_1) = \emptyset$ . This means that  $\mathbb{M}_{expanded} = \{n_0\}$  and  $\mathbb{M}_{leaf} = \{n_1\}$ . Furthermore  $\mathcal{N}_{to}(e_1) = n_1$  and  $\mathcal{N}_{from}(e_1) = n_0$ . If we assume that  $e_1$  is associated with action  $a_e \in \mathbb{A}(n_0)$ , than  $\mathcal{E}(n_0, a_e) = e_1$

### 1.3.2 Quick Overview

MCTS simulations consist of four phases: selection, evaluation, expansion, and back propagation. Every simulation starts from the MCTS's root node  $n_0 \in \mathbb{M}$ . The selection phase of the MCTS is used to select a leaf node  $n_L \in \mathbb{M}_{leaf}$ . This is done by using a selection function  $\sigma : \mathbb{M} \rightarrow \mathbb{M}$  defined in equation 4 on page 14. This function is used to find node  $n_{t+1}$  from node  $n_t$  for  $t \in [0, L[$ .  $n_{t+1}$  is selected by  $\sigma$ .

$$\sigma(n_t) = n_{t+1} \tag{1}$$

This means that  $n_1 = \sigma(n_0)$ ,  $n_2 = \sigma(n_1)$  .... When a leaf node is found, that leaf node is evaluated. In Alpha Zero this is achieved using the neural

network described in section 1.4. The neural network evaluates the leaf node using its evaluation function  $\theta : \mathbb{M} \rightarrow \mathbb{R} \times \mathbb{R}^{|\mathbb{A}_{possible}|}$ . The neural network's first output is an estimation of the expected reward. The neural network's second output is the action policy  $p \in \mathbb{R}^{|\mathbb{A}_{possible}|}$  that represents the advantageousness of the actions in  $\mathbb{A}_{possible}$ .  $n_L$  is now expanded. Expansion works by creating nodes  $\mathbb{M}_{new} = \{\mathcal{N}(\mathcal{A}(\mathcal{S}(n_L), a)) | a \in \mathbb{A}(n_L)\}$  unless these nodes already exist and then redefining  $\mathbb{M} = \mathbb{M} \cup \mathbb{M}_{new}$ . Furthermore,  $n_L$ 's edges are redefined as  $\mathbb{E}(n_L) = \{\mathcal{E}(n_L, a) | a \in \mathbb{A}(n_L)\}$ . By definition  $\mathcal{N}_{to}(\mathcal{E}(n_L, a)) = \mathcal{N}(\mathcal{A}(\mathcal{S}(n_L), a))$  for  $a \in \mathbb{A}(n_L)$ . This leads to  $n_L$  being moved from  $\mathbb{M}_{leaf}$  to  $\mathbb{M}_{expanded}$ . During back propagation the expected reward of  $n_L$  is used to update the expected reward of all nodes traversed during selection. This effectively improves the estimation of the expected reward. This four step simulation is carried out a certain amount of times. The estimation error for all estimations of the expected reward will converge to 0 as the amount of simulations increases.

### 1.3.3 Evaluation Basis

The MCTS's goal is to find good estimations of the reward for a certain action at a certain state. This reward estimation is  $Q : \mathbb{E}_{possible} \rightarrow \mathbb{R}$ . To define  $Q$ , the functions  $W : \mathbb{E} \rightarrow \mathbb{R}$  and  $N : \mathbb{E} \rightarrow \mathbb{N}_0$  are required.  $N(e)$  is the amount of times an edge  $e \in \mathbb{E}_{possible}$  has been traversed. This means how many times  $\sigma$  has chosen to follow the edge  $e$  to a new node.  $W(e)$  is the sum of the reward computations from all  $N(e)$  times the edge has been evaluated. Therefore the expected reward  $Q$  is defined as:

$$Q(e) = \frac{W(e)}{N(e)} \quad (2)$$

The fourth and last of these functions is  $P : \mathbb{E} \rightarrow \mathbb{R}$ .  $P$  is the policy function, it's the neural network's preliminary estimation of:

$$P(e) \approx \frac{N(e)}{\sum_{i \in \mathbb{E}(\mathcal{N}_{from}(e))} N(i)} \quad (3)$$

This function is used to guide the search to more promising edges before a lot of time is spent on simulation.

### 1.3.4 Leaf Selection

MCTS's evaluation starts by simulating future moves within the tree. This is done by selecting an edge and then following that edge to a new node. From there, the next edge and node are selected. This is repeated until a leaf node is reached. To select an edge and thus a node from the current node  $n \in \mathbb{M}$  the function  $\sigma$  is used. To define  $\sigma$  we must first define the edge evaluation function  $v : \mathbb{E} \rightarrow \mathbb{R}$ .  $v$  is defined as follows:

$$v(e) = Q(e) + c_{puct}P(e) \cdot \frac{\sqrt{\sum_{b \in \mathbb{E}(\mathcal{N}_{from}(e))} N(b)}}{1 + N(e)} \quad (4)$$

Where  $c_{puct} \in \mathbb{R}^+$  is the exploration constant used to define how important exploration is. The smaller  $c_{puct}$  is, more important  $Q$  and less important exploration and  $P$ .  $\sigma$ , for a given node  $n \in \mathbb{M}$ , is then defined as:

$$\sigma(n) = \mathcal{N}_{to}(\text{argmax}(\mathbb{E}(n))) \quad (5)$$

$\text{argmax}$  returns the edge  $e$  with the largest  $v(e)$ .  $\sigma$  is run, until its output is a leaf-node  $N_L \in \mathbb{M}_{leaf}$ .

### 1.3.5 Node Evaluation and Expansion

When a leaf node  $n_L \in \mathbb{M}_{leaf}$  is reached, that is not an end game node  $\mathcal{S}(n_L) \notin \mathbb{G}_{done}$ , the node is passed to the neural network. The neural network (see section 1.4) is used to predict the node's policy  $p$  and its value  $v$ .  $\theta(n_L) = (v, p)$ .  $p$  is used to create new nodes  $\mathbb{M}_{new} = \{\mathcal{N}(\mathcal{A}(\mathcal{S}(n_L), a)) : a \in \mathbb{A}(n_L)\}$ . The initial function outputs of the three edge functions for the edges  $e \in \mathbb{E}(n_L)$ , connecting  $n_L$  to  $\mathbb{M}_{new}$ , are.

$$N(e) = 0$$

$$W(e) = 0$$

$$P(e) = \pi(e)$$

$\pi(e)$  is the of the policy of the edge. It's defined as:

$$\pi(\mathcal{E}(n_L, a)) = p_{a+1}$$

where  $a$  is the edges action  $a \in \mathbb{A}(n_L)$ .  $\mathbb{M}$  and  $\mathbb{E}(n_L)$  are then redefined as  $\mathbb{M} = \mathbb{M} \cup \mathbb{M}_{new}$  and  $\mathbb{E}(n_L) = \{\mathcal{E}(n_L, a) \mid a \in \mathbb{A}(n_L)\}$ .  $M_{expanded}$  and  $M_{leaf}$  also change according to there definition.

### 1.3.6 Backfill

The value  $v$  is used to update the reward prediction for all nodes  $n_{[0,L]}$  traversed during the edge selection. Assuming that  $\rho : \mathbb{E}_{possible} \rightarrow \{1, -1\}$  is the player taking an action at an edge  $e \in \mathbb{E}_{possible}$ ,  $W$  and  $N$  are updated as follows for all nodes  $n_t$  with  $t \in [0, L]$ :

$$\begin{aligned} N(n_t) + 1 &\Rightarrow N(n_t) \\ W(n_t) + v \cdot \rho(n_t) \cdot \rho(n_L) &\Rightarrow W(n_t) \end{aligned}$$

See fig 10 on page 305 for example Simulation.

## 1.4 Neural Network

Search algorithms like MCTS are able to find advantageous action sequences. In game engines, the search algorithm is improved by using evaluation functions. These functions are generally created using human master knowledge. In the Alpha Zero algorithm, this evaluation function is a biheaded deep convolutional neural network trained by information gathered from the MCTS. In order to understand the training process, one must first understand how the neural network functions.

### 1.4.1 Introduction to Neural Networks

An artificial neural network or just neural network is a mathematical function inspired by biological brains. Although there are many types of neural networks, the only relevant one to this work is the feed forward network. These models consist of multiple linear computational layers separated by non-linear activation functions. Every layer takes the outputs of the previous layer, and applies a linear transformation to it [?]. There are many different feed-forward neural network layers and activation functions to chose from when designing a neural network. To focus this explanation, only the relevant ones will be discussed along with the back-propagation algorithm.

#### 1.4.1.1 Fully Connected Layer

A fully connected layer is the most basic layer. It applies a simple matrix multiplication. The layer takes a  $1 \times n$  dimensional matrix  $x \in \mathbb{R}^{1 \times n}$  as an

input and multiplies it by a weight matrix  $w \in \mathbb{R}^{n \times m}$ . This operation outputs a  $1 \times m$  dimensional matrix to which a bias  $b \in \mathbb{R}^{1 \times m}$  is added to form the output matrix  $v \in \mathbb{R}^{1 \times m}$  containing the output values of the layer.  $v$  is then fed to the next layer. The addition of the bias vector  $b$  is optional. In some situations it is worth dropping the bias in favour of computational speed. The fully connected layer forward propagation function shall be defined as  $\delta_{wb} : \mathbb{R}^n \rightarrow \mathbb{R}^m$

$$\delta_{wb}(x) = w \cdot x + b \quad (6)$$

#### 1.4.1.2 Convolutional Layer

Convolutional layers are commonly used for image processing. They perform the same operations over the entire image searching for certain patterns. In order to achieve this, a set of kernels  $\mathbb{K}$ , of size  $m \times n$ , are defined for the layer. Kernels are similar to fully connected layers. They consist of a weight tensor  $w \in \mathbb{R}^{m \times n \times l}$  and an optional bias scalar  $b \in \mathbb{R}$ . For every kernel  $k \in \mathbb{K}$ , the kernel's forward operation  $\xi_k : \mathbb{R}^{m \times n \times l} \rightarrow \mathbb{R}$  is defined as:

$$\xi_k(i) = \langle w_k, i \rangle_I + b \quad (7)$$

where  $\langle \rangle_I$  is the Tensor inner product defined in equation 42 on page 37. The convolutional operation  $\Lambda : \mathbb{R}^{i \times j \times l} \rightarrow \mathbb{R}^{i-m+1 \times j-n+1 \times |\mathbb{K}|}$  is an element wise operation. Given that  $I \in \mathbb{R}^{i \times j \times l}$  is the layer input, every element of  $\Lambda(I)_{abc}$  with  $a \in [1, i-m+1]$ ,  $b \in [1, j-n+1]$  and  $c \in [1, |\mathbb{K}|]$  is defined as:

$$\Lambda(I)_{abc} = \xi_{k_c}(I[[a, a+m[, [b, b+n[, [1, |\mathbb{K}|]]) \quad (8)$$

The submatrix indexing operation  $I[...]$  is defined in section 4.4 on page 37. For example given the following input tensor  $I \in \mathbb{R}^{4 \times 4 \times 1}$ :

$$I = \begin{bmatrix} [3] & [0] & [1] & [5] \\ [2] & [6] & [2] & [4] \\ [2] & [4] & [1] & [0] \\ [3] & [0] & [1] & [5] \end{bmatrix}$$



and the following kernel weight matrix  $w_k \in \mathbb{R}^{3 \times 3 \times 1}$  along with the scalar  $b \in \mathbb{R}$ ,

$$w_k = \begin{bmatrix} [-1] & [0] & [1] \\ [-2] & [0] & [2] \\ [-1] & [0] & [1] \end{bmatrix}$$

$$b = 7$$

there are four possible locations in which  $w_k$  can be placed within  $I$ . As there is only one kernel, the length of the set of all kernels  $|\mathbb{K}| = 1$ . This also means that  $\Lambda(I) \in \mathbb{R}^{2 \times 2 \times 1}$ . To calculate  $\Lambda(I)_{111}$ , we compute the kernel operation  $\xi_k(I[[1, 3], [1, 3], \{1\}])$

$$\Lambda_{111} \left( \begin{bmatrix} [3] & [0] & [1] & [5] \\ [2] & [6] & [2] & [4] \\ [2] & [4] & [1] & [0] \\ [3] & [0] & [1] & [5] \end{bmatrix} \right) = \begin{bmatrix} [3] & [0] & [1] \\ [2] & [6] & [2] \\ [2] & [4] & [1] \end{bmatrix} \circ \begin{bmatrix} [-1] & [0] & [1] \\ [-2] & [0] & [2] \\ [-1] & [0] & [1] \end{bmatrix} + 7 \quad (9)$$

$$= -1 \cdot 3 + 0 \cdot 0 + 1 \cdot 1 - 2 \cdot 2 + 0 \cdot 6 + 2 \cdot 2 - 1 \cdot 2 + 0 \cdot 4 + 1 \cdot 1 + 7$$

$$= 4$$

The same is done for  $\Lambda(I)_{121}$ ,  $\Lambda(I)_{211}$  and  $\Lambda(I)_{221}$ . This leads to a  $\Lambda(I)$  of:

$$\Lambda(I) = \begin{bmatrix} [4] & [3] \\ [4] & [2] \end{bmatrix}$$

#### 1.4.1.3 Activation Function

All neural network layers are linear functions. Thus, given two activation functions  $f_1(x) = ax + b$  and  $f_2(x) = cx + d$ , the chained function  $f(x) = f_1(f_2(x))$  is also linear because:

$$f(x) = f_1(f_2(x)) = a(cx + d) + b = acx + ad + b = ex + g \quad (10)$$

where  $a, b, c, d, e$  and  $g \in \mathbb{R}$ . In order to represent non linear functions, a non linear activation function  $f_a$  is added between two neural network layers. Thus,  $f(x)$  becomes  $f(x) = f_1(f_a(f_2(x)))$ . In this neural network, three different activation functions are used: *tanh*, *softmax*, and *LeakyReLU*. These functions are defined as follows:

**tanh:**

$\mathbb{R} \rightarrow \mathbb{R}$

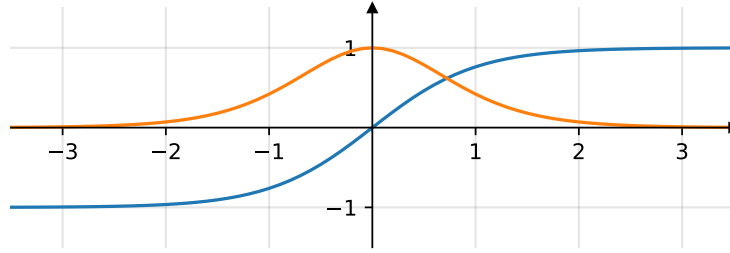


Figure 2: tanh function in blue and the tanh's derivative is in orange

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (11)$$

$$\frac{d}{dx} \tanh(x) = \text{sech}(x)^2 \quad (12)$$

**softmax:**

$\mathbb{R}^n \rightarrow \mathbb{R}^n$

For a given input vector  $v \in \mathbb{R}^n$ . The output vector  $o \in \mathbb{R}^n$  at every position  $i \in [1, n]$  is:

$$o_i = \text{softmax}(v)_i = \frac{e^{v_i}}{\sum_{j \in v} e^j} \quad (13)$$

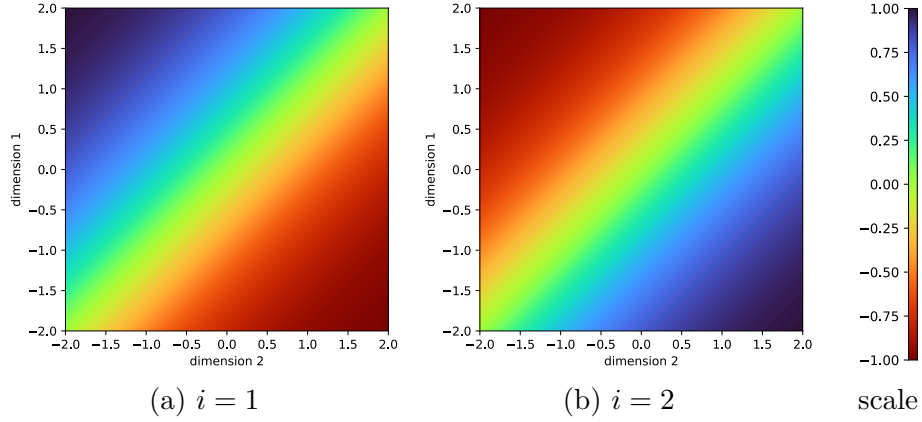


Figure 3: Graph of the softmax function from  $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ .  $i$  is the index of the output dimension. Therefore,  $i = 1$  refers to the output's first dimension and  $i = 2$  refers to it's second dimension.

Because the function's in- and outputs are  $n$  dimensional vectors, the derivative is an  $n \times n$  dimensional matrix. When taking its derivative,  $\frac{d}{dv_j} \text{softmax}(v)_i$ , there are two possible cases.

case  $j = i$ :

$$\begin{aligned}
 \frac{d}{dv_j} \left[ \frac{e^{v_i}}{\sum_{b \in v} e^b} \right] &= \frac{\sum_{b \in v} e^b \cdot e^{v_i} - e^{v_i} \cdot e^{v_j}}{(\sum_{b \in v} e^b)^2} \\
 &= \frac{e^{v_i}}{\sum_{b \in v} e^b} \cdot \frac{(\sum_{b \in v} e^b - e^{v_j})}{\sum_{b \in v} e^b} \\
 &= \text{softmax}(v)_i \cdot (1 - \text{softmax}(v)_j)
 \end{aligned}$$

case  $j \neq i$ :

$$\begin{aligned}
 \frac{d}{dv_j} \left[ \frac{e^{v_i}}{\sum_{b \in v} e^b} \right] &= -\frac{e^{v_j} \cdot e^{v_i}}{(\sum_{b \in v} e^b)^2} \\
 &= -\text{softmax}(v)_i \cdot \text{softmax}(v)_j
 \end{aligned}$$

Therefore, the derivative of the softmax function is:

$$\text{softmax}'(v)_{ij} = \begin{cases} \text{softmax}(v)_i \cdot (1 - \text{softmax}(v)_j) & i = j \\ -\text{softmax}(v)_i \cdot \text{softmax}(v)_j & i \neq j \end{cases} \quad (14)$$

**LeakyReLU:**

$\mathbb{R} \rightarrow \mathbb{R}$

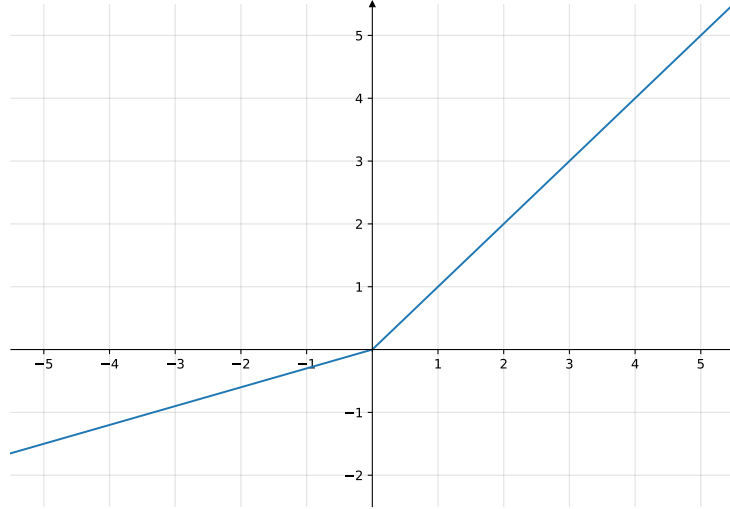


Figure 4: LeakyReLU with  $c = 0.3$

$$\text{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ x \cdot c & x < 0 \end{cases} \quad (15)$$

$$\frac{d}{dx}\text{LeakyReLU}(x) = \begin{cases} \frac{d}{dx}x & x \geq 0 \\ \frac{d}{dx}c \cdot x & x < 0 \end{cases}$$

$$\text{LeakyReLU}'(x) = \begin{cases} 1 & x > 0 \\ c & x < 0 \end{cases} \quad (16)$$

where  $c$  is a constant describing the slope of the function for negative input values. The derivative of the LeakyReLU function is undefined for  $x = 0$ . However as we will be performing gradient descent on these functions the derivative must be defined for all  $x \in \mathbb{R}$ . A possible definition that accomplishes the objective is:

$$g(x) = \begin{cases} 1 & x \geq 0 \\ c & x < 0 \end{cases} \quad (17)$$

#### 1.4.1.4 Training

Neural network training can be mathematically expressed as minimizing a loss function  $\ell$  describing how inaccurate the network is. In our case,  $\ell$  takes the neural network's predicted value vector  $Y_{pred}$  and the correct value vector  $Y_{true}$ .  $Y_{true}$  must be known before the computation begins. In AlphaZero,  $Y_{true}$  is generated by the MCTS. As with the activation, function there are many different possible loss functions. In this implementation, the mean-square-error(*mse*) loss function is used.  $mse : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$  is defined as:

$$\ell = \frac{|Y_{pred} - Y_{true}|^2}{n} \quad (18)$$

The network then performs gradient descent to find parameters that minimize  $\ell$ . To make this introduction easier, I will use a fully connected neural network. For every layer in the network, starting with the last one, it must be determined in which direction and by how much the output values  $Y_{pred_i} \in Y_{pred}$  should be “moved” to minimize  $\ell$ . This change is described by  $\Delta Y_j$ , where  $j$  is the index of the last layer. Then, the change in the inputs to the activation function  $f_a$  must be computed using the saved activation function inputs  $A$ .  $\Delta A$  will describe the change to  $A$ .

$$\Delta A = f'_a(A) \circ \Delta Y_j \quad (19)$$

The hadamard product  $\circ$  is defined in section 4.2 on page 36.

Next comes the update to the weight matrix  $w$ . Let  $\Delta w$  describe the change to  $w$  and let  $X$  be the input vector of the layer.  $\Delta w$  is than defined as:

$$\Delta w = \Delta A \cdot X^T \quad (20)$$

The layer's bias vector is updated in the direction of  $\Delta A$ :

$$\Delta b \sim \Delta A \quad (21)$$

Lastly, the change to the output of the previous layer  $\Delta Y_{j-1}$  is computed.

$$\Delta Y_{j-1} = \Delta A \cdot w^T \quad (22)$$

This process is repeated until the foremost layer of the neural network is reached. This layer has the index  $j = 0$ .

### 1.4.2 Network used by AlphaZero

The neural network in Alpha Zero is used to estimate the value  $v$  and policy  $p$  for any game state or node  $n$ .  $v$  is the neural network's estimation of the state's expected reward. The policy  $p \in \mathbb{R}^{|\mathbb{A}|}$  of a game state  $n$  represents the advantageousness of every action  $a \in \mathbb{A}$ , as estimated by the neural network.

#### 1.4.2.1 Neural Network input

The neural network input is a game state or node  $n$  represented by two 7 x 6 binary images stacked on top of each other. One image  $X$  represents the stones belonging to the current player. While the second image  $Y$  represents the stones belonging to the other player. In both images, the pixel values are one where a stone belonging to the player they represent is located and zero if the field is empty or a stone belonging to the other player is located there.  $X$  and  $Y$  are then stacked on top of each other in the third dimension to form the input tensor  $i_n = [X, Y] \in \mathbb{R}^{7 \times 6 \times 2}$ . Consider the following Connect4 board (fig 5 on page 22).

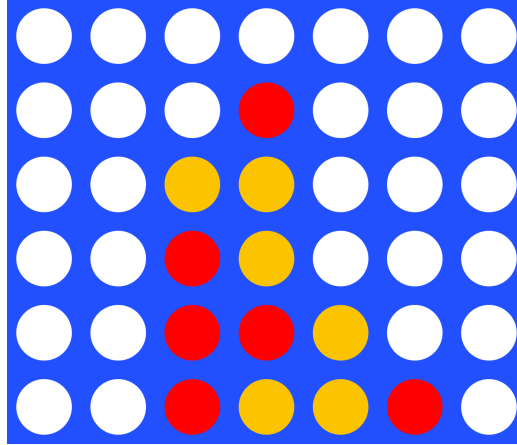


Figure 5

If red is the current player then:

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \textcolor{red}{1} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{orange}{0} & \textcolor{orange}{0} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{1} & \textcolor{orange}{0} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{orange}{0} & 0 & 0 \\ 0 & 0 & \textcolor{red}{1} & \textcolor{orange}{0} & \textcolor{orange}{0} & \textcolor{red}{1} & 0 \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \textcolor{red}{0} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{orange}{1} & \textcolor{orange}{1} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{0} & \textcolor{orange}{1} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{0} & \textcolor{red}{0} & \textcolor{orange}{1} & 0 & 0 \\ 0 & 0 & \textcolor{red}{0} & \textcolor{orange}{1} & \textcolor{orange}{1} & \textcolor{red}{0} & 0 \end{bmatrix}$$

For clarification, the numbers are coloured in the same colour as the stones at that position. After stacking  $X$  and  $Y$ ,  $i_n$  is:

$$i_n = \begin{bmatrix} [0, 0] & [0, 0] & [0, 0] & [0, 0] & [0, 0] & [0, 0] & [0, 0] \\ [0, 0] & [0, 0] & [0, 0] & \textcolor{red}{[1, 0]} & [0, 0] & [0, 0] & [0, 0] \\ [0, 0] & [0, 0] & \textcolor{orange}{[0, 1]} & \textcolor{orange}{[0, 1]} & [0, 0] & [0, 0] & [0, 0] \\ [0, 0] & [0, 0] & \textcolor{red}{[1, 0]} & \textcolor{orange}{[0, 1]} & [0, 0] & [0, 0] & [0, 0] \\ [0, 0] & [0, 0] & \textcolor{red}{[1, 0]} & \textcolor{red}{[1, 0]} & \textcolor{orange}{[0, 1]} & [0, 0] & [0, 0] \\ [0, 0] & [0, 0] & \textcolor{red}{[1, 0]} & \textcolor{orange}{[0, 1]} & \textcolor{orange}{[0, 1]} & \textcolor{red}{[1, 0]} & [0, 0] \end{bmatrix}$$

#### 1.4.2.2 Neural Network Architecture

The neural network used by Alpha Zero consists of three main sub-modules, namely the residual tower, the value head and the policy head. The residual tower's purpose is to preprocess the data for the two heads. The value head determines the value  $v$  from the output of the residual tower. While the policy head computes the policy  $p$ . The residual tower consists of a convolutional block followed by six residual blocks.

The convolutional block consists of the following:

1. A convolutional layer consisting of 75 filters with a kernel size of 3 x 3
2. Batch normalization [?]
3. A non-linear rectifier (LeakyReLU).

Every residual block consists of the following modules:

1. A convolutional layer consisting of 75 filters with a kernel size of 3 x 3
2. Batch normalization [?]
3. A non-linear rectifier (LeakyReLU)
4. A convolutional layer consisting of 75 filters with a kernel size of 3 x 3
5. Batch normalization [?]
6. Batch normalization outputs are added to the block's input.
7. A non-linear rectifier (LeakyReLU)

Outputs are then passed to the value and policy head of the network for further evaluation. The value head consists of the following modules:

1. A convolutional layer consisting of 10 filters with a kernel size of 1 x 1
2. A fully connected layer of size 210
3. A non-linear rectifier (LeakyReLU)
4. A fully connected layer of size 1
5.  $\tanh$  activation function

The policy head consists of the following modules:

1. A convolutional layer consisting of 2 filters with a kernel size of 1 x 1
2. A fully connected layer of size 1

The output of the policy head  $p_{pre}$  is then masked with the allowed actions to form  $p_{masked}$  in such a way that  $p_{masked}$  is  $-1000$  for all non-allowed actions. Finally,  $p_{masked}$  is passed through the softmax function to form  $p$ :

$$p = \text{softmax}(p_{masked}) \quad (23)$$



### 1.4.2.3 Training

Training is performed in batches of 256 states. The value head is updated using mean square error. The policy head is updated using mean square error as well. However all non-legal actions are ignored. This avoids unnecessary updating of the neural network. The value, the neural network is trained to predict for a certain MCTS node  $n$ , is equivalent to 1 if the player who took an action at node  $n$  did win,  $-1$  if that player did lose and 0 if the game ended in a tie. The policy  $p_{a_l}$  to train for, for a given legal action  $a_l \in \mathbb{A}(n)$  is:

$$p_{a_l} = \frac{N(n, a_l)}{\sum_{a \in \mathbb{A}(n)} N(n, a)} \quad (24)$$

For non legal actions  $a_n \in (\mathbb{A}_{possible} - \mathbb{A}(n))$ ,  $p_{a_n}$  is defined as:

$$p_{a_n} = p_{pre_{a_n}} \quad (25)$$

## 1.5 Data generation

The data used to train the neural network is generated by letting the best agent play several games against itself, until enough data has been generated to allow for training. In every game, at every game state, the MCTS performs 50 simulations. Once the simulations are done the action is chosen.

### 1.5.1 Action selection

There are two methods for action selection for a given node  $n_t$ : deterministic and probabilistic. The first will always return the action  $a = \text{argmax}(N(\mathcal{E}(n_t, a \in \mathbb{A}(n_t))))$  of the most traversed edge, while the second will return a random action where the probability of selecting an action  $a_i \in \mathbb{A}(n_t)$  is:

$$P(X = a_i, n_t) = \frac{N(\mathcal{E}(n_t, a_i))}{\sum_{j \in \mathbb{A}(n_t)} N(\mathcal{E}(n_t, j))} \quad (26)$$

( $\mathbb{A}(s)$  are the allowed actions for state  $s$ .) Action selection during the training phase shall initially be probabilistic, and deterministic later on. The handover point shall be defined as the configurational constant 'probabilistic\_moves'  $\in \mathbb{N}^+$ . During games outside the training loop, actions are always selected deterministically.

### 1.5.2 Memory

The memory stores a certain amount of memory elements. A memory element consists of a gamestate  $g \in \mathbb{G}$ , its action values  $v \in \mathbb{R}^{|\mathbb{A}|}$  and the true reward  $r \in \{1, -1, 0\} = R(g, a)$  where  $a$  is the action taken during play at that game state. The memory stores memory elements in a long list. After an action has been selected, but before any updates to the game simulation are made, the current game state is passed to temporary memory along with its action values  $v$ . Together they create a new memory element. This element's  $r$  is currently undefined.  $v$  is defined as:

$$v_a = \begin{cases} P(X = a, \mathcal{N}(g)) & a \in \mathbb{A}(g) \\ p_{pre_a} & \end{cases} \quad (27)$$

$\mathbb{A}(g)$  is the set of all legal actions.  $p_{pre}$  is defined in section ?? on page ??, and is used for all non legal actions.  $P$  is defined in equation 26 on page 25.

#### 1.5.2.1 Memory update

Once the game is over, the winning player is determined and the value  $r$  of every memory element in the temporary memory is updated.  $r$  is 1 if the player taking an action at that state won, -1 if he lost and 0 if the game ended in a draw. The updated states are then passed to memory.

#### 1.5.2.2 Model Training

Once the memory size exceeds 30'000 states, the self-playing stops and the neural network is trained as described in section: 1.4.2.3.

## 1.6 Model evaluation

In order to train the neural network, the "best player" generates data used to train the current network. After every time the current neural network has been updated, it plays 20 games against the best player. If it wins more than 1.3 times as often as the current best player, it is considered better. If this is the case, the neural network of the "current player" is saved to file and the old "best player" is replaced with the "current player" to become the new "best player". It is advantageous to force the network to win 1.3 times as often as that reduces the chance of the network just getting lucky.

## 2 Evaluation

To give us an idea of how good a player is, it would be useful to express performance using a single number. This number should not only give us a ranking but also allow for predictions of the winner of a game between two players and thus give us a measure of the relative strength of the players. One such rating method is the so called elo-rating method. [?]

### 2.1 Elo-rating

The elo-rating system assigns every player  $p$  a number  $r_p \in \mathbb{R}$ . In general, the larger  $r_p$  the better the player. More specifically, given two players  $a$  and  $b$  with elo-ratings  $r_a$  and  $r_b$ , the expected chance  $E$  of  $a$  winning against  $b$  is [?]:

$$E = \frac{1}{1 + e^{(r_b - r_a)/400}} \quad (28)$$

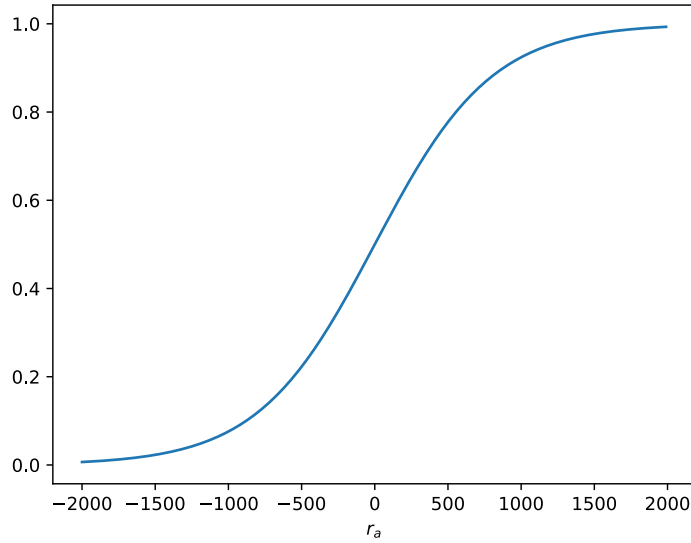


Figure 6: elo-rating win probability for  $r_b = 0$

This function describes a sigmoid curve. This makes sense, because if the players have major strength discrepancies  $E$  converges to 1 or 0. When  $a$  and  $b$  play a game against each other, their elo,ratings are updated as follows[?]:

$$r_{n+1} = r_n + K(W - E) \quad (29)$$

with:

$r_{n+1}$  the new rating for the player.

$r_n$  the current rating of the player.

$W = s_a$  which is defined by equation 30 where  $a$  is the player to be updated.

$E$  the expected chance of winning, see equation 28.

$K$  is a constant controlling the sensitivity of the update function.

However, to avoid slow convergence of elo-ratings, a more direct formula is used to approximate the rating of an agent  $a$ . This is done by playing a predetermined amount of games against player  $b$  whose elo-rating  $r_b$  is known and unchanged throughout this process. First,  $a$  and  $b$  play a predetermined amount of games  $m$  and the score  $s_a$  of  $a$  is computed as [?]:

$$s_a = \frac{1}{m} \sum \begin{cases} 1 & a \text{ wins} \\ \frac{1}{2} & \text{tie} \\ 0 & a \text{ loses} \end{cases} \quad (30)$$

Assuming that this is the probability of  $a$  winning against  $b$ ,  $a$ 's elo-rating can be computed by solving equation 28 to  $r_a$  (fig 7 on page 29):

$$r_a = r_b - \ln \left( \frac{1 - s_a}{s_a} \right) \cdot 400 \quad (31)$$

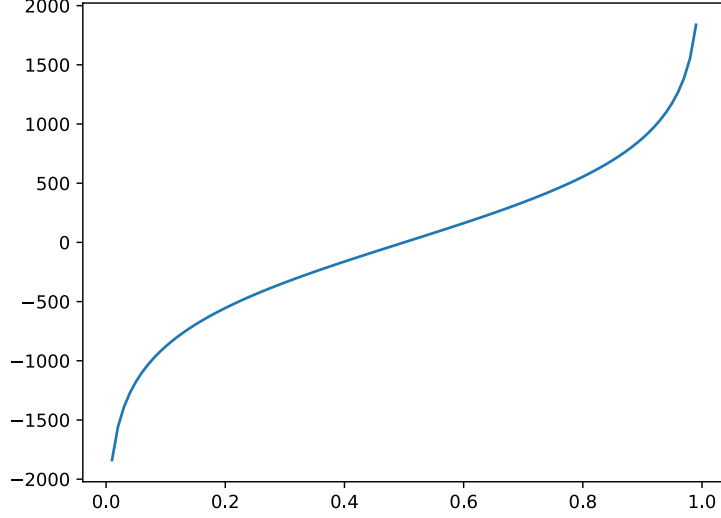


Figure 7: elo inverse function

Since a ranking of all the agents already exists (see section 1.6 on page 26), an agent’s elo-rating can be computed by playing against an older version and then using equation 31 to determine its elo-rating.

### 2.1.1 Relativity of the Elo-rating

The only problem is that elo is a relative rating. The rating of any other agent depends on its performance against other agents and their elo-ratings. Therefore, one must give the system a base rating for at least one predefined agent. In this case, there are no previously known elo-rated agents, so I defined the untrained agent’s elo-rating as 100. All other elo-ratings are relative to that.

## 2.2 Elo results

The rating  $r_i$  of any agent version  $i$  must in general be greater than the rating of the last version  $r_i > r_{i-1}$ . Furthermore, the expected minimal increase in rating  $\Delta r_{min} = r_i - r_{i-1}$  is:

$$\Delta r_{min} = -\ln\left(\frac{1 - s_i}{s_i}\right) \cdot 400 \quad (32)$$

As a certain scoring threshold  $\theta = 1.3$  was used during training to minimize the effect of noise in the evaluation, a prediction of  $s_i$  can be made. Given that  $s_a$  and  $s_b$  are the scores of two players that play against each other, then by definition:

$$s_a + s_b = 1 \quad (33)$$

Due to the imposed scoring threshold  $\theta$  and the assumption that there are no ties:

$$s_a \geq s_b \cdot \theta \quad (34)$$

(if  $s_a$  has a higher version number than  $s_b$ )

For  $\theta = 1.3$  this means that the expected average change in rating  $\Delta r$ :

$$\Delta r \geq -\ln\left(\frac{1}{\theta}\right) \cdot 400 = \Delta r_{min} \cong 105 \quad (35)$$

Collected data shows this to be true (fig 8 on page 31). The same data shows that the average  $\Delta r$  is in fact roughly 408, which would equate to a  $\theta$  of

$$\theta = \frac{1}{e^{\frac{-\Delta E}{400}}} \cong 2.8 \quad (36)$$

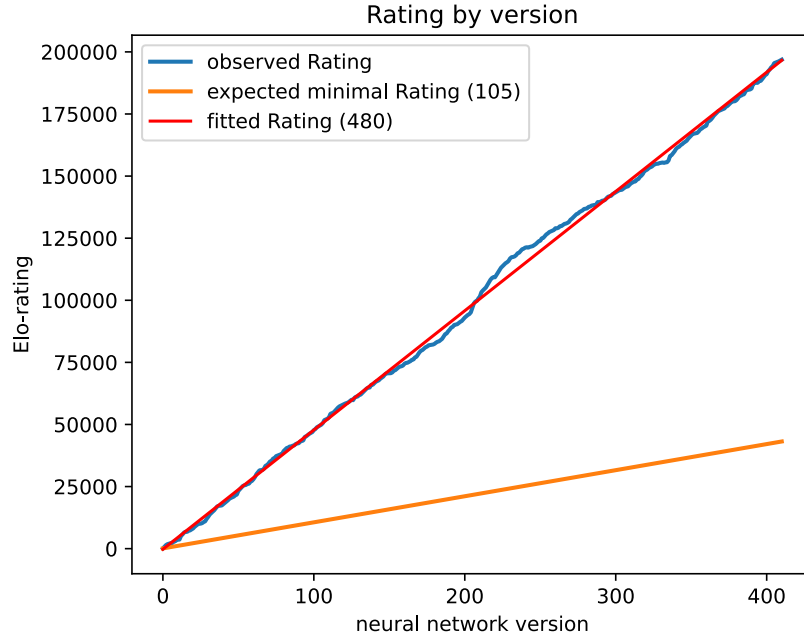


Figure 8: Elo-rating of agents based on their version along with the expected minimal rating  $\Delta r_{min}$  and the best fitted rating  $\Delta r$ .

### 3 Servers and Clients

In computer science, server-client-communications are a form of distributed application, that allows multiple machines to communicate and share data. In general, the server will wait for connections, while the client will initialize a connection with the server. To accomplish this, the server must listen to a certain port and the client must know the servers ip and port. In our case, the communications use the TCP and HTTPS protocols. Alpha Zero uses three distinct servers: an AI server, a data server and an Apache web server.

#### 3.1 The Web Server

Alpha Zero’s web server uses the Apache web server application. The web server is used to host static files such as the source code for the iOS client (see section 3.4.2 on page 36), a debug version of the same client, the domain name of the AI server and the domain name of the data server. All these files were located at <https://wandhoven.ddns.net/code/AlphaZero/>

#### 3.2 The Data Server

The data server stores all global information. This was just the elo-rating to begin with, but was later expanded to handle all data. This explains the somewhat strange communication protocol. Requests to the server begin by sending a 4 byte signed integer  $a$  identifying the general action the server must perform. The first action  $a = 1$  will return the elo-rating of a certain agent. This will require a further 4 bytes identifying the agent. The second action  $a = 2$  will set an agents elo-rating. Two 4 byte integers are sent, the first identifying the agent and the second the elo-rating to set to. The third action  $a = -1$  will require a 4 byte signed integer  $e$  and return the agent’s identifier with an elo-rating equal to  $r$  defined as:

$$r = \min(\{x \in \mathbb{E} | x \geq e\}) \quad (37)$$

where  $\mathbb{E}$  is the set of the elo-ratings of all agents. The last action  $a = -2$  will access the custom data part of the server. This subsection will require

an integer describing how many bytes the request consists of. The request is encoded using the python pickle library and consists of either a tuple containing a string, and a list of strings; or a tuple containing a string, a list of strings, and any other data type. In the first case, the system will return the value of the saved data associated with that request. In the second case, the value of the associated the variable will be set to whatever the third value is.

The first two variables of the tuple are a string  $f$  and a list of strings  $k$ .  $f$  tells the server in which file the variable is stored. Therefore, the server will load the json file with the name  $f$ .  $k$  is the list of keys used to index the dictionary  $f$ .

For example, with  $f = \text{example.json}$ , the server would decode the json file "example.json" shown in listing 1 on page 33. Assuming that  $k = ["\text{address}", "\text{city}"]$ , the server would first search for key "address" in the outer most directory. At that key, there is another dictionary, which is then searched for the key "city". At that key, there is a string ("New York"), which would be returned to the client.



```
1 {
2   "firstName": "John",
3   "lastName": "Smith",
4   "isAlive": true,
5   "age": 27,
6   "address": {
7     "streetAddress": "21 2nd Street",
8     "city": "New York",
9     "state": "NY",
10    "postalCode": "10021-3100"
11  },
12  "phoneNumbers": [
13    {
14      "type": "home",
15      "number": "212 555-1234"
16    },
17    {
18      "type": "office",
19      "number": "646 555-4567"
20    }
21  ],
22  "children": [],
23  "spouse": null
24 }
```

Listing 1: example.json  
from <https://en.wikipedia.org/wiki/JSON>

### 3.3 The AI Server

The AI server is used to evaluate a state and determine the best action using Alpha Zero. This is done by sending the server the state and waiting for it to send back the action.

### 3.3.1 State Transmission

To send a state to the server, the state's  $6 \times 7$  board is converted into an array of 85 boolean values<sup>2</sup>. The first 42 booleans represent whether or not the starting player has a stone at that position. Positions are mapped from left to right and then top to bottom. The table below shows the order in which the positions will be added to the boolean array. The next 42 booleans are identical to the first but for the non-starting player. The last position tells the server which player is taking the next action.

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  |
| 7  | 8  | 9  | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| 35 | 36 | 37 | 38 | 39 | 40 | 41 |

Consider the following game state (fig 9 on page 34) at which the starting player is at turn.

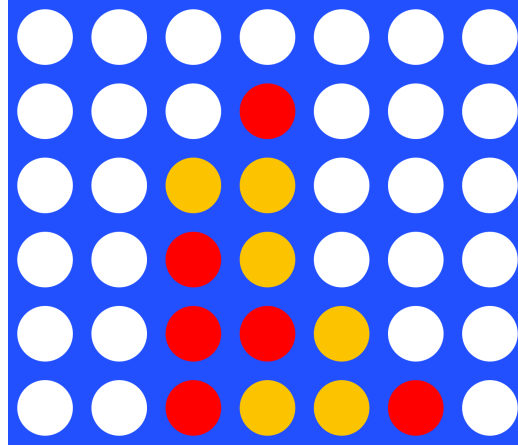


Figure 9: example game state

---

<sup>2</sup>Stored as integers 0 and 1 in memory

This state's boolean array  $a$  is:

$$a = \text{vec} \left( \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \textcolor{red}{1} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{orange}{0} & \textcolor{orange}{0} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{1} & \textcolor{orange}{0} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{orange}{0} & 0 & 0 \\ 0 & 0 & \textcolor{red}{1} & \textcolor{orange}{0} & \textcolor{orange}{0} & \textcolor{red}{1} & 0 \end{bmatrix} \right) \frown \text{vec} \left( \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \textcolor{red}{0} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{orange}{1} & \textcolor{orange}{1} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{0} & \textcolor{orange}{1} & 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{0} & \textcolor{orange}{0} & \textcolor{orange}{1} & 0 & 0 \\ 0 & 0 & \textcolor{red}{0} & \textcolor{orange}{1} & \textcolor{orange}{1} & \textcolor{red}{0} & 0 \end{bmatrix} \right) \frown \langle 1 \rangle \quad (38)$$

Where  $\text{vec}$  is the matrix vectorization defined in section 4.5 on page 38 and  $\frown$  is the vector concatenation defined in section 4.6 on page 39. To the front of this vector, the version of the AI we want to play against is added. If the version is invalid the server will default to the best version.

### 3.3.2 Action Selection

The AI's action is selected by running MCTS simulations and choosing the action with the most evaluations. It is the same algorithm as the deterministic method defined in section 1.5.1 on page 25.

### 3.3.3 Action Transmission

The action is stored and transmitted as a four byte integer.

## 3.4 Clients

There are two clients for the AlphaZero system. The first is a python client for DOS, macOS, Linux, etc., and the second was developed for iOS using pythonista.

### 3.4.1 Desktop Client

The Desktop client will allow the player to play against an AI with a slightly better elo-rating than the player's. This is done by creating an account on the data server. The data server will give the client a unique number representing its account. The Client will then proceed to save this number and request the player's elo-rating. Finally, the AI version with the closest but better elo-rating to that of the player is requested. Now the client renders the game board and randomly selects a starting player. The client will then wait for

user inputs and query the server as appropriate, i.e. what player is at turn. When the game is done, the client's log of the game is uploaded to the data server and the player will be shown the appropriate end-of-the-game screen. The client also has the possibility to request game logs and replay games. This client version uses the python socket library for communication and Tkinter to render the game board.

### 3.4.2 iOS Client

The iOS version does the same thing as the Desktop version with a few differences. Firstly, it renders the board using the pythonista scene library. Secondly, it does not create or store accounts. The player will always play against the best AI version. Thirdly, it does not have the possibility to replay games. Lastly, the Client will request its actual source code from the web server to allow for easier updating.

## 4 Further definitions

### 4.1 Set Exclusion

Let  $\mathbb{A} - \mathbb{B}$  be the set exclusion of  $\mathbb{A}$  and  $\mathbb{B}$ .

$$\mathbb{A} - \mathbb{B} = \mathbb{A} \setminus \mathbb{B} = \{x : x \in \mathbb{A} \text{ and } x \notin \mathbb{B}\} \quad (39)$$

### 4.2 Hadamard product

Let  $A$  and  $B$  be  $2 \times m \times n$  matrices. For all  $i \in [1, m]$  and  $j \in [1, n]$  the hadamard product  $A \circ B$  is defined as:

$$(A \circ B)_{ij} = A_{ij} \cdot B_{ij} \quad (40)$$

For example consider the following  $2 \times 3$  matrices:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & b_{32} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \\ a_{31}b_{31} & a_{32}b_{32} \end{bmatrix}$$

## 4.3 Inner Product

### 4.3.1 Matrices

Let  $A \in \mathbb{R}^{m \times n}$  and  $B \in \mathbb{R}^{m \times n}$  be two  $n \times m$  matrices.

Let their Inner Product  $\langle A, B \rangle_I : \mathbb{R}^{m \times n}, \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$  be defined as:

$$\langle A, B \rangle_I = \sum_{i=1}^m \sum_{j=1}^n A_{ij} B_{ij} = \sum_{i=1}^m \sum_{j=1}^n (A \circ B)_{ij} \quad (41)$$

For example consider the following  $2 \times 3$  matrices:

$$\left\langle \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & b_{32} \end{bmatrix}, \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \right\rangle_I = \begin{aligned} & a_{11}b_{11} + a_{12}b_{12} + a_{21}b_{21} \\ & + a_{22}b_{22} + a_{31}b_{31} + a_{32}b_{32} \end{aligned}$$

### 4.3.2 $n$ -dimensional Tensors

Let  $A \in \mathbb{R}^{m \times \dots}$  and  $B \in \mathbb{R}^{m \times \dots}$  be two  $n$  dimensional  $m \times \dots$  tensors with  $n > 2$

Let the Inner product  $\langle A, B \rangle_I : \mathbb{R}^{m \times \dots}, \mathbb{R}^{m \times \dots} \rightarrow \mathbb{R}$  be defined as:

$$\langle A, B \rangle_I = \sum_{i=0}^m \langle A_i, B_i \rangle_I \quad (42)$$

## 4.4 Submatrix

Let  $m = m_{ijk}$  be an  $m \times n \times o$  dimensional tensor.

Let  $<>_{S_{x \times y, ab}}$  be the matrix slicing operator.  $x \times y$  is the size of the submatrix the operation should output.  $ab$  is the top left position of the submatrix within the outer matrix. For the operation to be defined, the following must be true:  $x \in [0, m[$ ,  $y \in [0, n[$ ,  $a \in [1, m - x]$  and  $b \in [1, n - y]$ . The submatrix  $< m >_{S_{x \times y, ab}}$  is defined as:

$$< m >_{S_{x \times y, ab}} = \begin{bmatrix} [m_{ab1}, \dots, m_{abo}] & \dots & [m_{(a+x)b1}, \dots, m_{(a+x)bo}] \\ \vdots & \ddots & \vdots \\ [m_{a(b+y)1}, \dots, m_{abo}] & \dots & [m_{(a+x)(b+y)o}] \end{bmatrix}$$

## 4.5 Vectorization

The vectorization of an  $m \times n$  matrix  $A$ , denoted  $\text{vec}(A)$ , is the  $mn \times 1$  column vector obtained by stacking the columns of the matrix  $A$  on top of one another:

$$\text{vec}(A) = [A_{11} \dots A_{1m} A_{21} \dots A_{2m} \dots A_{n1} \dots A_{nm}]^T \quad (43)$$

Taken verbatim from:[?]

For example, the  $3 \times 2$  matrix  $A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$  vectorizes to

$$\text{vec}(A) = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}$$

### 4.5.1 Tensors

Let  $T \in \mathbb{R}^{n \times \dots}$  be a  $n$ -dimensional Tensor. Let  $\text{vec}(T)$  be defined as:

$$\text{vec}(T) = T_0 \frown T_1 \frown \dots \frown T_n \quad (44)$$

Where  $\frown$  is defined defined in section 4.6 on page 39. For those who are familiar with python's numpy library,  $\text{vec}$  is equivalent to `numpy.flatten`. For example consider the following 3 d-Tensor vectorization:

$$\text{vec} \left( \left( \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \right) \right) = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix}$$

## 4.6 Vector Concatination

Vector Concatination of two vectors  $v$  and  $u$  of dimensions  $n_v$  and  $n_u$ , denoted  $v \frown u$ , is the  $n_v + n_u$  dimensional vector obtained by placing both vectors one on top of the other.

$$v \frown u = \begin{pmatrix} v_1 \\ \vdots \\ v_{n_v} \\ u_1 \\ \vdots \\ u_{n_u} \end{pmatrix} \quad (45)$$

## Acknowledgements

- Leonie Scheck, Frederik Ott, Nico Steiner, Wolfgang Wandhoven for aiding in evaluating the AI against human players and providing feedback.

## 5 Code

### 5.1 AlphaZeroPytorch

#### 5.1.1 AlphaZeroPytorch.h

```
1 // AlphaZeroPytorch.h : Include file for standard system include
   files ,
2 // or project specific include files .
3
4 #pragma once
5
6 #include <iostream>
7
8 // TODO: Reference additional headers your program requires here
   .
```



### 5.1.2 AlphaZeroPytorch.cpp

```
1 // Entry point of training loop
2 //
3
4 #include "AlphaZeroPytorch.h"
5 #include <ai/playGame.hpp>
6 #include <io.hpp>
7 #include <chrono>
8 #include <thread>
9 #include "makeFiles.hpp"
10
11
12 int main(int argc, char ** argv)
13 {
14     /*std::ofstream out("out.txt");
15     std::streambuf* coutbuf = std::cout.rdbuf(); //save old buf
16     std::cout.rdbuf(out.rdbuf()); //redirect std::cout to out.txt!
17     */
18     if (torch::cuda::cudnn_is_available())
19     {
20         std::cout << "\33[1;32mcuDNN is available\33[0m" << std::endl;
21     }
22     else if (torch::hasXLA())
23     {
24         std::cout << "\33[1;32mXLA is available\33[0m" << std::endl;
25     }
26     else
27     {
28         std::cout << "\33[1;31mWarning: cuDNN is unavailable,
29         consider using a CUDA enabled GPU\33[0m" << std::endl;
30     }
31     std::vector<char*> devices = { DEVICES };
32     for (auto const& device : devices)
33     {
34         std::cout << device << ", ";
35     }
36     std::cout << std::endl << "started training" << std::endl;
37     createFolders();
38     AlphaZero::ai::train(-1);
39 #if ProfileLogger
40     debug::Profiler::profiler.log();
```

```
41 #endif
42     return 0;
43 }
```

### 5.1.3 cmake.sh

```
1 cmake . -DCMAKE_PREFIX_PATH=/cpp/libtorch
2 make
```

### 5.1.4 CMakeLists.txt

```
1 # CMakeList.txt : CMake project for AlphaZeroPytorch, include
   source and define
2 # project specific logic here.
3 #
4 project(AlphaZero)
5
6 set(CMAKE_CXX_STANDARD 17)
7 cmake_minimum_required(VERSION 3.8)
8 set(CMAKE_BUILD_TYPE Debug)
9
10 message("searching for Pytorch ...")
11 find_package(Torch REQUIRED)
12
13 message("adding source files ...")
14 file(GLOB SOURCES_Files
15     "include/*.cpp"
16     "include/ai/*.cpp"
17     "include/game/*.cpp"
18 )
19 file(GLOB HEADER_Files
20     "include/*.hpp"
21     "include/ai/*.hpp"
22     "include/game/*.hpp"
23 )
24 file(GLOB SERVER_Files
25     "include/Server/*.cpp"
26 )
27
28 file(GLOB OUTER_Files
29     "include/*.cpp"
30 )
31
32 file(GLOB GAME_Files
33     "include/game/*.cpp"
34 )
35
36 file(GLOB TEST_SOURCE
37     "include/test/*.cpp"
38     "include/ai/*.cpp"
39     "include/game/*.cpp"
40     "include/*.cpp"
41 )
42
```

```

43
44 # Add source to this project's executable.
45 add_executable (train "AlphaZeroPytorch.cpp" "AlphaZeroPytorch.h
    " ${SOURCES_Files} ${HEADER_Files})
46 add_executable (GameReplayer "Replay.cpp" "include/game/game.cpp
    " "include/config.cpp")
47 add_executable (runTest "test.cpp" ${TEST_SOURCE} ${
    SOURCES_Files})
48 add_executable (runServer "runServer.cpp" ${SOURCES_Files} ${
    SERVER_Files})
49 add_executable (convert "convertToJceFormat.cpp" "include/ai/
    modelWorker.cpp" "${OUTER_Files}")
50 add_executable (eloRaiting "doEloRaiting.cpp" ${SOURCES_Files} $
    {SERVER_Files})
51
52 target_compile_definitions(train PRIVATE cpuct_=2.0f)
53 target_compile_definitions(GameReplayer PRIVATE cpuct_=2.0f)
54 target_compile_definitions(runTest PRIVATE cpuct_=2.0f)
55 target_compile_definitions(runServer PRIVATE cpuct_=1.0f)
56 target_compile_definitions(convert PRIVATE cpuct_=2.0f)
57 target_compile_definitions(eloRaiting PRIVATE cpuct_=2.0f)
58
59
60 message("linking libs ...")
61 target_link_libraries(train "${TORCH_LIBRARIES}")
62 target_link_libraries(GameReplayer "${TORCH_LIBRARIES}")
63 target_link_libraries(runTest "${TORCH_LIBRARIES}")
64 target_link_libraries(runServer "${TORCH_LIBRARIES}")
65 target_link_libraries(convert "${TORCH_LIBRARIES}")
66 target_link_libraries(eloRaiting "${TORCH_LIBRARIES}")
67
68 message("adding includes ...")
69
70 if (WIN32)
71     add_executable(showLoss "showLoss.cpp" "include/log.cpp")
72     target_include_directories(showLoss PUBLIC "${CMAKE_SOURCE_DIR
        }/include")
73     target_include_directories(showLoss PUBLIC "C:/Users/Julia/
        AppData/Local/Programs/Python/Python39/include")
74     target_link_libraries(showLoss "C:/Users/Julia/AppData/Local/
        Programs/Python/Python39/libs/python39_d.lib")
75     target_link_libraries(showLoss "${TORCH_LIBRARIES}")
76
77
78     target_include_directories(train PUBLIC "D:/MyCode/CppLibs/

```

```

    include")
79 target_include_directories(convert PUBLIC "D:/MyCode/CppLibs/
    include")
80 target_include_directories(GameReplayer PUBLIC "D:/MyCode/
    CppLibs/include")
81 target_include_directories(runTest PUBLIC "D:/MyCode/CppLibs/
    include")
82 target_include_directories(showLoss PUBLIC "D:/MyCode/CppLibs/
    include")
83 target_include_directories(runServer PUBLIC "D:/MyCode/CppLibs
    /include")
84 target_include_directories(eloRaiting PUBLIC "D:/MyCode/
    CppLibs/include")
85
86 target_link_libraries (runServer "lib/sockpp-static-d")
87 target_link_libraries (eloRaiting "lib/sockpp-static-d")
88 endif (WIN32)
89 if (UNIX)
90     target_include_directories(train PUBLIC "/cpp/spdlog/include")
91
92     target_include_directories(runServer PUBLIC "/cpp/spdlog/
        include")
93     target_include_directories(runServer PUBLIC "/cpp/sockpp/
        include")
94
95     target_include_directories(eloRaiting PUBLIC "/cpp/spdlog/
        include")
96     target_include_directories(eloRaiting PUBLIC "/cpp/sockpp/
        include")
97
98     target_include_directories(convert PUBLIC "/cpp/spdlog/include
        ")
99
100
101     target_include_directories(GameReplayer PUBLIC "/cpp/spdlog/
        include")
102
103     target_include_directories(runTest PUBLIC "/cpp/spdlog/include
        ")
104
105     find_library(sockpp_location sockpp)
106     message("sockpp is at: ${sockpp_location}")
107     target_link_libraries(train "${sockpp_location}")
108     target_link_libraries(runServer "${sockpp_location}")
109     target_link_libraries(eloRaiting "${sockpp_location}")

```

```
110 endif(UNIX)
111
112
113 target_include_directories(train PUBLIC "${CMAKE_SOURCE_DIR}/
    include")
114 target_include_directories(runServer PUBLIC "${CMAKE_SOURCE_DIR}
    }/include")
115 target_include_directories(GameReplayer PUBLIC "${
    CMAKE_SOURCE_DIR}/include")
116 target_include_directories(runTest PUBLIC "${CMAKE_SOURCE_DIR}/
    include")
117 target_include_directories(convert PUBLIC "${CMAKE_SOURCE_DIR}/
    include")
118 target_include_directories(eloRaiting PUBLIC "${CMAKE_SOURCE_DIR}
    }/include")
119
120
121 message("done")
```

### 5.1.5 convertToJceFormat.cpp

```
1 // Entry point of the conversion from torch archive to jce and
  back conversion
2
3 #include <iostream>
4 #include <ai/model.hpp>
5 #include "makeFiles.hpp"
6
7 #define FILENAME "model.jce.bin"
8
9 int main(int argc, char** argv)
10 {
11     createFolders();
12     int version=-2;
13     std::cout << "What version do you want to convert -1 for
      current, -2 for inverse conversion: ";
14     std::cin >> version;
15
16
17     AlphaZero::ai::Model model("cpu");
18
19     if (version >= 0)
20     {
21         model.load_version(version);
22         model.jce_save_current(FILENAME);
23     }
24     else if (version == -1)
25     {
26         model.load_current();
27         model.jce_save_current(FILENAME);
28     }
29     else if (version == -2)
30     {
31         model.jce_load_from_file(FILENAME);
32         model.save_as_current();
33     }
34     else
35     {
36         return -1;
37     }
38
39     return 1;
40 }
```



### 5.1.6 doEloRating.cpp

```
1 // rate all existing agents
2
3 #include <Server/eloClient.hpp>
4 #include <ai/playGame.hpp>
5 #include <math.h>
6
7 void evaluateAgent(int agent, int games, AlphaZero::elo::
    eloClient const& elo, std::shared_ptr<AlphaZero::ai::Agent>
    lastAgent, std::shared_ptr<AlphaZero::ai::Agent> currentAgent
8 )
9 {
10     std::cout << "evaluating elo for: " << agent << std::endl;
11
12     int othersElo = elo.getElo(agent - 1);
13     std::cout << "others elo is: " << othersElo << std::endl;
14     AlphaZero::Game::Game* game = new AlphaZero::Game::Game();
15     AlphaZero::ai::Memory* memory = new AlphaZero::ai::Memory();
16
17     if (agent - 1 > 0)
18     {
19         lastAgent->model->load_version(agent - 1);
20     }
21     if (agent > 0)
22     {
23         currentAgent->model->load_version(agent);
24     }
25     auto data = AlphaZero::ai::playGames_inThreads(game, lastAgent
26         .get(), currentAgent.get(), memory, 2, 1, games, "
27         eloEvaluation");
28
29     int wins = data[currentAgent.get()];
30     int losses = data[lastAgent.get()];
31     int ties = games - wins - losses;
32
33     float score = ((float)wins + 0.5f * (float)ties) / ((float)games
34         );
35     if (score > 0.99)
36     {
37         score = 0.99f;
38     }
39     float Relo = (float)othersElo - log((1 - score) / score) *
40         400;
41     elo.setElo(agent, (int)Relo);
```

```

37
38     std::cout << wins << " wins, " << ties << " ties and " <<
        losses << " losses" << std::endl;
39     std::cout << "win Ratio is : " << score << std::endl;
40     std::cout << "new rating is: " << Relo << std::endl << std::
        endl;
41
42     delete game;
43     delete memory;
44 }
45
46 int main()
47 {
48     std::vector<char*> devices = { DEVICES };
49     std::shared_ptr<AlphaZero::ai::Agent> lastAgent = std::
        make_shared<AlphaZero::ai::Agent>(devices);
50     std::shared_ptr<AlphaZero::ai::Agent> currentAgent = std::
        make_shared<AlphaZero::ai::Agent>(devices);
51
52     AlphaZero::elo::eloClient elo;
53     std::cout << elo.setElo(0, 100) << std::endl;
54
55     int agent = 409;
56     while (true)
57     {
58         evaluateAgent(agent, 40, elo, lastAgent, currentAgent);
59         agent++;
60     }
61     return 1;
62 }

```

### 5.1.7 makeFiles.hpp

```
1 #pragma once
2
3 #ifndef UNIX
4 #include <filesystem>
5 #endif
6 #include <jce/string.hpp>
7 #include <string>
8
9 void createFolder(std::string str)
10 {
11     createFolder(str.c_str());
12 }
13
14 void createFolder(char name[])
15 {
16     #ifndef UNIX
17         std::filesystem::create_directories(name);
18     #else
19         const char* foo = "mkdir -p ";
20         char* full_text = new char[100];
21         strcpy(full_text, foo);
22         strcat(full_text, name);
23         system(full_text);
24     #endif
25 }
26
27 void inline createFolders()
28 {
29     char folder[100];
30
31     sprintf(folder, "models/run_%d", runVersion);
32     createFolder(folder);
33
34     sprintf(folder, "memory/run_%d", runVersion);
35     createFolder(folder);
36
37     sprintf(folder, "logs/c++");
38     createFolder(folder);
39
40     sprintf(folder, "logs/games");
41     createFolder(folder);
42 }
```

### 5.1.8 Replay.cpp

```
1 // Replay a game (unused)
2
3 #include <iostream>
4 #include <io.hpp>
5 #include <config.hpp>
6
7 int main(int argc, char ** argv)
8 {
9     #if SaverType == 2
10         auto saver = AlphaZero::io::ActionsOnly::GameSaver();
11     #elif SaverType == 1
12         auto saver = AlphaZero::io::FullState::GameSaver();
13     #endif
14     #if SaverType != 0
15         saver.load("test.bin");
16         saver.ConsoleReplay(0);
17     #endif
18     return 1;
19 }
```

### 5.1.9 runServer.cpp

```
1 // Entry point of the Ai server
2
3 #include <config.hpp>
4 #include <Server/server.hpp>
5 #include <game/game.hpp>
6
7 int main()
8 {
9     #if OPSMode == 1
10         AlphaZero::Server::TCPServer server;
11         server.mainLoop();
12
13     #elif OPSMode == 2
14         std::cout << "\33[1;31mUsing Test Server! \n\tset OPSMode to 1
15             for server if not testing\33[0m" << std::endl;
16         AlphaZero::Server::TestServer server(PORT);
17         server.mainLoop();
18     #endif
19 }
```

### 5.1.10 showLoss.cpp

```
1 // ehhhh. (not the fogiest clue)
2
3 #include <iostream>
4 #include <log.hpp>
5 #include <python.h>
6
7 int main(int argc, char** argv)
8 {
9     debug::log::lossLogger log("logs/games/loss.bin");
10
11     return 1;
12 }
```

### 5.1.11 test.cpp

```
1 // run the test script
2
3 #include <test/testSuit.hpp>
4 #include <jce/save.hpp>
5 #include <jce/load.hpp>
6 #include "makeFiles.hpp"
7 #include <vector>
8
9
10 int main(int argc, char** argv)
11 {
12     createFolders();
13     AlphaZero::test::runTests();
14
15     /*std::vector<int> count;
16     std::ifstream in("models/run_1/iterationCounter.jce");
17     jce::load(in, count);
18     in.close();
19     for (auto const& val : count)
20     {
21         std::cout << val << ", ";
22     }*/
23     return 1;
24 }
```

## 5.1.12 include

### 5.1.12.1 config.hpp

```
1 #pragma once
2 #include <log.hpp>
3 #include <bitset>
4 #include <mutex>
5
6 #ifdef unix
7 #define UNIX
8 #endif
9
10 // #ifdef UNIX
11 // #define DEVICES "cuda:0"
12 // #endif
13 // #ifndef UNIX
14 // #define DEVICES "cpu"
15 // #endif
16 #define DEVICES "cpu"
17
18 #define OPSMode 1
19
20 extern std::mutex console_mutex;
21 extern std::mutex rand_mutex;
22
23 /*
24 |-----|-----|
25 | OPSMode | Description |
26 |-----|-----|
27 | 1       | Run Server   |
28 |-----|-----|
29 | 2       | Run Tester   |
30 |-----|-----|
31 */
32
33 #define GameChecksLegalMoved true // the game will check if a
    move is legal not neded for training
34 #define stateSize 84
35 #define Training true
36 #define DEBUG false
37
38 #define U_computation(edge) (this->cput * edge.P * std::sqrt((
    float)Nb) / (float)(1 + edge.N))
39
```



```

40
41 // runn setting
42 #define runVersion 1
43 #define loadVersion -1
44
45 // Net settings
46 #define MaxQuDefault -99999
47 #define reg_const 0.0001
48 #define learningRage 0.1
49 #define Momentum 0.9
50
51 // simulation setting
52 #define MCTSSimulations 50
53 // #define cpuct_ 2.0f
54 #define ProbabilisticMoves 10
55 #define Alpha 0.9
56 #define EPSILON 0.2f
57
58 // memory setting
59 #define memory_size 30000
60
61 // self play
62 #define EPOCHS 1
63 #define GEN_THREADS 60
64 #define probabilistic_moves 10 // how many moves are prabilistic
    in the begining of the game to aid in exploration
65
66 // training
67 #define Training_loops 20
68 #define Training_batch 256
69 #define Training_epochs 5
70
71 // turney
72 #define Turnement_probabilisticMoves 2
73 #define TurneyEpochs 1
74 #define TurneyThreads 20
75 #define scoringThreshold 1.3
76
77 // console
78 #define RenderTrainingProgress false
79 #define RenderGenAndTurneyProgress false
80
81 // Saving
82 #define SaverType 0
83 /* +-----+

```

```

84 | SaverType | Description |
85 |-----|-----|
86 | 0 | no Saver |
87 |-----|-----|
88 | 1 | save full state to file |
89 |-----|-----|
90 | 2 | Save taken Actions to file |
91 |-----|-----|*/
92 #define SaverIntSize 1
93
94
95 typedef std::bitset<stateSize> IDType;

```

### 5.1.12.2 config.cpp

```
1 #include "config.hpp"
2
3 std::mutex console_mutex;
4 std::mutex rand_mutex;
```

### 5.1.12.3 io.hpp

```
1 #pragma once
2 #include <jce/load.hpp>
3 #include <jce/save.hpp>
4 #include "config.hpp"
5
6 // classes that record games and stores them to file.
7
8 namespace AlphaZero
9 {
10     namespace io
11     {
12         namespace FullState
13         {
14             class GameSaver
15             {
16                 // list of states the game passed though
17                 public: std::list<std::list< std::shared_ptr<Game::
GameState>>>> states;
18                 // add a state to the states list last list
19                 public: void addState(std::shared_ptr<Game::
GameState> state);
20                 // add a game to the states list
21                 public: void addGame();
22                 // remove all games
23                 public: void clear();
24                 // save the saver to file
25                 public: void save(char filename[]);
26                 // load a saver from file
27                 public: void load(char filename[]);
28
29                 // replay game in console (debug)
30                 public: void ConsoleReplay(int itx);
31             };
32         }
33         namespace ActionsOnly
34         {
35             class GameSaver
36             {
37                 // list of action taken during game pass
38                 public: std::list<std::list<unsigned int>> states;
39                 // add an action to the current game
40                 public: void addState(int);
41                 // add a game to the list of games
```

```

42         public: void addGame();
43         // remove all games
44         public: void clear();
45         // save saver to file
46         public: void save(char filename[]);
47         // load saver from file
48         public: void load(char filename[]);
49
50         // replay game in console (debug)
51         public: void ConsoleReplay(int idx);
52     };
53 }
54 }
55 }
56
57 inline void AlphaZero::io::FullState::GameSaver::addState( std::
shared_ptr<Game::GameState> state)
58 {
59     this->states.back().push_back(state);
60 }
61
62 inline void AlphaZero::io::FullState::GameSaver::addGame()
63 {
64     this->states.push_back( std::list<std::shared_ptr<Game::
GameState>>());
65 }
66
67 inline void AlphaZero::io::FullState::GameSaver::clear()
68 {
69     this->states.clear();
70 }
71
72 inline void AlphaZero::io::FullState::GameSaver::save(char
filename[])
73 {
74     std::ofstream fout;
75     fout.open(filename, std::ios::binary);
76     if (fout.is_open())
77     {
78         jce::save(fout, this->states);
79         fout.close();
80     }
81     else
82     {
83         throw "Game saver file not opened.";

```

```

84     }
85 }
86
87 inline void AlphaZero::io::FullState::GameSaver::load(char
      filename[])
88 {
89     std::ifstream infile(filename, std::ios::binary);
90
91     if (infile.is_open())
92     {
93         jce::load(infile, this->states);
94         infile.close();
95     }
96     else
97     {
98         throw "Game saver file not open.";
99     }
100 }
101
102 inline void AlphaZero::io::FullState::GameSaver::ConsoleReplay(
      int idx)
103 {
104     for (auto const& state : *std::next(this->states.begin(),
      idx))
105     {
106         state->render();
107     }
108 }
109
110
111 inline void AlphaZero::io::ActionsOnly::GameSaver::addState(int
      actions)
112 {
113     this->states.back().push_back(actions);
114 }
115
116 inline void AlphaZero::io::ActionsOnly::GameSaver::addGame()
117 {
118     this->states.push_back(std::list<unsigned int>());
119 }
120
121 inline void AlphaZero::io::ActionsOnly::GameSaver::clear()
122 {
123     this->states.clear();
124 }

```

```

125
126 inline void AlphaZero::io::ActionsOnly::GameSaver::save(char
    filename[])
127 {
128     std::ofstream file(filename, std::ios::binary);
129     if (file.is_open())
130     {
131         jce::save(file, this->states);
132         file.close();
133     }
134     else
135     {
136         throw "Game saver file not opened.";
137     }
138 }
139
140 inline void AlphaZero::io::ActionsOnly::GameSaver::load(char
    filename[])
141 {
142     std::ifstream file(filename, std::ios::binary);
143     if (file.is_open())
144     {
145         jce::load(file, this->states);
146         file.close();
147     }
148     else
149     {
150         throw "Game saver file not opened.";
151     }
152 }
153
154 inline void AlphaZero::io::ActionsOnly::GameSaver::ConsoleReplay
    (int idx)
155 {
156     Game::Game game = Game::Game();
157
158     for (int action : *std::next(this->states.begin(), idx))
159     {
160         game.render();
161         game.takeAction(action);
162     }
163     game.render();
164 }

```

#### 5.1.12.4 log.hpp

```
1 #pragma once
2 #define threads 0
3
4 //logging
5 #define MainLogger true
6 #define MCTSLogger false
7 #define MemoryLogger false
8 #define ProfileLogger false
9 #define ModelLogger true
10 #define LossLogger true
11
12
13 #include <unordered_map>
14 #include <spdlog/sinks/basic_file_sink.h>
15 // #include <memory>
16 #if ProfileLogger
17 #include "timer.hpp"
18 #endif
19 #include <stdio.h>
20 #include <chrono>
21 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
    || ModelLogger || LossLogger)
22
23 // logging header
24
25 namespace debug {
26 #if ProfileLogger
27     // Profiler used to profile the AI
28     namespace Profiler {
29         class MCTSProfiler {
30             // its a timer.
31             private: utils::Timer timer;
32             // times by processid.
33             public: std::unordered_map<unsigned int, double> times;
34             // how long the profiler spent without an assigned
35             profiler.
36             private: double rest;
37
38             // whether the Object has not switchOperation is called
39             for the first time
40             private: bool first = true;
41             // false if the Profiler has an assigned profiler. (else
42             true)
```



```

40     private: bool toRest = false;
41     // the id of the current process
42     private: unsigned int currentTime;
43
44     // switch to a diferant process
45     public: void switchOperation(unsigned int id);
46     // stop profiling the current process and start running on
47     rest.
48     public: void stop();
49     // print log to profile logger
50     public: void log();
51     // print to a diferant logger.
52     public: void log(std::shared_ptr<spdlog::logger> logger);
53     };
54     extern debug::Profiler::MCTSPProfiler profiler;
55 #endif
56 namespace log {
57     // create a logger of a certain name on a certain file
58     std::shared_ptr<spdlog::logger> createLogger(const char*
59     name, const char* file);
60     template<typename T>
61     // log vector to logger. and put some text befor every
62     line
63     void logVector(std::shared_ptr<spdlog::logger> logger, std::
64     vector<T>, char out[]);
65     // log vector of intagers to logger
66     void logVector(std::shared_ptr<spdlog::logger> logger, std::
67     vector<int>);
68     // log vector of floats to logger.
69     void logVector(std::shared_ptr<spdlog::logger> logger, std::
70     vector<float>);
71
72     class lossLogger
73     {
74     public: lossLogger();
75     public: lossLogger(const char file[]);
76
77     protected: std::vector<std::vector<std::pair<float, float>>>
78     vals;
79
80     public: void addValue(const float val, const float poly);
81     public: void addValue(const std::pair<float, float>& val);
82     public: void newBatch();
83     public: void save(const char file[]);

```

```

78
79     public: std::pair<float, float> operator [] (std::pair<size_t,
80         size_t> idx) const;
81     public: std::vector<std::pair<float, float>> operator [] (
82         size_t idx) const;
83     public: bool operator==(const lossLogger&);
84 };
85
86 #if MainLogger
87     extern std::shared_ptr<spdlog::logger> mainLogger;
88 #endif
89 #if MCTSLogger
90     extern std::shared_ptr<spdlog::logger> MCTS_Logger;
91 #endif
92 #if MemoryLogger
93     extern std::shared_ptr<spdlog::logger> memoryLogger;
94 #endif
95 #if ProfileLogger
96     extern std::shared_ptr<spdlog::logger> profileLogger;
97 #endif
98 #if ModelLogger
99     extern std::shared_ptr<spdlog::logger> modelLogger;
100 #endif
101 #if LossLogger
102     extern lossLogger _lossLogger;
103 #endif
104 }
105
106 inline std::shared_ptr<spdlog::logger> debug::log::createLogger(
107     const char* name, const char* file) {
108     // Create a daily logger - a new file is created every day on
109     // 2:30am
110     int success = remove(file);
111     auto logger = spdlog::basic_logger_mt(name, file);
112     return logger;
113 }
114
115 inline debug::log::lossLogger::lossLogger()
116 {
117     this->newBatch();
118 }
119
120 inline void debug::log::lossLogger::addValue(const float a,
121     const float b)

```

```

118 {
119     std::pair<float, float> data = { a, b };
120     this->addValue(data);
121 }
122
123 inline void debug::log::lossLogger::addValue(const std::pair<
124     float, float>& val)
125 {
126     this->vals.back().push_back(val);
127 }
128
129 inline void debug::log::lossLogger::newBatch()
130 {
131     this->vals.push_back({});
132 }
133
134 inline std::pair<float, float> debug::log::lossLogger::operator
135     [] (std::pair<size_t, size_t> idx) const
136 {
137     return this->vals[idx.first][idx.second];
138 }
139
140 inline std::vector<std::pair<float, float>> debug::log::
141     lossLogger::operator [] (size_t idx) const
142 {
143     return this->vals[idx];
144 }
145
146 template<typename T>
147 inline void debug::log::logVector(std::shared_ptr<spdlog::logger>
148     > logger, std::vector<T> vec, char out[])
149 {
150     logger->info(out, (vec[0]), (vec[1]), (vec[2]), (vec[3]),
151         (vec[4]), (vec[5]), (vec[6]));
152     logger->info(out, (vec[7]), (vec[8]), (vec[9]), (vec[10]),
153         (vec[11]), (vec[12]), (vec[13]));
154     logger->info(out, (vec[14]), (vec[15]), (vec[16]), (vec[17]),
155         (vec[18]), (vec[19]), (vec[20]));
156     logger->info(out, (vec[21]), (vec[22]), (vec[23]), (vec[24]),
157         (vec[25]), (vec[26]), (vec[27]));
158     logger->info(out, (vec[28]), (vec[29]), (vec[30]), (vec[31]),
159         (vec[32]), (vec[33]), (vec[34]));
160     logger->info(out, (vec[35]), (vec[36]), (vec[37]), (vec[38]),
161         (vec[39]), (vec[40]), (vec[41]));
162 }

```

```

153
154 inline void debug::log::logVector(std::shared_ptr<spdlog::logger
    > logger, std::vector<int> vec)
155 {
156     char out[] = "Action vals are: {:3d}, {:3d}, {:3d}, {:3d}, {:3
        d}, {:3d}, {:3d}";
157     logVector(logger, vec, out);
158 }
159
160 inline void debug::log::logVector(std::shared_ptr<spdlog::logger
    > logger, std::vector<float> vec)
161 {
162     char out[] = "Action vals are: {:1.2f}, {:1.2f}, {:1.2f},
        {:1.2f}, {:1.2f}, {:1.2f}, {:1.2f}";
163     logVector(logger, vec, out);
164 }
165 #if ProfileLogger
166 inline void debug::Profiler::MCTSPProfiler::switchOperation(
    unsigned int id)
167 {
168     this->stop();
169     this->first = false;
170     this->toRest = false;
171     this->timer.reset();
172     this->currentTime = id;
173 }
174
175 inline void debug::Profiler::MCTSPProfiler::stop()
176 {
177
178     if (!this->first) {
179         if (toRest) {
180             this->rest = this->rest + this->timer.elapsed();
181         }
182         double currentNum;
183         if (this->times.count(currentTime) == 0) {
184             currentNum = 0.0f;
185         }
186         else {
187             currentNum = this->times.at(this->currentTime);
188         }
189         double res = currentNum + this->timer.elapsed();
190         this->times.insert_or_assign(currentTime, res);
191     }
192     this->toRest = true;

```

```

193 }
194
195 inline void debug::Profiler::MCTSPProfiler::log() {
196     this->log(debug::log::profileLogger);
197 }
198
199 inline void debug::Profiler::MCTSPProfiler::log(std::shared_ptr<
    spdlog::logger> logger)
200 {
201     logger->info("using {} threads", threads);
202     logger->info("run Info:");
203     logger->info("");
204
205     // this is only true on the computers im using as im just
    entering the infomation here to save time
206 #ifdef WIN32
207     logger->info("os: Windows 10");
208     logger->info("CPU: Intel(R) Core(TM) i5-8350U CPU @ 1.70 GHz ");
209     logger->info("GPU: None");
210     logger->info("memory: 8 GB");
211 #else
212     logger->info("OS: Ubuntu 18.04");
213     logger->info("CPU: ??");
214     logger->info("GPU: Nvidia P4 cuda 11.4");
215     logger->info("memory: 7.8 Gb");
216 #endif
217
218     logger->info("
    ");
219     for (auto const& pair : this->times) {
220         logger->info("Profiler time id {} took {} s", pair.first,
            pair.second);
221     }
222     logger->info("everything else took: {}", this->rest);
223
224     logger->info("");
225     logger->info("0 : MCTS and NN forward");
226     logger->info("3 : Game Stuff");
227     logger->info("4 : Memory shuffeling");
228     logger->info("5 : NN Backward");
229 }
230 #endif
231 #endif

```

### 5.1.12.5 log.cpp

```
1 #include "log.hpp"
2 #include <stdio.h>
3 #include <jce/save.hpp>
4 #include <jce/load.hpp>
5
6
7
8 #if MainLogger
9 std::shared_ptr<spdlog::logger> debug::log::mainLogger = debug::
    log::createLogger("mainLogger", "logs/c++/mainLogger.log");
10 #endif
11 #if MCTSLogger
12 std::shared_ptr<spdlog::logger> debug::log::MCTSLogger = debug
    ::log::createLogger("MCTSLogger", "logs/c++/MCTSLogger.log"
    );
13 #endif
14 #if MemoryLogger
15 std::shared_ptr<spdlog::logger> debug::log::memoryLogger = debug
    ::log::createLogger("memoryLogger", "logs/c++/memoryLogger.
    log");
16 #endif
17 #if ProfileLogger
18 std::shared_ptr<spdlog::logger> debug::log::profileLogger =
    debug::log::createLogger("profileLogger", "logs/c++/
    profileLogger.log");
19
20 debug::Profiler::MCTSPProfiler debug::Profiler::profiler = debug
    ::Profiler::MCTSPProfiler();
21 #endif
22 #if ModelLogger
23 std::shared_ptr<spdlog::logger> debug::log::modelLogger = debug
    ::log::createLogger("ModelLogger", "logs/c++/ModelLogger.log"
    );
24 #endif
25 #if LossLogger
26 debug::log::lossLogger debug::log::_lossLogger = debug::log::
    lossLogger();
27 #endif
28
29 debug::log::lossLogger::lossLogger(const char file[])
30 {
31     std::ifstream in(file, std::ios::binary);
32     if (in.is_open())
```

```

33 {
34     jce::load(in, this->vals);
35 }
36 else
37 {
38     std::cout << "\33[31;1mFailed to load lossLogger from " <<
39     file << "\33[0m" << std::endl;
40 }
41 in.close();
42 }
43 void debug::log::lossLogger::save(const char file[])
44 {
45     std::ofstream out (file, std::ios::binary);
46     if (out.is_open())
47     {
48         jce::save(out, this->vals);
49     }
50     else
51     {
52         std::cout << "\33[31;1mFailed to save lossLogger to " <<
53         file << "\33[0m" << std::endl;
54     }
55     out.close();
56 }
57 bool debug::log::lossLogger::operator==(const lossLogger& other)
58 {
59     if (other.vals.size() == this->vals.size())
60     {
61         for (size_t batch = 0; batch < other.vals.size(); batch++)
62         {
63             if (other.vals[batch].size() == this->vals[batch].size())
64             {
65                 for (size_t idx = 0; idx < other.vals[batch].size(); idx
66                 +++)
67                 {
68                     if (other[idx] != (*this)[idx])
69                     {
70                         return false;
71                     }
72                 }
73             }
74             else
75             {

```

```
75         return false;
76     }
77 }
78     return true;
79 }
80     return false;
81 }
```



### 5.1.12.6 timer.hpp

```
1 #pragma once
2 #include <iostream>
3 #include <chrono>
4
5 namespace utils {
6     class Timer
7     {
8     public:
9         Timer();
10        void reset();
11        double elapsed() const;
12
13    private:
14        typedef std::chrono::high_resolution_clock clock_;
15        typedef std::chrono::duration<double, std::ratio<1>> >
16        second_;
17        std::chrono::time_point<clock_> beg_;
18    };
19
20
21 inline utils::Timer::Timer()
22 {
23     this->beg_ = clock_::now();
24 }
25
26 inline void utils::Timer::reset()
27 {
28     this->beg_ = clock_::now();
29 }
30
31 inline double utils::Timer::elapsed() const
32 {
33     return std::chrono::duration_cast<second_>(clock_::now()
34     - beg_).count();
35 }
```

### 5.1.12.7 ai

#### 5.1.12.7.1 agent.hpp

```
1 #pragma once
2 #include <ai/modelSynchronizer.hpp>
3 #include <ai/memory.hpp>
4 #include <jce/vector.hpp>
5 #include "utils.hpp"
6 #include <thread>
7
8 namespace AlphaZero {
9     namespace ai {
10         // Alpha zero agent used to call the MCTS and train the
11         // Neural Network.
12         class Agent {
13             // MCTS tree by thread id. (thread the game is running in)
14             public: std::unordered_map<size_t, std::shared_ptr<MCTS>>
15             tree;
16             // pointer to the model synchronizer that handles the
17             // model.
18             public: std::unique_ptr<AlphaZero::ai::ModelSynchronizer>
19             model;
20
21             // create an agent and initialize the models on multiple
22             // devices
23             public: Agent(std::vector<char*> devices);
24
25             // get the MCTS tree for the current thread
26             public: MCTS* getTree();
27
28             // remove all MCTS trees
29             public: void reset();
30
31             // get action and action values from the MCTS. Also call
32             // simulations and evaluate NN
33             public: std::pair<int, std::pair<std::vector<int>, float>>>
34             getAction(std::shared_ptr<Game::GameState> state, bool
35             probabilistic);
36
37             // run a single MCTS simulation
38             public: void runSimulations(Node*, MCTS* tree);
39
40             // evaluate and expand a leaf node.
41             private: float evaluateLeaf(Node*, MCTS* tree);
```

```

34
35     // train the neural network
36     public: void fit(std::shared_ptr<Memory> memory, unsigned
short iteration);
37
38     // evaluate the neural network
39     public: std::pair<float, std::vector<float>> predict(std::
shared_ptr<Game::GameState> state);
40     public: void predict(ModelData* data);
41     public: void predict(std::list<ModelData*> data);
42
43     // get action deterministically (largest edge.N)
44     private: std::pair<int, std::pair<std::vector<int>, float>>
derministicAction(Node* node);
45
46     // get action probabilistically (random waited by edge.N)
47     private: std::pair<int, std::pair<std::vector<int>, float>>
prabilisticAction(Node* node);
48     };
49     // call simulations untill MCTSIter == MCTSSimulations (
config.hpp)
50     void runSimulationsCaller(AlphaZero::ai::Agent* agent, Node*
node, MCTS* tree);
51 }
52 }
53
54 inline AlphaZero::ai::MCTS* AlphaZero::ai::Agent::getTree()
55 {
56     auto a = std::hash<std::thread::id>{}(std::this_thread::get_id
());
57     if (this->tree.count(a))
58     {
59         return this->tree[a].get();
60     }
61     else
62     {
63         auto tree = std::make_shared<MCTS>();
64         this->tree.insert({ a, tree });
65         return tree.get();
66     }
67 }
68
69 inline void AlphaZero::ai::Agent::reset()
70 {
71     this->tree.clear();

```

```

72 }
73
74 inline void AlphaZero::ai::Agent::runSimulations(Node* node,
75 MCTS* tree)
76 {
77     std::pair<Node*, std::list<Edge*>> serchResults = tree->
78     moveToLeaf(node);
79     float val = this->evaluateLeaf(serchResults.first, tree);
80     tree->backFill(serchResults.second, serchResults.first, val);
81     tree->addMCTSIter();
82 }
83
84 inline void AlphaZero::ai::runSimulationsCaller(AlphaZero::ai::
85 Agent* agent, Node* node, MCTS* tree)
86 {
87     while (tree->MCTSIter < MCTSSimulations) {
88         agent->runSimulations(node, tree);
89     }
90 }
91
92 inline float AlphaZero::ai::Agent::evaluateLeaf(Node* node, MCTS
93 * tree)
94 {
95     if (!node->state->done){
96         std::shared_ptr<Game::GameState> nextState;
97         Node* nextNode;
98         auto data = ModelData(node);
99         this->predict(&data);
100         for (auto& action : node->state->allowedActions) {
101             nextState = node->state->takeAction(action);
102             nextNode = tree->addNode(nextState);
103             Edge newEdge = Edge(nextNode, node, action, data.polys[
104             action].item<float>()); //the last is the prob
105             node->addEdge( action, newEdge);
106         }
107         return data.value;
108     }
109     return (float)std::get<0>(node->state->val);
110 }
111
112 inline void AlphaZero::ai::Agent::fit(std::shared_ptr<Memory>
113 memory, unsigned short run)
114 {
115     std::cout << "\33[35;1mretraining\33[0m" << std::endl;
116     for (int idx = 0; idx < Training_loops ; idx++) {

```

```

111 #if RenderTrainingProgress
112     jce::consoleUtils::render_progress_bar((float)idx / (float)
        Training_loops);
113 #endif
114     auto batch = Model::getBatch(memory, Training_batch);
115     for (size_t trainingEpoch = 0; trainingEpoch <
        Training_epochs; trainingEpoch++)
116         this->model->fit(batch, run, idx);
117
118 #if LossLogger
119     debug::log::lossLogger.newBatch();
120 #endif
121 }
122 this->model->synchronizeModels();
123 #if LossLogger
124     debug::log::lossLogger.save("logs/games/loss.bin");
125 #endif
126 #if RenderTrainingProgress
127     jce::consoleUtils::render_progress_bar(1.0f, true);
128 #endif
129 }
130
131 inline std::pair<float, std::vector<float>> AlphaZero::ai::Agent
    ::predict(std::shared_ptr<Game::GameState> state)
132 {
133     auto preds = this->model->predict(state);
134     float& val = preds.first;
135     std::vector<float> polys = std::vector<float>(action_count);
136
137     c10::Device device ("cpu");
138
139     torch::Tensor mask = torch::ones(
140         { 1, action_count },
141         c10::TensorOptions().device(c10::Device("cpu")).dtype(at::
            kBool)
142     );
143
144     for (auto idx : state->allowedActions)
145     {
146         mask[0][idx] = false;
147     }
148
149     torch::Tensor out = torch::softmax(torch::masked_fill(preds.
        second.cpu(), mask, -1000.0f), 1);
150

```

```

151     for (auto const& idx : state->allowedActions) {
152         polys[idx] = out[0][idx].item<float>();
153     }
154
155     return { val, polys };
156 }
157
158 inline void AlphaZero::ai::Agent::predict(ModelData* data)
159 {
160     this->model->addData(data);
161 }
162
163 inline void AlphaZero::ai::Agent::predict(std::list<ModelData*>
164     data)
165 {
166     this->model->predict(data);
167 }
168
169 inline std::pair<int, std::pair<std::vector<int>, float>>
170     AlphaZero::ai::Agent::derministicAction(Node* node)
171 {
172     int action = 0;
173     unsigned int max_N = 0;
174     unsigned int sum = 0;
175     std::vector<int> probs = jce::vector::gen(action_count, 0);
176     for (auto const& iter : node->edges) {
177         if (iter.second.N > max_N) {
178             max_N = iter.second.N;
179             action = iter.first;
180         }
181         probs[iter.second.action] = iter.second.N;
182     }
183     return { action, { probs, node->edges[action].Q } };
184 }
185
186 inline std::pair<int, std::pair<std::vector<int>, float>>
187     AlphaZero::ai::Agent::prabilisticAction(Node* node)
188 {
189     int action = -1;
190     int idx = 0;
191     unsigned int sum = 0;
192     std::vector<int> probs = jce::vector::gen(action_count, 0);
193
194     for (auto const& iter : node->edges) {
195         probs[iter.second.action] = (iter.second.N);

```

```
193     }
194     auto action_probs = (rand() % ai::getSumm(probs));
195
196     for (auto const& val : probs) {
197         action_probs -= val;
198         if (action_probs < 0) {
199             action = idx;
200             break;
201         }
202         idx++;
203     }
204     return { action, { probs, node->edges[action].Q } };
205 }
```

### 5.1.12.7.2 agent.cpp

```
1 #include "agent.hpp"
2 #include <stdlib.h>
3
4 AlphaZero::ai::Agent::Agent(std::vector<char*> devices)
5 {
6     this->tree = {};
7     this->model = std::make_unique<ModelSynchronizer>(devices);
8 }
9
10 std::pair<int, std::pair<std::vector<int>, float>>> AlphaZero::ai
    ::Agent::getAction(std::shared_ptr<Game::GameState> state,
        bool probabilistic)
11 {
12     #if ProfileLogger
13         debug::Profiler::profiler.switchOperation(0);
14     #endif
15     auto tree = this->getTree();
16     tree->MCTSIter = 0;
17     Node* node = tree->addNode(state);
18     #if threads > 0
19         std::vector<std::thread> threadvec;
20         for (int i = 0; i < threads; i++) {
21             threadvec.push_back(std::thread(runSimulationsCaller, this,
22                 node));
23         }
24     #endif
25     runSimulationsCaller(this, node, tree);
26     #if threads > 0
27         for (auto& thread : threadvec) {
28             thread.join();
29         }
30     #endif
31     try {
32         #if ProfileLogger
33             debug::Profiler::profiler.switchOperation(3);
34         #endif
35         if (probabilistic) {
36             return this->prabilisticAction(node);
37         }
38         else {
39             return this->derministicAction(node);
40         }
41     }
```



```
41 catch (const std::exception& ex) {  
42     std::cerr << "\33[31;1mError in Agent::getAction\33[0m" <<  
         std::endl;  
43     std::cerr << ex.what() << std::endl;  
44     throw ex;  
45 }  
46 }
```

### 5.1.12.7.3 MCTS.hpp

```
1 #pragma once
2 #include <mutex>
3 #include <game/game.hpp>
4 #include <jce/vector.hpp>
5
6 // remove one mutex – the Node mutex
7
8
9 namespace AlphaZero {
10     namespace ai {
11         class Node;
12
13         /*Class Representing the action connecting 2 nodes together
14         . It handles all The MCTS relevant variables and the mutex
15         for
16         * parallization
17         */
18         class Edge {
19             // The number of times the Edge was traversed
20             public: int N = 0;
21             // the probability initialized by the NN
22             public: float P = 0;
23             // the action asociated with the action
24             public: int action = 0;
25             // the amount of times this lead to a win
26             public: float W = 0;
27             // the win probability
28             public: float Q = 0;
29
30             // the node the edge leads from
31             public: Node* outNode;
32
33             // the node the edge leads to
34             public: Node* inNode;
35
36             // build an edge between two nodes on a certain action
37             with a certain prediction p
38             public: Edge(Node* outNode, Node* inNode, int action, float
39 p);
40
41             // debugging function used find faulty initializations (
42             depricated)
43             public: Edge();
```

```

39
40     // increase N by 1
41 public: void traverse();
42 };
43
44     // class representing a node in a MCTS graph
45 class Node {
46
47     // the game state the node is asociated with
48 public: std::shared_ptr<Game::GameState> state;
49
50     // the map of actions to edges that can be taken at the
    node
51 public: std::unordered_map<int, Edge> edges;
52
53     // create the node on a certain game state
54 public: Node(std::shared_ptr<Game::GameState>);
55
56     // check if the node is a leaf (edges.size() == 0)
57 public: bool isLeaf();
58
59     // add an edge to the map of edges.
60 public: void addEdge(int id, Edge& edge);
61 };
62
63     // generate a list with a float for every action in the
    game. the list is 0
64     // if the action is illegal and the computed Q by the mcts
    if not. the index in
65     // the list is the action number
66 std::vector<float> getQ(Node*);
67
68     // class that handles MCTS simulations
69 class MCTS {
70     // counter used to count the ammount of simulations
    performed by the MCTS.
71 public: unsigned short MCTSIter = 0;
72     // map of nodes by game state.
73 private: std::unordered_map<IDType, std::unique_ptr<Node>>
    MCTS_tree;
74
75     // add 1 to MCTSIter
76 public: void addMCTSIter();
77     // create a MCTS object and initialize it (do nothing)
78 public: MCTS();

```

```

79
80     // save cpuct to the model from the config file.
81     public: float cpuct = cpuct_;
82
83     // start from the given node and follow the procedure
84     // outlined in the paper to
85     // move to a leaf node
86     public: std::pair <Node*, std::list<Edge*>> moveToLeaf(Node
87 *));
88
89     // back up from the leaf node and update the values
90     // appropriately.
91     public: void backFill(std::list<Edge*>&, Node* leaf, float
92 val);
93
94     // get the node by Game id
95     public: Node* getNode(IDType);
96
97     // get a node by game state (will automatically create
98     // nodes if necessary)
99     public: Node* addNode(std::shared_ptr<Game::GameState> state
100 );
101
102     // remove all nodes from the graph.
103     public: void reset();
104 };
105 }
106 }
107
108 inline std::vector<float> AlphaZero::ai::getQ(Node* node)
109 {
110     std::vector<float> data = jce::vector::gen(42, 0.0f);
111     for (auto const& pos : node->edges)
112     {
113         data[pos.first] = pos.second.Q;
114     }
115     return data;
116 }
117
118 inline void AlphaZero::ai::MCTS::addMCTSIter()
119 {
120     // this->MCTSIterMutex.lock();
121     MCTSIter++;
122     // this->MCTSIterMutex.unlock();
123 }

```

```

118
119 inline bool AlphaZero::ai::Node::isLeaf()
120 {
121     return this->edges.size() == 0;
122 }
123
124 inline void AlphaZero::ai::Node::addEdge(int id, Edge& edge)
125 {
126     // this->lock.lock();
127     this->edges.insert({ id, edge });
128     // this->lock.unlock();
129 }
130
131 inline AlphaZero::ai::MCTS::MCTS() {}
132
133 inline AlphaZero::ai::Node* AlphaZero::ai::MCTS::addNode(std::
    shared_ptr<Game::GameState> state)
134 {
135     if (this->MCTS_tree.count(state->id()) == 0) {
136
137         // this->NodeInserionMutex.lock();
138         this->MCTS_tree.insert({ state->id(), std::make_unique<Node
            >(state) });
139         // this->NodeInserionMutex.unlock();
140     }
141     return this->getNode(state->id());
142 }
143
144 inline void AlphaZero::ai::MCTS::reset()
145 {
146     // this->NodeInserionMutex.lock();
147     this->MCTS_tree.clear();
148     // this->NodeInserionMutex.unlock();
149 }
150
151 inline AlphaZero::ai::Node* AlphaZero::ai::MCTS::getNode(IDType
    key)
152 {
153     return this->MCTS_tree[key].get();
154 }
155
156
157 inline void AlphaZero::ai::Edge::traverse()
158 {
159     // this->inNode->lock.lock();

```

```
160     this->N++;  
161     // this->inNode->lock.unlock();  
162 }
```

#### 5.1.12.7.4 MCTS.cpp

```
1 #include "MCTS.hpp"
2 #include <limits>
3 #include <jce/vector.hpp>
4
5 AlphaZero::ai::Node::Node(std::shared_ptr<Game::GameState> state
6 )
7 {
8     this->state = state;
9 }
10 AlphaZero::ai::Edge::Edge(Node* _outNode, Node* _inNode, int
11     _action, float _p)
12 {
13     this->P = _p;
14     this->action = _action;
15     this->outNode = _outNode;
16     this->inNode = _inNode;
17     this->N = 0;
18     this->W = 0;
19     this->Q = 0;
20 }
21 AlphaZero::ai::Edge::Edge()
22 {
23     std::cout << "Edge default constructor" << std::endl;
24     return;
25 }
26
27 std::pair<AlphaZero::ai::Node*, std::list<AlphaZero::ai::Edge*>>
28     AlphaZero::ai::MCTS::moveToLeaf(Node* node)
29 {
30     std::list<Edge*> backTrackList;
31     while (true) {
32         if (node->isLeaf()) {
33             return { node, backTrackList };
34         }
35         else {
36             float U;
37             int Nb = 0;
38             for (auto const& iter : node->edges)
39             {
40                 Nb += iter.second.N;
```

```

41
42     Edge* opsEdge = &(node->edges.begin()->second);
43     int opsAction;
44
45     float maxQu;
46     bool nothasQU = true;
47
48     for (auto& iter : node->edges) {
49         U = U_computation(iter.second);
50         if (nothasQU || U + iter.second.Q > maxQu) {
51             opsEdge = &(iter.second);
52             opsAction = iter.first;
53             maxQu = U + iter.second.Q;
54             nothasQU = false;
55         }
56     }
57     opsEdge->traverse();
58     backTrackList.push_back(opsEdge);
59     node = opsEdge->outNode;
60 }
61 }
62 }
63
64 void AlphaZero::ai::MCIS::backFill(std::list<Edge*>& backTrace,
65     Node* leaf, float val)
66 {
67     float currentPlayer = (float)leaf->state->player * val;
68
69     for (auto const& edge : backTrace) {
70         // edge->inNode->lock.lock();
71
72         edge->W = edge->W + currentPlayer * (float)edge->inNode->
73             state->player;
74         edge->Q = edge->W / (float)edge->N;
75         // edge->inNode->lock.unlock();
76     }
77 }

```



### 5.1.12.7.5 memory.hpp

```
1 #pragma once
2 #include <iostream>
3 #include <fstream>
4 #include <sstream>
5
6 #include <queue>
7 #include <game/game.hpp>
8 #include "utils.hpp"
9 #include <jce/vector.hpp>
10
11
12 namespace AlphaZero {
13     namespace ai {
14         // class that represents a single Memory state
15         class MemoryElement {
16             // the MCIS computed value (value head)
17             public: int value;
18
19             // the game state the Memory element was built on.
20             public: std::shared_ptr<Game::GameState>state;
21
22             // the action values computed by the MCIS. (policy head)
23             public: std::vector<float> av;
24
25             // create a memory element around a game state and with
26             // certain action values.
27             public: MemoryElement(std::shared_ptr<Game::GameState>, std
28             ::vector<int>);
29             // create empty Element
30             public: MemoryElement();
31             };
32
33             // holds the memory elemnt created by a game. The Elements
34             // are heald by the
35             // temporary memory unitll the games winner is determined
36             // and the programm has
37             // computed the value for every elemnt in tmp memory.
38             class TemporaryMemory
39             {
40             public:
41                 // wether or not to write save elements (deactivate memory
42                 // for tourney)
43                 public: bool active;
```

```

39     // create temporary memory with active bool
40     public: TemporaryMemory( bool );
41
42     // vector used to save the Elements in.
43     public: std::vector<std::shared_ptr<MemoryElement>>
tempMemory;
44
45     // commit a game state and action value vector to memory
46     // (Create a Memory element and add it to the tempMemory
vector)
47     public: void commit( std::shared_ptr<Game::GameState>, std::
vector<int>& );
48     };
49
50     // full memory used to train the model and receives data
from
51     class Memory {
52         // mutex used to ensure that only one tmp memory is read
in at any given time
53         private: std::mutex mu;
54
55         // whether or not the Memory should actually save memory
Elements
56         public: bool active = true;
57
58         // build a temporary memory
59         public: TemporaryMemory getTempMemory();
60
61         // vector of all memory elements in the Memory
62         public: std::vector<std::shared_ptr<MemoryElement>> memory;
63
64         // constructor (does nothing)
65         public: Memory();
66
67         // perform memory update from temporary memory and set
every elements value correctly
68         public: void updateMemory( int player, int value,
TemporaryMemory* memory );
69
70         // get a random state from the Memory
71         public: std::shared_ptr<MemoryElement> getState();
72
73         // save Memory to file with a certain version
74         public: void save( int version );
75

```

```

76     // call save(-1)
77     public: void save();
78
79     // save to a specific file name
80     public: void save(char filename[]);
81
82     // load specific memory version
83     public: void load(int version);
84
85     // call load(-1)
86     public: void load();
87
88     // load from custom file path
89     public: void load(char filename[]);
90
91     // render all states in memory (debugging)(depricated)
92     public: void render();
93
94     // subroutine of the public updateMemory function (dose all
95     // the actuall work)
96     private: void updateMemory(int val, TemporaryMemory* memory)
97     ;
98     };
99 }
100
101 inline AlphaZero::ai::TemporaryMemory::TemporaryMemory(bool val)
102 {
103     this->active = val;
104 }
105
106 inline AlphaZero::ai::TemporaryMemory AlphaZero::ai::Memory::
107     getTempMemory()
108 {
109     AlphaZero::ai::TemporaryMemory memory(this->active);
110     return memory;
111 }
112
113 inline AlphaZero::ai::MemoryElement::MemoryElement(std::
114     shared_ptr<Game::GameState> _state, std::vector<int> _av)
115 {
116     this->state = _state;
117     float sum = (float)getSumm(_av);
118     this->av = jce::vector::gen(_av.size(), 0.0f);
119     for (size_t idx = 0; idx < this->av.size(); idx++)

```

```

117 {
118     float tmp = ((float)_av[idx]);
119     this->av[idx] = tmp / sum;
120 }
121 }
122 inline AlphaZero::ai::MemoryElement::MemoryElement()
123 {
124 }
125 inline AlphaZero::ai::Memory::Memory()
126 {
127 }
128 inline void AlphaZero::ai::TemporaryMemory::commit(std::
    shared_ptr<Game::GameState> state, std::vector<int>& av)
129 {
130     if (this->active) {
131         std::vector<std::pair<std::shared_ptr<Game::GameState>, std
            ::vector<int>>> idents = Game::identities(state, av);
132         for (auto const& data : idents) {
133             tempMemory.push_back(std::make_shared<MemoryElement>(data.
                first, data.second));
134         }
135     }
136 }
137
138 inline void AlphaZero::ai::Memory::updateMemory(int player, int
    value, TemporaryMemory* memory)
139 {
140     this->updateMemory(player * value, memory);
141 }
142
143 inline std::shared_ptr<AlphaZero::ai::MemoryElement> AlphaZero::
    ai::Memory::getState()
144 {
145     unsigned long long idx = rand() % this->memory.size();
146     std::shared_ptr<MemoryElement> element = this->memory[idx];
147     this->memory.erase(this->memory.begin() + idx);
148     return element;
149 }
150
151 inline void AlphaZero::ai::Memory::render()
152 {
153     for (auto const& element : this->memory)
154     {
155         element->state->render();
156     }

```

```

157 }
158
159 inline void AlphaZero::ai::Memory::updateMemory(int val,
    TemporaryMemory* memory)
160 {
161     while (memory->tempMemory.size() > 0) {
162         std::shared_ptr<MemoryElement>& element = memory->tempMemory
            .back();
163         element->value = element->state->player * val;
164         this->mu.lock();
165         this->memory.push_back(element);
166         this->mu.unlock();
167         memory->tempMemory.pop_back();
168     }
169 }

```

### 5.1.12.7.6 memory.cpp

```
1 #include "memory.hpp"
2 #include "config.hpp"
3 #include <jce/load.hpp>
4 #include <jce/save.hpp>
5
6 inline void getName(char out[], int version, int run)
7 {
8     sprintf(out, "memory/run-%d/V-%d.memory", run, version);
9 }
10
11 void AlphaZero::ai::Memory::save(int version)
12 {
13     char nameBuff[100];
14     getName(nameBuff, version, runVersion);
15     this->save(nameBuff);
16 }
17
18 void AlphaZero::ai::Memory::save()
19 {
20     this->save(-1);
21 }
22
23 void AlphaZero::ai::Memory::save(char filename[])
24 {
25     std::ofstream out(filename, std::ios::binary);
26     if (out.is_open())
27     {
28         jce::save(out, this->memory);
29         out.close();
30     }
31     else
32     {
33         throw "Game saver file not opened.";
34     }
35 }
36
37 void AlphaZero::ai::Memory::load(int version)
38 {
39     char nameBuff[100];
40     getName(nameBuff, version, runVersion);
41     this->load(nameBuff);
42 }
43
```

```

44 void AlphaZero::ai::Memory::load()
45 {
46     char nameBuff[100];
47     getName(nameBuff, -1, runVersion);
48     try{
49         std::cout << "Loading memory from file ... ";
50         this->load(nameBuff);
51         std::cout << "\33[1;32mSuccess\33[0m" << std::endl;
52     }
53     catch (...)
54     {
55         std::cout << "\33[1;31mFailed!\33[0m" << std::endl;
56     }
57 }
58
59 void AlphaZero::ai::Memory::load(char filename[])
60 {
61     std::ifstream in(filename, std::ios::binary);
62     if (in.is_open())
63     {
64         jce::load(in, this->memory);
65         in.close();
66     }
67     else
68     {
69         throw "Game saver file not opened.";
70     }
71 }

```

### 5.1.12.7.7 model.hpp

```
1 // TorchTestCMake.h : Include file for standard system include
   files ,
2 // or project specific include files .
3
4 #pragma once
5
6 #include <iostream>
7 #include <game/game.hpp>
8 #include "memory.hpp"
9 #include <string>
10 #include <tuple>
11 #include <jce/string.hpp>
12 #include <string>
13 #include <cmath>
14 #include "modelWorker.hpp"
15 #include <jce/save.hpp>
16 #include <jce/load.hpp>
17 #include <test/testUtils.hpp>
18
19
20 namespace AlphaZero {
21     namespace ai {
22         // class that implements the top layer wich consists of the
23         // following sub-layers:
24         // - Conv2d with kernel size of 3x3
25         // - LayerNorm
26         // - LeakyReLu
27         class TopLayer : public torch::nn::Module {
28         public: torch::nn::Conv2d conv1;
29         public: torch::nn::LayerNorm batch;
30         public: torch::nn::LeakyReLU relu;
31         private: int kernel1;
32
33         public: TopLayer(int inp, int out, int kernelsize1);
34         public: torch::Tensor forward(torch::Tensor);
35         public: void moveTo(c10::Device device);
36         };
37         // Residual layer consisting of the following sub-layers:
38         // - Conv2d with a 3x3 kernel
39         // - LayerNorm
40         // - LeakyReLU
41         class ResNet : public torch::nn::Module {
42         public: torch::nn::Conv2d conv1, conv2;
```



```

42     public: torch::nn::LayerNorm batch, batch2;
43     public: torch::nn::LeakyReLU activ;
44     private: int kernel1, kernel2;
45
46     public: ResNet(int inp, int out, int kernelsize1, int
kernelsize2);
47     public: torch::Tensor forward(torch::Tensor);
48     public: void moveTo(c10::Device device);
49     };
50
51     // value head layer that is used to compute the value of a
game state. It consists of the following sub-layers:
52     //     - Conv2D with kernel size 1x1
53     //     - LeakyReLU
54     //     - flatten
55     //     - Linear layer (lin1)
56     //     - LeakyRelu
57     //     - Linear Layer (lin2)
58     //     - tanh
59     class Value_head : torch::nn::Module {
60     private: bool isSecondRun = false;
61     private: torch::Tensor tmpX;
62
63     public: torch::nn::Conv2d conv;
64     public: torch::nn::Linear lin1, lin2;
65     public: torch::nn::LeakyReLU relu;
66     public: torch::nn::Tanh tanh;
67     private: int size;
68
69     public: Value_head(int inp, int hidden_size, int out, int
kernels);
70     public: torch::Tensor forward(torch::Tensor);
71     public: void moveTo(c10::Device device);
72     };
73
74     // policy head used to compute the policy of a certain state
. It consists of the following sub-layers
75     //     - Conv2d with a kernel size of 1x1
76     //     - LeakyReLU
77     //     - flatten
78     //     - Linear layer
79     class Policy_head : torch::nn::Module {
80     public: torch::nn::Conv2d conv;
81     public: torch::nn::Linear lin1;
82     public: torch::nn::LeakyReLU relu;

```

```

83     private: int size;
84
85     public: Policy_head(int inp, int hidden, int out);
86     public: torch::Tensor forward(torch::Tensor);
87     public: void moveTo(c10::Device device);
88 };
89
90     // some type definitions [...]
91     typedef torch::nn::MSELoss Loss;
92     typedef torch::optim::SGD Optimizer;
93     typedef torch::optim::SGDOptions OptimizerOptions;
94
95     // full model consisting of the following layers:
96     // - TopLayer (75 fillters)
97     // - 6 ResidualBlocks (75 fillters)
98     // *** the model splits into two heads (ValueHead and
99     PolicyHead**
100    // - ValueHead (10 fillters, lin1 size: 210, lin2 size: 1)
101
102    // - policyHead (2 fillters, linear size: 42)
103    class Model : public torch::nn::Module {
104    private: torch::nn::Conv2d headLayer;
105    private: TopLayer top;
106    private: ResNet res1, res2, res3, res4, res5, res6;
107    private: Value_head value_head;
108    private: Policy_head policy_head;
109
110    private: char* device;
111
112    private: Loss loss;
113    private: Optimizer optim;
114
115    public: Model(char* device);
116    public: std::pair<torch::Tensor, torch::Tensor> forward(
117    torch::Tensor);
118
119    // train the network to output a certain result
120    public: std::pair<float, float> train(const std::pair<torch
::Tensor, torch::Tensor>& x, const std::pair<torch::Tensor,
torch::Tensor>& y);
121
122    // pass game state though model (encode to tensor and pass
forward)
123    public: std::pair<float, torch::Tensor> predict(std::
shared_ptr<Game::GameState> state);

```

```

121
122     // get data batch from memory
123     public: static std::tuple<torch::Tensor, torch::Tensor,
torch::Tensor> getBatch(std::shared_ptr<Memory> memory,
unsigned int batchSize);
124
125     // forward pass with model data
126     public: void predict(ModelData* data);
127
128     // forward pass with multiple model data containers
129     public: void predict(std::list<ModelData*> data);
130
131     // forward though network than call train
132     public: void fit(const std::tuple<torch::Tensor, torch::
Tensor, torch::Tensor>& batch, const unsigned short& run,
const unsigned short& trainingLoop);
133
134     // save version of model
135     public: void save_version(unsigned int version);
136
137     // save a current version
138     public: void save_as_current();
139
140     // save to a specific file
141     public: void save_to_file(char* filename);
142     // alternative jce save
143     public: void jce_save_current(char* filename);
144
145     // load specific model version
146     public: void load_version(unsigned int version);
147
148     // load current model
149     public: void load_current();
150
151     // load model from given file
152     public: void load_from_file(char* filename);
153
154     // alternative jce loading method
155     public: void jce_load_from_file(char* filename);
156
157     // copy model into this one
158     public: void copyModel(Model*);
159     private: void copyParameters(torch::OrderedDict<std::string,
torch::Tensor> prams);
160

```

```

161     // move model to diferant device
162     public: void moveTo(c10::Device device);
163
164
165     // functions used to add custom modules to this graph.
166     private: TopLayer register_custom_module(TopLayer net);
167     private: ResNet register_custom_module(ResNet net, std::
string layer);
168     private: Value_head register_custom_module(Value_head net);
169     private: Policy_head register_custom_module(Policy_head net)
;
170
171     };
172 }
173 }
174 // customizable section
175 #define modelTest false
176 #define randomModel false
177 #define convSize 5
178
179 inline AlphaZero::ai::Model::Model(char* _device) :
180     top(this->register_custom_module(TopLayer(2, 75, convSize))),
181     res1(this->register_custom_module(ResNet(75, 75, convSize,
convSize, "Residual_1"))),
182     res2(this->register_custom_module(ResNet(75, 75, convSize,
convSize, "Residual_2"))),
183     res3(this->register_custom_module(ResNet(75, 75, convSize,
convSize, "Residual_3"))),
184     res4(this->register_custom_module(ResNet(75, 75, convSize,
convSize, "Residual_4"))),
185     res5(this->register_custom_module(ResNet(75, 75, convSize,
convSize, "Residual_5"))),
186     res6(this->register_custom_module(ResNet(75, 75, convSize,
convSize, "Residual_6"))),
187     value_head(this->register_custom_module(Value_head(75, 420,
210, 10))),
188     policy_head(this->register_custom_module(Policy_head(75, 84,
42))),
189     optim(Optimizer(this->parameters(), OptimizerOptions(
learningRate).momentum(Momentum))),
190     device(_device)
191 {
192     this->moveTo(c10::Device(_device));
193 }
194

```

```

195 inline std::pair<torch::Tensor, torch::Tensor> AlphaZero::ai::
    Model::forward(torch::Tensor x)
196 {
197     #if randomModel
198         return { torch::rand({x.size(0), 1}), torch::rand({x.size(0),
            action_count}) } };
199     #else
200         x = this->top.forward(x);
201         x = this->res1.forward(x);
202         x = this->res2.forward(x);
203         x = this->res3.forward(x);
204         x = this->res4.forward(x);
205         x = this->res5.forward(x);
206         x = this->res6.forward(x);
207
208         // compute individual heads
209         torch::Tensor value = this->value_head.forward(x.clone());
210         torch::Tensor poly = this->policy_head.forward(x.clone());
211
212         return { value, poly };
213     #endif
214 };
215 // end of cutimizable section
216
217 inline AlphaZero::ai::TopLayer::TopLayer(int inp, int out, int
    kernelsize1) :
218     conv1(this->register_module("conv1", torch::nn::Conv2d(torch::
        nn::Conv2dOptions(inp, out, kernelsize1)))),
219     batch(this->register_module("batch", torch::nn::LayerNorm(
        torch::nn::LayerNormOptions({ out, input_shape_y,
            input_shape_x }))),
220     relu(this->register_module("ReLU", torch::nn::LeakyReLU(torch
        ::nn::LeakyReLU()))),
221     kernell(kernelsize1 / 2)
222 {
223 }
224 inline torch::Tensor AlphaZero::ai::TopLayer::forward(torch::
    Tensor x)
225 {
226     x = torch::nn::functional::pad(x, torch::nn::functional::
        PadFuncOptions({ kernell, kernell, kernell, kernell }));
227     x = this->conv1(x);
228     x = this->batch(x);
229     x = this->relu(x);
230     return x;

```

```

231 }
232
233 inline void AlphaZero::ai::TopLayer::moveTo(c10::Device device)
234 {
235     this->conv1->to(device, true);
236     this->batch->to(device, true);
237     this->relu->to(device, true);
238 }
239
240 inline AlphaZero::ai::ResNet::ResNet(int inp, int out, int
    kernelsize1, int kernelsize2) :
241     kernel1(kernelsize1), kernel2(kernelsize2),
242     conv1(this->register_module("conv1", torch::nn::Conv2d(torch::
        nn::Conv2dOptions(inp, out, kernelsize1)))),
243     conv2(this->register_module("conv2", torch::nn::Conv2d(torch::
        nn::Conv2dOptions(out, out, kernelsize2)))),
244     batch(this->register_module("batch1", torch::nn::LayerNorm(
        torch::nn::LayerNormOptions({ out, input_shape_y,
        input_shape_x })))),
245     batch2(this->register_module("batch2", torch::nn::LayerNorm(
        torch::nn::LayerNormOptions({ out, input_shape_y,
        input_shape_x })))),
246     activ(this->register_module("activ", torch::nn::LeakyReLU(
        torch::nn::LeakyReLU()))))
247 {
248     if (torch::cuda::is_available()) {
249         this->moveTo(c10::Device("cuda:0"));
250     }
251 }
252
253 inline torch::Tensor AlphaZero::ai::ResNet::forward(torch::
    Tensor x)
254 {
255     #if modelTest
256         std::cout << x.sizes() << std::endl;
257     #endif
258     auto y = x.clone();
259     x = torch::nn::functional::pad(x, torch::nn::functional::
        PadFuncOptions({ kernel1 / 2, kernel1 / 2, kernel1 / 2,
        kernel1 / 2 }));
260     x = this->conv1(x);
261     x = this->batch(x);
262     x = this->activ(x);
263
264     x = torch::nn::functional::pad(x, torch::nn::functional::

```

```

        PadFuncOptions({ kernel2 / 2, kernel2 / 2, kernel2 / 2,
        kernel2 / 2 }));
265 x = this->conv2(x);
266 x = this->batch2(x);
267 return this->activ(x + y);
268 }
269
270 inline void AlphaZero::ai::ResNet::moveTo(c10::Device device)
271 {
272     this->conv1->to(device, true);
273     this->conv2->to(device, true);
274     this->batch->to(device, true);
275     this->batch2->to(device, true);
276     this->activ->to(device, true);
277 }
278
279 inline AlphaZero::ai::Value_head::Value_head(int inp, int
        hidden_size, int out, int convOut) :
280     conv(this->register_module("conv", torch::nn::Conv2d(torch::nn
        ::Conv2dOptions(inp, convOut, 1)))),
281     lin1(this->register_module("lin1", torch::nn::Linear(torch::nn
        ::LinearOptions(hidden_size, out)))),
282     lin2(this->register_module("lin2", torch::nn::Linear(torch::nn
        ::LinearOptions(out, 1))),
283     relu(this->register_module("relu", torch::nn::LeakyReLU())),
284     tanh(this->register_module("tanh", torch::nn::Tanh()))
285 {
286     this->size = hidden_size;
287
288     if (torch::cuda::is_available())
289     {
290         this->moveTo(c10::Device("cuda:0"));
291     }
292 }
293
294 inline torch::Tensor AlphaZero::ai::Value_head::forward(torch::
        Tensor x)
295 {
296     #if modelTest
297         std::cout << "value" << std::endl;
298         std::cout << x.sizes() << std::endl;
299     #endif
300     x = this->conv(x);
301     x = this->relu(x);
302     x = this->lin1(x.reshape({ x.size(0), this->size }));

```

```

303     x = this->relu(x);
304     x = this->lin2(x);
305     x = this->tanh(x);
306     return x;
307 }
308
309 inline void AlphaZero::ai::Value_head::moveTo(c10::Device device
310 )
311 {
312     this->conv->to(device, true);
313     this->lin1->to(device, true);
314     this->relu->to(device, true);
315     this->lin2->to(device, true);
316     this->tanh->to(device, true);
317 }
318 inline AlphaZero::ai::Policy_head::Policy_head(int inp, int
319     hidden, int out) :
320     conv(this->register_module("conv", torch::nn::Conv2d(torch::nn
321     ::Conv2dOptions(inp, 2, 1))),
322     lin1(this->register_module("lin1", torch::nn::Linear(hidden,
323     out))),
324     relu(this->register_module("relu", torch::nn::LeakyReLU()))
325 {
326     this->size = hidden;
327
328     if (torch::cuda::is_available()) {
329         this->moveTo(c10::Device("cuda:0"));
330     }
331 }
332
333 inline torch::Tensor AlphaZero::ai::Policy_head::forward(torch::
334     Tensor x)
335 {
336     #if modelTest
337         std::cout << "poly" << std::endl;
338         std::cout << x.sizes() << std::endl;
339     #endif
340     x = this->conv(x);
341     x = this->relu(x);
342     x = this->lin1(x.reshape({ x.size(0), this->size }));
343     return x;
344 }
345
346 inline void AlphaZero::ai::Policy_head::moveTo(c10::Device

```



```

        device)
343 {
344     this->conv->to(device, true);
345     this->lin1->to(device, true);
346 }
347
348 inline torch::Tensor polyLoss(torch::Tensor a, torch::Tensor b)
349 {
350     #if true
351         auto c = torch::where(b == 0, a, b);
352         return torch::mse_loss(a, c);
353     #else
354         return torch::mse_loss(a, b);
355     #endif
356 }
357
358 inline std::pair<float, float> AlphaZero::ai::Model::train(const
    std::pair<torch::Tensor, torch::Tensor>& x, const std::pair<
    torch::Tensor, torch::Tensor>& y)
359 {
360     //std::cout << x.first << std::endl << y.first << std::endl;
361
362     auto valLoss = 0.5f * torch::mse_loss(x.first, y.first);
363     auto plyLoss = 0.5f * polyLoss(x.second, y.second);
364     auto loss = (valLoss + plyLoss);
365
366     loss.backward();
367     this->optim.step();
368     this->optim.zero_grad();
369     std::pair<float, float> error = { torch::mean(valLoss).item().
        toFloat(), torch::mean(plyLoss).item().toFloat() };
370
371     if (std::isnan(error.first))
372     {
373         //std::cout << valLoss << std::endl << plyLoss << std::endl;
374         std::cout << x.first << std::endl << y.first << std::endl;
375         std::cout << x.second << std::endl << y.second << std::endl;
376         return error;
377     }
378
379     return error;
380 }
381
382 inline std::pair<float, torch::Tensor> AlphaZero::ai::Model::
    predict(std::shared_ptr<Game::GameState> state)

```

```

383 {
384     torch::Tensor NNInput = state->toTensor().to(c10::Device(this
        ->device));
385     std::pair<torch::Tensor, torch::Tensor> NNOut = this->forward(
        NNInput);
386     float value = NNOut.first[0].item<float>();
387     return { value, NNOut.second };
388 }
389
390 inline void AlphaZero::ai::Model::predict(ModelData* data)
391 {
392     torch::Tensor NNInput = data->node->state->toTensor().to(c10::
        Device(this->device));
393     std::pair<torch::Tensor, torch::Tensor> NNOut = this->forward(
        NNInput);
394
395     torch::Tensor mask = torch::ones(
396         { 1, action_count },
397         c10::TensorOptions().device(c10::Device("cpu")).dtype(at::
        kBool)
398     );
399
400     for (auto idx : data->node->state->allowedActions)
401     {
402         mask[0][idx] = false;
403     }
404     //std::cout << std::endl << NNOut.first << std::endl << NNOut.
        second << std::endl;
405     data->value = NNOut.first[0].item<float>();
406     data->polys = torch::softmax(torch::masked_fill(NNOut.second.
        cpu(), mask, -1000.0f), 1)[0];
407 }
408
409 inline void AlphaZero::ai::Model::predict(std::list<ModelData*>
    data)
410 {
411     torch::Tensor NNInput = torch::zeros({ (int)data.size(),
        input_snape_z, input_shape_y, input_shape_x });
412     torch::Tensor mask = torch::ones(
413         { (int)data.size(), action_count },
414         c10::TensorOptions().device(c10::Device("cpu")).dtype(at::
        kBool)
415     );
416     auto iter = data.begin();
417     for (unsigned short idx = 0; idx < data.size(); idx++)

```

```

418 {
419     NNInput[idx] = (*iter)->node->state->toTensor()[0];
420     for (auto action : (*iter)->node->state->allowedActions)
421     {
422         mask[idx][action] = false;
423     }
424     iter++;
425 }
426
427 std::pair<torch::Tensor, torch::Tensor> NNOut = this->forward(
    NNInput.to(c10::Device(this->device)));
428
429 //std::cout << std::endl << NNOut.first << std::endl << NNOut.
    second << std::endl;
430 mask = mask.to(c10::Device(this->device));
431
432 auto soft = torch::softmax(torch::masked_fill(NNOut.second,
    mask, -1000.0f), 1).cpu();
433
434 iter = data.begin();
435 for (unsigned int idx = 0; idx < data.size(); idx++)
436 {
437     (*iter)->value = NNOut.first[idx].item<float>();
438     (*iter)->polys = soft[idx];
439     iter++;
440 }
441 }
442
443 inline std::tuple<torch::Tensor, torch::Tensor, torch::Tensor>
    AlphaZero::ai::Model::getBatch(std::shared_ptr<Memory> memory
    , unsigned int batchSize)
444 {
445     std::tuple<torch::Tensor, torch::Tensor, torch::Tensor> output
    =
446     {
447         at::zeros({batchSize, input_shape_z, input_shape_y,
            input_shape_x}),
448         at::zeros({batchSize, action_count}),
449         at::zeros({batchSize, 1})
450     };
451     for (unsigned short idx = 0; idx < batchSize; idx++) {
452         auto state = memory->getState();
453         state->state->toTensor(std::get<0>(output), idx);
454         std::get<1>(output)[idx] = at::from_blob(state->av.data(), {
            action_count }).toType(torch::kFloat16);

```

```

455     std::get<2>(output)[idx] = torch::tensor({ state->value });
456 }
457 return output;
458 }
459
460 inline void AlphaZero::ai::Model::fit(const std::tuple<torch::
    Tensor, torch::Tensor, torch::Tensor>& batch, const unsigned
    short& run, const unsigned short& trainingLoop)
461 {
462     std::pair<torch::Tensor, torch::Tensor> NNVals = this->forward
        (std::get<0>(batch).to(c10::Device(this->device)));
463     std::pair<float, float> error = this->train(NNVals,
464         {
465             std::get<2>(batch).to(c10::Device(this->device)),
466             std::get<1>(batch).to(c10::Device(this->device))
467         });
468     #if ModelLogger
469         debug::log::modelLogger->info("model error in iteration {} on
            batch {} had valueError of {} and polyError of {}", run,
            trainingLoop, std::get<0>(error), std::get<1>(error));
470     #endif
471     #if LossLogger
472         debug::log::lossLogger.addValue(error);
473     #endif
474 }
475
476 inline void AlphaZero::ai::Model::save_version(unsigned int
    version)
477 {
478     char buffer[50];
479     std::sprintf(buffer, "models/run_%d/V_%d.torch", runVersion,
        version);
480     std::cout << buffer << std::endl;
481     this->save_to_file(buffer);
482 }
483
484 inline void AlphaZero::ai::Model::save_as_current()
485 {
486     char buffer[50];
487     std::sprintf(buffer, "models/run_%d/currentModel.torch",
        runVersion);
488     this->save_to_file(buffer);
489 }
490
491 inline void AlphaZero::ai::Model::save_to_file(char* filename)

```

```

492 {
493     torch::serialize::OutputArchive out;
494     this->save(out);
495     std::string model_path = std::string(filename);
496     out.save_to(model_path);
497 }
498
499 inline void AlphaZero::ai::Model::jce_save_current(char*
        filename)
500 {
501     std::ofstream out(filename, std::ios::binary);
502     jce::save(out, this->named_parameters(true));
503     out.close();
504 }
505
506
507 inline void AlphaZero::ai::Model::load_version(unsigned int
        version)
508 {
509     std::cout << "loading ...";
510     char buffer[50];
511     std::sprintf(buffer, "models/run-%d/V-%d.torch", runVersion,
        version);
512     this->load_from_file(buffer);
513     std::cout << " loaded Version " << version << std::endl;
514 }
515
516 inline void AlphaZero::ai::Model::load_current()
517 {
518     char buffer[50];
519     std::sprintf(buffer, "models/run-%d/currentModel.torch",
        runVersion);
520     this->load_from_file(buffer);
521 }
522
523 inline void AlphaZero::ai::Model::load_from_file(char* filename)
524 {
525     torch::serialize::InputArchive inp;
526     std::string model_path = std::string(filename);
527     inp.load_from(model_path);
528     this->load(inp);
529 }
530
531 inline void AlphaZero::ai::Model::jce_load_from_file(char*
        filename)

```

```

532 {
533     std::cout << "loading ...\t";
534     torch::autograd::GradMode::set_enabled(false);
535     torch::OrderedDict<std::string, torch::Tensor> map;
536     std::ifstream in(filename, std::ios::binary);
537     if (in.is_open())
538     {
539         jce::load(in, map);
540         this->copyParameters(map);
541     }
542     test::printSuccess(in.is_open());
543     in.close();
544     torch::autograd::GradMode::set_enabled(true);
545 }
546
547 inline void AlphaZero::ai::Model::copyModel(AlphaZero::ai::Model
548 * model)
549 {
550     torch::autograd::GradMode::set_enabled(false);
551     auto new_params = model->named_parameters(true);
552     this->copyParameters(new_params);
553     torch::autograd::GradMode::set_enabled(true);
554 }
555 inline void AlphaZero::ai::Model::copyParameters(torch::
556 OrderedDict<std::string, torch::Tensor> new_params)
557 {
558     auto params = this->named_parameters(true);
559     auto buffers = this->named_buffers(true);
560     for (auto& val : new_params) {
561         auto name = val.key();
562         auto* t = params.find(name);
563         if (t != nullptr) {
564             t->copy_(val.value());
565         }
566         else {
567             t = buffers.find(name);
568             if (t != nullptr) {
569                 t->copy_(val.value());
570             }
571         }
572     }
573 }
574 inline void AlphaZero::ai::Model::moveTo(c10::Device device)

```

```

575 {
576     this->top.moveTo(device);
577
578     this->res1.moveTo(device);
579     this->res2.moveTo(device);
580     this->res3.moveTo(device);
581     this->res4.moveTo(device);
582     this->res5.moveTo(device);
583     this->res6.moveTo(device);
584
585     this->value_head.moveTo(device);
586     this->policy_head.moveTo(device);
587 }
588
589 inline AlphaZero::ai::TopLayer AlphaZero::ai::Model::
    register_custom_module(TopLayer net)
590 {
591     register_module("TopLayer_conv", net.conv1);
592     register_module("TopLayer_batch", net.batch);
593     register_module("TopLayer_ReLU", net.relu);
594     return net;
595 }
596
597 inline AlphaZero::ai::ResNet AlphaZero::ai::Model::
    register_custom_module(ResNet net, std::string layer)
598 {
599     register_module(layer + "_conv1", net.conv1);
600     register_module(layer + "_conv2", net.conv2);
601     register_module(layer + "_batch1", net.batch);
602     register_module(layer + "_batch2", net.batch2);
603     register_module(layer + "_active", net.activ);
604     return net;
605 }
606 inline AlphaZero::ai::Value_head AlphaZero::ai::Model::
    register_custom_module(Value_head net)
607 {
608     register_module("value_conv", net.conv);
609     register_module("value_lin1", net.lin1);
610     register_module("value_lin2", net.lin2);
611     register_module("value_ReLU", net.relu);
612     register_module("value_tanh", net.tanh);
613     return net;
614 }
615 inline AlphaZero::ai::Policy_head AlphaZero::ai::Model::
    register_custom_module(Policy_head net)

```

```
616 {  
617     register_module("policy_conv", net.conv);  
618     register_module("policy_linear", net.lin1);  
619     return net;  
620 }
```



### 5.1.12.7.8 modelSynchronizer.hpp

```
1 #pragma once
2
3 #include <mutex>
4 #include <thread>
5 #include <thread>
6 #include <iostream>
7 #include <memory>
8 #include "model.hpp"
9
10 // this is compleatly useless is it not.
11
12 namespace AlphaZero
13 {
14     namespace ai
15     {
16         // interface between model.hpp and agent.hpp. It handles
17         // synchronization of models over multiple devices.
18         class ModelSynchronizer
19         {
20             // array containing the actual models
21             private: std::vector<std::unique_ptr<Model>> models;
22
23             // model index iteratror.
24             private: unsigned short pos = 0;
25
26             // mutex used to split the threads over the devices.
27             private: std::mutex modelGetMutex;
28
29             // create models on devices and generate synchronizer.
30             public: ModelSynchronizer(std::vector<char*> devices);
31
32             // evaluate Model data
33             public: void addData(ModelData* data);
34
35             // get the model to perform the prediction
36             private: Model* getModel();
37
38             // copy model[0] from other synchronizer and synchronize
39             // its models.
40             public: void copyModel(ModelSynchronizer*);
41
42             // trains models[0] and updatas the others the same way.
```

```

42     public: void fit(const std::tuple<torch::Tensor, torch::
Tensor, torch::Tensor>& batch, const unsigned short& run,
const unsigned short& trainingLoop);
43
44     // save models[0] as current
45     public: void save_as_current();
46
47     // save models[0] under a certain version
48     public: void save_version(unsigned int version);
49
50     // save models[0] to a specific file
51     public: void save_to_file(char* filename);
52
53     // save models[0] as jce format
54     public: void jce_save_current(char* filename);
55
56     // load current model file and call synchronizeModels
57     public: void load_current();
58
59     // load model by version from file and call
synchronizeModels
60     public: void load_version(unsigned int version);
61
62     // load model from custom file file and call
synchronizeModels
63     public: void load_from_file(char* filename);
64
65     // load model file in jce format and call
synchronizeModels
66     public: void jce_load_from_file(char* filename);
67
68     // sets the parameters of all models to that of models[0]
69     public: void synchronizeModels();
70
71     // generates the model data and calls addData for
evaluation
72     public: std::pair<float, torch::Tensor> predict(std::
shared_ptr<Game::GameState> state, size_t idx=0);
73     public: void predict(ModelData* data, size_t idx=0);
74     public: void predict(std::list<ModelData*> data, size_t idx
=0);
75     };
76 }
77 namespace test
78 {

```

```

79     namespace ModelSynchronizer
80     {
81         std::thread addTestData(ai::ModelData* data, ai::
ModelSynchronizer* sync);
82         void _addTestData(ai::ModelData* data, ai::
ModelSynchronizer* sync);
83     }
84 }
85 }
86
87 inline AlphaZero::ai::ModelSynchronizer::ModelSynchronizer(std::
vector<char*> devices)
88 {
89     for (auto const& device : devices)
90     {
91         this->models.push_back(std::make_unique<Model>(device));
92     }
93     this->synchronizeModels();
94 }
95 inline void AlphaZero::ai::ModelSynchronizer::addData(ModelData*
_data)
96 {
97     /*_data->value = 2;
98     std::list<ModelData*>data_l;
99     data_l.push_back(_data);*/
100     this->getModel()->predict(_data);
101 }
102 inline AlphaZero::ai::Model* AlphaZero::ai::ModelSynchronizer::
getModel()
103 {
104     this->modelGetMutex.lock();
105     auto outputModel = this->models[this->pos].get();
106     this->pos++;
107     if (this->pos >= this->models.size())
108     {
109         this->pos = 0;
110     }
111     this->modelGetMutex.unlock();
112     return outputModel;
113 }
114 inline void AlphaZero::ai::ModelSynchronizer::copyModel(
ModelSynchronizer* syncher)
115 {
116     for (auto const& model : this->models)
117     {

```

```

118     model->copyModel(syncher->models[0].get());
119 }
120 }
121 inline void AlphaZero::ai::ModelSynchronizer::fit(const std::
    tuple<torch::Tensor, torch::Tensor, torch::Tensor>& batch,
    const unsigned short& run, const unsigned short& trainingLoop
    )
122 {
123     this->models[0]->fit(batch, run, trainingLoop);
124 }
125 inline void AlphaZero::ai::ModelSynchronizer::save_as_current()
126 {
127     this->models[0]->save_as_current();
128 }
129 inline void AlphaZero::ai::ModelSynchronizer::save_to_file(char*
    filename)
130 {
131     this->models[0]->save_to_file(filename);
132 }
133 inline void AlphaZero::ai::ModelSynchronizer::jce_save_current(
    char* filename)
134 {
135     this->models[0]->jce_save_current(filename);
136 }
137 inline void AlphaZero::ai::ModelSynchronizer::save_version(
    unsigned int version)
138 {
139     this->models[0]->save_version(version);
140 }
141 inline void AlphaZero::ai::ModelSynchronizer::load_current()
142 {
143     this->models[0]->load_current();
144     this->synchronizeModels();
145 }
146 inline void AlphaZero::ai::ModelSynchronizer::load_from_file(
    char* filename)
147 {
148     this->models[0]->load_from_file(filename);
149     this->synchronizeModels();
150 }
151 inline void AlphaZero::ai::ModelSynchronizer::jce_load_from_file
    (char* filename)
152 {
153     this->models[0]->jce_load_from_file(filename);
154 }

```

```

155 inline void AlphaZero::ai::ModelSynchronizer::load_version(
    unsigned int version)
156 {
157     this->models[0]->load_version(version);
158     this->synchronizeModels();
159 }
160 inline void AlphaZero::ai::ModelSynchronizer::synchronizeModels
    ()
161 {
162     auto copyFrom = this->models[0].get();
163     for (auto const& model : this->models)
164     {
165         if (model.get() != copyFrom)
166         {
167             model->copyModel(copyFrom);
168         }
169     }
170 }
171 inline void AlphaZero::ai::ModelSynchronizer::predict(ModelData*
    data, size_t idx)
172 {
173     this->models[idx]->predict(data);
174 }
175 inline void AlphaZero::ai::ModelSynchronizer::predict(std::list<
    ModelData*> data, size_t idx)
176 {
177     this->models[idx]->predict(data);
178 }
179 inline std::pair<float, torch::Tensor> AlphaZero::ai::
    ModelSynchronizer::predict(std::shared_ptr<Game::GameState>
    state, size_t idx)
180 {
181     return this->models[idx]->predict(state);
182 }
183
184
185 inline std::thread AlphaZero::test::ModelSynchronizer::
    addTestData(ai::ModelData* data, ai::ModelSynchronizer* sync)
186 {
187     std::thread thread(_addTestData, data, sync);
188     return thread;
189 }
190
191 inline void AlphaZero::test::ModelSynchronizer::_addTestData(ai
    ::ModelData* data, ai::ModelSynchronizer* sync)

```

```
192 {  
193   sync->addData( data );  
194 }
```

### 5.1.12.7.9 modelWorker.hpp

```
1 #pragma once
2
3 #include "MCTS.hpp"
4
5 // also useless
6
7 namespace AlphaZero
8 {
9     namespace ai
10    {
11        class Node;
12
13        class ModelData
14        {
15        public: Node* node;
16        public: torch::Tensor polys;
17        public: float value;
18
19        public: ModelData(Node* node);
20        public: void print();
21        };
22    }
23 }
24
25 inline AlphaZero::ai::ModelData::ModelData(Node* _node)
26 {
27     this->node = _node;
28 }
29
30 inline void print() {}
```

#### 5.1.12.7.10 modelWorker.cpp

```
1 #include "modelWorker.hpp"
2 #include "MCTS.hpp"
3
4 /*
5 void AlphaZero::ai::ModelData::print()
6 {
7     std::cout << "state: " << std::endl;
8     this->node->state->render();
9     std::cout << "polys: " << std::endl;
10    std::cout << torch::reshape(this->polys, { action_shape }) <<
        std::endl;
11    std::cout << "value: " << this->value << std::endl;
12 }*/
```



### 5.1.12.7.11 playGame.hpp

```
1 #pragma once
2
3 #include <ai/agent.hpp>
4 #include <ai/memory.hpp>
5
6
7 namespace AlphaZero {
8     namespace ai {
9         struct gameOutput
10         {
11
12             // map that counts how often a player has won.
13             std::unordered_map<Agent*, int> map;
14             // mutex used updating 'map' from multiple threads
15             std::mutex ex;
16
17             // create a gameOutput object for games between tow
18             agents
19             gameOutput(Agent*, Agent*);
20             // add a win for a certain player.
21             void updateValue(Agent*);
22         };
23         // main training loop
24         // create agents, generate data though self play, train an
25         agent, and evaluate the next one. (save if better)
26         void train(int);
27         // play a certain amount of games between two (can be the
28         same one) and record the actionValues and states.
29         void playGames(gameOutput* output, Agent* agent1, Agent*
30         agent2, Memory* memory, int probMoves, int Epochs, char RunId
31         [], int goesFirst = 0, bool log = false);
32         // create the various threads the games run in and collect
33         the data at the end.
34         std::unordered_map<Agent*, int> playGames_inThreads(Game::
35         Game* game, Agent* agent1, Agent* agent2, Memory* memory, int
36         probMoves, int Epochs, int Threads, char RunId[], int
37         goesFirst = 0, bool log = false);
38     }
39 }
40
41 namespace test {
42     // play test game
43     void playGame(std::shared_ptr<Game::Game> game, std::
44     shared_ptr<ai::Agent> player1, std::shared_ptr<ai::Agent>
45     player2, int goesFirst=0);
46 }
```

```

33 }
34 }
35
36 inline std::unordered_map<AlphaZero::ai::Agent*, int> AlphaZero
    ::ai::playGames_inThreads(Game::Game* game, Agent* agent1,
        Agent* agent2, Memory* memory, int probMoves, int Epochs, int
        Threads, char RunId[], int goesFirst, bool log)
37 {
38     gameOutput output(agent1, agent2);
39
40     std::vector<std::thread> workers;
41     for (size_t idx = 0; idx < Threads; idx++)
42     {
43         bool doLog = (log && (idx == 0));
44         workers.push_back(std::thread(playGames, &output, agent1,
            agent2, memory, probMoves, Epochs, RunId, goesFirst, doLog));
45     }
46
47     for (auto& worker : workers)
48     {
49         worker.join();
50     }
51
52     return output.map;
53 }
54
55 inline void AlphaZero::ai::gameOutput::updateValue(Agent* idx)
56 {
57     this->ex.lock();
58     this->map[idx] = this->map[idx] + 1;
59     this->ex.unlock();
60 }
61
62 inline AlphaZero::ai::gameOutput::gameOutput(Agent* agent1,
        Agent* agent2)
63 {
64     this->map.insert({ agent1, 0 });
65     this->map.insert({ agent2, 0 });
66 }

```

### 5.1.12.7.12 playGame.cpp

```
1 #include "playGame.hpp"
2 #include <jce/load.hpp>
3 #include <jce/save.hpp>
4 #include <io.hpp>
5
6
7 void AlphaZero::test::playGame(std::shared_ptr<Game::Game> game,
8     std::shared_ptr<ai::Agent> player1, std::shared_ptr<ai::
9     Agent> player2, int goesFirst)
10 {
11     if (goesFirst == 0) {
12         goesFirst = 1;
13         if (rand() % 2) {
14             goesFirst = -1;
15         }
16     }
17     player1->reset();
18     player2->reset();
19     std::unordered_map<int, std::shared_ptr<ai::Agent>> players =
20         { {goesFirst, player1}, {-goesFirst, player2} };
21     int action;
22     while (!game->state->done) {
23         action = players[game->state->player]->getAction(game->state
24             , false).first;
25         game->takeAction(action);
26     }
27 }
28
29 void AlphaZero::ai::train(int version)
30 {
31     unsigned short iteration = 0;
32     std::vector<char*> devices = { DEVICES };
33     std::shared_ptr<Memory> memory = std::make_shared<Memory>();
34     std::shared_ptr<Game::Game> game = std::make_shared<Game::Game>
35         >();
36     std::shared_ptr<Agent> currentAgent = std::make_shared<Agent>(
37         devices);
38     std::shared_ptr<Agent> bestAgent = std::make_shared<Agent>(
39         devices);
40
41     std::vector<int> requiredIterations;
42
43     memory->load();
```

```

37 char nameBuff[100];
38
39 std::sprintf(nameBuff, "models/run_%d/versionCount.jce",
40               runVersion);
41 std::ifstream fin(nameBuff, std::ios::binary);
42 if (fin.is_open())
43 {
44     jce::load(fin, version);
45     std::cout << "found model version: " << version << std::endl
46     ;
47
48     fin.close();
49
50     std::sprintf(nameBuff, "models/run_%d/iterationCounter.jce",
51                 runVersion);
52     fin.open(nameBuff, std::ios::binary);
53     if (fin.is_open())
54     {
55         jce::load(fin, requiredIterations);
56         std::cout << "loaded required Iterations: it hs size: " <<
57         requiredIterations.size() << std::endl;
58     }
59     else
60     {
61         std::cout << "could not find sutalbe iterationCounter";
62     }
63     fin.close();
64
65     bestAgent->model->load_version(version);
66     currentAgent->model->load_version(version);
67 }
68 else
69 {
70     std::cout << "model version config not found. Defaulting to
71     0" << std::endl;
72     version = 0;
73 }
74 currentAgent->model->copyModel(bestAgent->model.get());
75
76 while (true) {
77     iteration++;
78     memory->active = true;
79 #if MainLogger
80     debug::log::mainLogger->info("playing version: {}", version)
81     ;

```

```

76 #endif
77
78     std::cout << "playing Generational Games:" << std::endl;
79
80 #if ModelLogger
81     debug::log::modelLogger->info("Running Training Games");
82 #endif
83
84     sprintf(nameBuff, "logs/games/game-%d-Generator.gameLog",
85             iteration);
86     playGames_inThreads(game.get(), bestAgent.get(), bestAgent.
87                         get(), memory.get(), probabilistic_moves, EPOCHS, GEN_THREADS,
88                         nameBuff, 1, false);
89     std::cout << "memory size is: " << memory->memory.size() <<
90     std::endl;
91     if (memory->memory.size() > memory_size) {
92 #if ProfileLogger
93     debug::Profiler::profiler.switchOperation(5);
94 #endif
95     currentAgent->fit(memory, iteration);
96 #if ProfileLogger
97     debug::Profiler::profiler.stop();
98 #endif
99     memory->active = false;
100     std::cout << "playing Tournament Games:" << std::endl;
101 #if MainLogger
102     debug::log::mainLogger->info("RETRAINING
103     =====");
104 #endif
105     sprintf(nameBuff, "logs/games/game-%d-Turney.gameLog",
106             iteration);
107 #if ModelLogger
108     debug::log::modelLogger->info("Running Tourney Games");
109 #endif
110     auto score = playGames_inThreads(game.get(), bestAgent.get
111                                     (), currentAgent.get(), memory.get(),
112                                     Turnement_probabilisticMoves, TurneyEpochs, TurneyThreads,
113                                     nameBuff, 0, true);
114
115     std::cout << "Turney ended with: " << score[currentAgent.
116     get()] << " : " << score[bestAgent.get()] << std::endl;
117     if (score[currentAgent.get()] > score[bestAgent.get()] *
118     scoringThreshold) {
119         version++;

```

```

110
111     currentAgent->model->save_as_current();
112     bestAgent->model->copyModel(currentAgent->model.get());
113     bestAgent->model->save_version(version);
114
115     std::sprintf(nameBuff, "models/run_%d/versionCount.jce",
runVersion);
116     std::ofstream fout(nameBuff, std::ios::binary);
117     jce::save(fout, version);
118     fout.close();
119
120     memory->save();
121
122     requiredIterations.push_back(iteration);
123     std::sprintf(nameBuff, "models/run_%d/iterationCounter.
jce", runVersion);
124     fout.open(nameBuff, std::ios::binary);
125     jce::save(fout, requiredIterations);
126     fout.close();
127
128     iteration = 0;
129 }
130 }
131 }
132 }
133
134 void AlphaZero::ai::playGames(gameOutput* output, Agent* agent1,
Agent* agent2, Memory* memory, int probMoves, int Epochs,
char RunId[], int _goesFist, bool do_log)
135 {
136     std::srand(std::chrono::time_point_cast<std::chrono::
nanoseconds>(std::chrono::system_clock::now()).
time_since_epoch().count());
137     auto game = std::make_unique<Game::Game>();
138     int goesFist = (_goesFist == 0) ? 1 : _goesFist;
139     #if ProfileLogger
140         debug::Profiler::profiler.switchOperation(3);
141     #endif
142
143     #if SaverType == 1
144         io::FullState::GameSaver saver = io::FullState::GameSaver();
145     #elif SaverType == 2
146         io::ActionsOnly::GameSaver saver = io::ActionsOnly::GameSaver
();
147     #endif

```

```

148
149   for (int epoch = 0; epoch < Epochs; epoch++) {
150   #if RenderGenAndTurneyProgress
151       jce::consoleUtils::render_progress_bar((float)epoch / (float
152       )Epochs);
153   #endif
154   #if ProfileLogger
155       debug::Profiler::profiler.switchOperation(3);
156   #endif
157   #if SaverType == 2 || SaverType == 1
158       saver.addGame();
159   #endif
160   #if MainLogger
161       if (epoch == 0 && do_log) {
162           debug::log::mainLogger->info("
163           ");
164           debug::log::mainLogger->info("=====
165           playing Next match =====");
166           debug::log::mainLogger->info("
167           ");
168       }
169   #endif
170   if (_goesFist == 0)
171   {
172       goesFist = -goesFist;
173   }
174
175   std::unordered_map<int, Agent*> players = {
176       {goesFist, agent1},
177       {-goesFist, agent2}
178   };
179   agent1->getTree()->reset();
180   agent2->getTree()->reset();
181
182   auto tmpMemory = memory->getTempMemory();
183
184   #if MainLogger
185       if (epoch == 0 && do_log)
186       {
187           debug::log::mainLogger->info("player {} will start",
188           goesFist);
189       }
190   #endif

```

```

186     game->reset();
187     int turn = 0;
188     while (!game->state->done) {
189         turn++;
190         //std::cout << turn << std::endl;
191         auto actionData = players[game->state->player]->getAction(
game->state, probMoves > turn);
192         tmpMemory.commit(game->state, actionData.second.first);
193 #if SaverType == 1
194     saver.addState(game->state);
195 #elif SaverType == 2
196     saver.addState(actionData.first);
197 #endif
198 #if MainLogger
199     if (epoch == 0 && do_log) {
200         game->state->render(debug::log::mainLogger);
201         debug::log::mainLogger->info("MSCT vals: {:1.5f}",
actionData.second.second);
202         debug::log::logVector(debug::log::mainLogger, actionData
.second.first);
203         debug::log::mainLogger->info("NN vals: {:1.5f}", players
[game->state->player]->predict(game->state).first);
204         debug::log::logVector(debug::log::mainLogger, players[
game->state->player]->predict(game->state).second);
205
206         debug::log::mainLogger->info("selected action is: {}",
actionData.first);
207     }
208 #endif
209     game->takeAction(actionData.first);
210 }
211 #if SaverType == 1
212     saver.addState(game->state);
213 #endif
214 #if MainLogger
215     if (epoch == 0 && do_log) {
216         game->state->render(debug::log::mainLogger);
217     }
218 #endif
219 #if ProfileLogger
220     debug::Profiler::profiler.switchOperation(4);
221 #endif
222     memory->updateMemory(game->state->player, std::get<0>(game->
state->val), &tmpMemory);
223     if (true)

```



```

224     {
225         if (std::get<0>(game->state->val) != 0)
226         {
227             output->updateValue(players[game->state->player * std::
get<0>(game->state->val)]);
228         }
229     }
230 #if ProfileLogger
231     debug::Profiler::profiler.stop();
232 #endif
233 }
234
235 #if SaverType == 2 || SaverType == 1
236     saver.save(RunId);
237 #endif
238
239 #if RenderGenAndTurneyProgress
240     jce::consoleUtils::render_progress_bar(1.0f, true);
241 #endif
242 }

```

### 5.1.12.7.13 utils.hpp

```
1 #pragma once
2 #include <ostream>
3 #include <istream>
4 #include <vector>
5
6 namespace AlphaZero {
7     namespace ai {
8         // Softmax function inp is an iterable of numbers
9         template<typename T>
10            //  $e^i/e^i$  (softmax function)
11            void softmax(T& i);
12
13            //  $i/i$ 
14            template<typename T>
15
16            void linmax(T& i);
17
18            template<typename T>
19            // v
20            T getSumm(std::vector<T>& v);
21    }
22 }
23
24 template<typename T>
25 inline void AlphaZero::ai::softmax(T& inp){
26     typedef float number;
27
28     number m = -10e100;
29     for (number const& z : inp){
30         if (m < z) {
31             m = z;
32         }
33     }
34
35     number sum = 0.0;
36     for (number const& z : inp) {
37         sum += exp(z - m);
38     }
39
40     number constant = m + log(sum);
41     for (number& z : inp) {
42         z = exp(z - constant);
43     }
```

```

44     throw "Depricated function";
45     return;
46 }
47
48 template<typename T>
49 void AlphaZero::ai::linmax(T& inp)
50 {
51     float sum = 0;
52     for (auto const& idx : inp)
53     {
54         sum = sum + idx;
55     }
56     for (auto& idx : inp)
57     {
58         idx = idx / sum;
59     }
60     return;
61 }
62
63 template<typename T>
64 T AlphaZero::ai::getSumm(std::vector<T>& val)
65 {
66     T out = 0;
67     for (const T& value : val)
68     {
69         out = out + value;
70     }
71     return out;
72 }

```

### 5.1.12.8 game

#### 5.1.12.8.1 game.hpp

```
1 #pragma once
2 /*
3 The game of connect for coded with game states. This allows The
4 MCTS to simulate the game without changing it.
5 */
6 #include <iostream>
7 #include <vector>
8 #include <list>
9 #include <memory>
10 #include <tuple>
11 #include <unordered_map>
12 #include <bitset>
13 #include <unordered_set>
14 #include <torch/torch.h>
15
16 #include "config.hpp"
17
18 // input_shape is the shape of the game board. x, y is the shape
19 // of the actual game board and z is number of stacked planes (
20 // in this case one for each
21 // player).
22 #define input_shape_x 7
23 #define input_shape_y 6
24 #define input_shape_z 2
25 #define action_count 42
26 #define action_shape 6, 7
27 #define boardOffset 42 // the size of a layer of the board in the
28 // buffer. (the amount of fields)(x * y)
29 // the actual name of the game
30 #define gameName "connect4"
31
32 namespace AlphaZero {
33     namespace Game {
34         // Game State class contains all information of a certain
35         // board position. a board with the positions of all pieces
36         // along with the current player. It also computes legal
37         // Actions A and win information
38         class GameState {
```

```

36 // the current player 1 or -1
37 public: int player;
38
39 // true if game is done (4 in a row or all filled) and false
    if not.
40 public: bool done;
41
42 // winning information who won and by how much. the tuple
    contains the following information in this order <current
    player win (1) or
43 // current player loose (-0) or tie (0), current player
    points (1 for win -1 for loose and 0 for tie), other player
    points (- current // player points
44 public: std::tuple<int, int, int> val;
45
46 // 84 bit bitmap that contains the current board shape. The
    first 42 bits are the positions of the player ones stones (0
    for empty, 1 for
47 // stone present) and the second 42 bits are the same for
    player -1. The 6x7 board is encoded by placing the 6 rows
    next to each other
48 // starting from the front so the top left would be bit 0 and
    the bottom right bit 41.
49 public: IDType gameBoard;
50
51 // list of allowed actions. every Action is the index of the
    bit where the stone would be placed.
52 public: std::vector<int> allowedActions;
53
54
55 // construct from known game state and player.
56 public: GameState(IDType board, int _player);
57
58 // construct using default state.
59 public: GameState();
60
61 // utilit function called by the constructors to avoid
    duplicate code. (all the initialization done by both of the
    constructors)
62 private: void initialize(IDType board, int _player);
63
64 // simulate an action from the current game state. (compute
    the state you would reach from this state by taking the
    following action)
65 public: std::shared_ptr<GameState> takeAction(int action);

```

```

66
67 // check weather the game is done. (will set the done
boolean and the val tuple)
68 public: void gameIsDone();
69
70 // compute all allowed actions and writes it to the allowed
actions list.
71 protected: void getAllowedActions();
72
73 // function that will return 1 if player 1 has a stone at
the position specified by id, -1 if player 2 does and 0 if
nether.
74 public: int IdIndex(int id);
75
76 //returns the id of the game state. In this case the game
board will surface as it contains all information. (unable to
remove stones)
77 public: IDType id();
78
79 // renders the game state to the console in a way that is
readable for humans.
80 public: void render();
81 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
|| ModelLogger)
82 // same as render() but rendes to a specified logger.
83 public: void render(std::shared_ptr<spdlog::logger> logger);
84 #endif
85 // sets the piece at id to 0, 1 or -1 (val). b is the board
that the piece will be set to. (see gameBoard for encoding)
86 public: void static IdIndex(int id, int val, IDType& b);
87
88 // converts the game state to a tensor, that is than passed
thought the model.
89 public: torch::Tensor toTensor();
90
91 // dose tha same thing but is more efficient for stacked
game states during training. It will set tensor[idx] to the
tensor representing
92 // this game state.
93 public: void toTensor(torch::Tensor& tensor, unsigned short
idx=0);
94
95 // the character representing the pieca at the position val.
(a + if a stone can be placed there)
96 private: char getPiece(int val);

```

```

97
98 // recursive function used to determin the height of a colum
    // to determin where stones can be placed. The returned int is
    // a possible
99 // placement position if the bool is true. if the bool is
    // false the colum is full.
100 private: std::pair<int, bool> getAllowedColumHeight(int);
101 };
102
103 // hash function used for hash mapes using the board as a
    // hash key.
104 struct StateHash
105 {
106     std::size_t operator()(std::pair<std::shared_ptr<GameState>
    >, std::vector<int>>> const& s) const noexcept;
107 };
108 // returns all identidal game states and action maps (N) to
    // the passed one.
109 std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
    GameState>, std::vector<int>>> identities(std::shared_ptr<
    GameState> state, std::vector<int>& actionProbs);
110
111
112 // The Actual game contains and handles the gamestates for
    // the current generation Game.
113 class Game {
114
115     // pointer to the current game state.
116     public: std::shared_ptr<GameState> state;
117
118
119     // constructor will initialize a game state.
120     public: Game();
121
122     // reset game to initial position
123     public: void reset();
124
125     // take an action as defined by the game state
126     public: void takeAction(int action);
127
128     // human action (action is the colum that the stone should
    // be placed in, than the action is determined)
129     public: bool takeHumanAction(int action);
130
131     // call the states render function.

```

```

132     public: void render();
133 };
134
135 // Test the game.
136 inline void test() {
137     AlphaZero::Game::Game* game = new AlphaZero::Game::Game();
138
139     while (!game->state->done) {
140         std::vector<int> vec = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
141                                10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
142                                25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
143                                40, 41 };
144         auto idents = identities(game->state, vec);
145         idents[1].first->render();
146         std::cout << "your action: ";
147         int action;
148         std::cin >> action;
149         game->takeHumanAction(action);
150 #if Windows
151         system("cls");
152 #else
153         system("clear");
154 #endif
155     }
156     game->render();
157
158     std::cout << std::endl << "the last player just won";
159 }
160 }
161
162 inline std::size_t AlphaZero::Game::StateHash::operator()(std::
163     pair<std::shared_ptr<GameState>, std::vector<int>>> const& s)
164     const noexcept {
165     return s.first->gameBoard.to_ulong();
166 }
167
168 inline int AlphaZero::Game::GameState::IdIndex(int id)
169 {
170     if (this->gameBoard[id] == 1) {
171         return 1;
172     }
173     else if (this->gameBoard[id + boardOffset] == 1) {
174         return -1;
175     }
176 }

```



```

172     return 0;
173 }
174
175 inline void AlphaZero::Game::GameState::IdIndex(int id, int val,
176 IDType& b)
177 {
178     if (val == 0) {
179         b.set(id, 0);
180         b.set(id + boardOffset, 0);
181         return;
182     }
183     if (val == -1) {
184         id += boardOffset;
185     }
186     b.set(id, 1);
187 }
188
189 inline char AlphaZero::Game::GameState::getPiece(int id)
190 {
191     std::unordered_map<int, char> renderData = { {0, '-'}, {1, 'X'}
192     }, {-1, 'O'} };
193     if (std::find(this->allowedActions.begin(), this->
194     allowedActions.end(), id) != this->allowedActions.end()) {
195         return '+';
196     }
197     auto va = renderData[this->IdIndex(id)];
198     return va;
199 }
200
201 inline IDType AlphaZero::Game::GameState::id()
202 {
203     return this->gameBoard;
204 }
205
206 inline void AlphaZero::Game::Game::render() {
207     this->state->render();
208 }
209
210 inline torch::Tensor AlphaZero::Game::GameState::toTensor()
211 {
212     at::Tensor outTensor = at::zeros({ 1, input_shape_z,
213     input_shape_y, input_shape_x });
214     this->toTensor(outTensor);
215     return outTensor;
216 }

```

```

213
214 inline void AlphaZero::Game::GameState::toTensor(torch::Tensor&
      tensor, unsigned short idx)
215 {
216     unsigned short pos = 0;
217     unsigned int offset = (this->player == -1) ? 0 : boardOffset;
218     for (unsigned short z = 0; z < input_shape_z; z++) {
219         for (unsigned short y = 0; y < input_shape_y; y++) {
220             for (unsigned short x = 0; x < input_shape_x; x++) {
221                 tensor[idx][z][y][x] = (float) this->gameBoard[(pos +
      offset) % stateSize];
222                 pos++;
223             }
224         }
225     }
226 }

```

### 5.1.12.8.2 game.cpp

```
1 #include "game.hpp"
2
3 #define columnOffset 7
4
5 AlphaZero::Game::GameState::GameState(IDType board, int _player)
6 {
7     this->initialize(board, _player);
8 }
9
10 AlphaZero::Game::GameState::GameState()
11 {
12     this->initialize(IDType(), 1);
13 }
14
15 void AlphaZero::Game::GameState::initialize(IDType board, int
    _player)
16 {
17     this->gameBoard = board;
18     this->player = _player;
19     this->getAllowedActions();
20     this->gameIsDone();
21 }
22
23 std::shared_ptr<AlphaZero::Game::GameState> AlphaZero::Game::
    GameState::takeAction(int action)
24 {
25     IDType newBoard = this->gameBoard;
26     GameState::IdIndex(action, this->player, newBoard);
27
28     std::shared_ptr<GameState> newState = std::make_shared<
        GameState>(newBoard, -this->player);
29     return newState;
30 }
31
32 void AlphaZero::Game::GameState::gameIsDone()
33 {
34     std::vector<std::vector<int>> winOptions = {
35         /*
36         +-----+
37         | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
38         +-----+
39         | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
40         +-----+
41         */
36
37
38
39
40
```

```

41 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
42 +---+---+---+---+---+---+---+
43 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
44 +---+---+---+---+---+---+---+
45 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
46 +---+---+---+---+---+---+---+
47 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |
48 +---+---+---+---+---+---+---+
49 */
50 //horizontal
51 {0,1,2,3},
52 {1,2,3,4},
53 {2,3,4,5},
54 {3,4,5,6},
55
56 {7,8,9,10},
57 {8,9,10,11},
58 {9,10,11,12},
59 {10,11,12,13},
60
61 {14,15,16,17},
62 {15,16,17,18},
63 {16,17,18,19},
64 {17,18,19,20},
65
66 {21,22,23,24},
67 {22,23,24,25},
68 {23,24,25,26},
69 {24,25,26,27},
70
71 {28,29,30,31},
72 {29,30,31,32},
73 {30,31,32,33},
74 {31,32,33,34},
75
76 {35,36,37,38},
77 {36,37,38,39},
78 {37,38,39,40},
79 {38,39,40,41},
80 //vertical
81 {0 , 7,14,21},
82 {7 ,14,21,28},
83 {14,21,28,35},
84
85 {1 ,8 ,15,22},

```

```

86     {8 ,15,22,29} ,
87     {15,22,29,36} ,
88
89     {2,9,16,23} ,
90     {9,16,23,30} ,
91     {16,23,30,37} ,
92
93     {3 ,10,17,24} ,
94     {10,17,24,31} ,
95     {17,24,31,38} ,
96
97     {4 ,11,18,25} ,
98     {11,18,25,32} ,
99     {18,25,32,39} ,
100
101     {5 ,12,19,26} ,
102     {12,19,26,33} ,
103     {19,26,33,40} ,
104
105     {6 ,13,20,27} ,
106     {13,20,27,34} ,
107     {20,27,34,41} ,
108
109     //diagonal topleft-bottomRight
110     {14,22,30,38} ,
111
112     {7, 15,23,31} ,
113     {15,23,31,39} ,
114
115     {0, 8, 16,24} ,
116     {8, 16,24,32} ,
117     {16,24,32,40} ,
118
119     {1, 9 ,17,25} ,
120     {9, 17,25,33} ,
121     {17,25,33,41} ,
122
123     {2, 10,18,26} ,
124     {10,18,26,34} ,
125
126     {3, 11,19,27} ,
127
128     //diagonal topright-bottomleft
129     {3, 9, 15,21} ,
130

```

```

131     {4, 10,16,22},
132     {10,16,22,28},
133
134     {5, 11,17,23},
135     {11,17,23,29},
136     {17,23,29,35},
137
138     {6, 12,18,24},
139     {12,18,24,30},
140     {18,24,30,36},
141
142     {13,19,25,31},
143     {19,25,31,37},
144
145     {20,26,32,38},
146 };
147 bool tie = true;
148 for (int idx = 0; idx < action_count; idx++) {
149     if (this->IdIndex(idx) == 0) {
150         tie = false;
151         break;
152     }
153 }
154 if (tie) {
155     this->done = true;
156     this->val = { 0,0,0 };
157     return;
158 }
159 for (auto option : winOptions) {
160     int count = 0;
161     for (int pos : option) {
162         count += this->IdIndex(pos);
163     }
164     if (count == -4 * this->player) {
165         this->done = true;
166         this->val = { -1, -1, 1 }; // winForThisPlayer, points for
167                                     this player, points for other player
168         return;
169     }
170 }
171 this->done = false;
172 this->val = { 0, 0, 0 };
173 }
174 inline std::pair<int, bool> AlphaZero::Game::GameState::

```

```

175     getAllowedColumnHeight(int idx) {
176     if (this->IdIndex(idx) != 0) {
177         return { idx, false };
178     }
179     if (idx >= 35) {
180         return {idx, true};
181     }
182     else if (this->IdIndex(idx + columnOffset)!=0) {
183         return { idx, true };
184     }
185     else {
186         return this->getAllowedColumnHeight(idx + columnOffset);
187     }
188 }
189 void AlphaZero::Game::GameState::getAllowedActions()
190 {
191     this->allowedActions.clear();
192     for (int idx = 0; idx < 7; idx++) {
193         std::pair<int, bool> data = this->getAllowedColumnHeight(idx)
194         ;
195         if (data.second) {
196             this->allowedActions.push_back(data.first);
197         }
198     }
199 }
200 void AlphaZero::Game::GameState::render()
201 {
202     console_mutex.lock();
203     for (int row = 0; row < action_count;) {
204         for (int iter = 0; iter < 7; iter++) {
205             std::cout << this->getPiece(row) << " ";
206             row++;
207         }
208         std::cout << std::endl;
209     }
210     std::cout << std::endl;
211     console_mutex.unlock();
212 }
213
214 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
215     || ModelLogger)
216 void AlphaZero::Game::GameState::render(std::shared_ptr<spdlog::
217     logger> logger)

```

```

216 {
217     for (int idx = 0; idx < 6; idx++) {
218         char line1[13] = {
219             this->getPiece(0 + columnOffset * idx), ' ',
220             this->getPiece(1 + columnOffset * idx), ' ',
221             this->getPiece(2 + columnOffset * idx), ' ',
222             this->getPiece(3 + columnOffset * idx), ' ',
223             this->getPiece(4 + columnOffset * idx), ' ',
224             this->getPiece(5 + columnOffset * idx), ' ',
225             this->getPiece(6 + columnOffset * idx)
226         };
227         logger->info(line1);
228     }
229 }
230 #endif
231
232 AlphaZero::Game::Game::Game()
233 {
234     this->state = std::make_shared<GameState>();
235 }
236
237 void AlphaZero::Game::Game::reset()
238 {
239     this->state = std::make_shared<GameState>();
240 }
241
242 void AlphaZero::Game::Game::takeAction(int action)
243 {
244     auto newState = this->state->takeAction(action);
245     this->state = newState;
246 }
247
248 bool AlphaZero::Game::Game::takeHumanAction(int action)
249 {
250     for (auto const& allowed : this->state->allowedActions) {
251         if ((allowed - action) % 7 == 0) {
252             this->takeAction(allowed);
253             return true;
254         }
255     }
256     return false;
257 }
258
259 inline std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
    std::vector<int>> mirrorGameState(std::shared_ptr<AlphaZero::

```



```

260     Game::GameState> state, std::vector<int>& actionProbs) {
261     IDType boardBuffer;
262     std::vector<int> probs = {
263         actionProbs[6], actionProbs[5], actionProbs[4],
264         actionProbs[3], actionProbs[2], actionProbs[1],
265         actionProbs[0],
266         actionProbs[13], actionProbs[12], actionProbs[11],
267         actionProbs[10], actionProbs[9], actionProbs[8],
268         actionProbs[7],
269         actionProbs[20], actionProbs[19], actionProbs[18],
270         actionProbs[17], actionProbs[16], actionProbs[15],
271         actionProbs[14],
272         actionProbs[27], actionProbs[26], actionProbs[25],
273         actionProbs[24], actionProbs[23], actionProbs[22],
274         actionProbs[21],
275         actionProbs[34], actionProbs[33], actionProbs[32],
276         actionProbs[31], actionProbs[30], actionProbs[29],
277         actionProbs[28],
278         actionProbs[41], actionProbs[40], actionProbs[39],
279         actionProbs[38], actionProbs[37], actionProbs[36],
280         actionProbs[35]
281     };
282     #define assignStateSinge(idx1, idx2) AlphaZero::Game::GameState::
283         IdIndex(idx1, state->IdIndex(idx2), boardBuffer)
284     #define assignState(idx1, idx2) assignStateSinge(idx1, idx2);
285         assignStateSinge(idx2, idx1);
286
287     assignState(0, 6); assignState(1, 5); assignState(2, 4);
288         assignStateSinge(3, 3);
289     assignState(7, 13); assignState(8, 12); assignState(9, 11);
290         assignStateSinge(10, 10);
291     assignState(14, 20); assignState(15, 19); assignState(16, 18);
292         assignStateSinge(17, 17);
293     assignState(21, 27); assignState(22, 26); assignState(23, 25);
294         assignStateSinge(24, 24);
295     assignState(28, 34); assignState(29, 33); assignState(30, 32);
296         assignStateSinge(31, 31);
297     assignState(35, 41); assignState(36, 40); assignState(37, 39);
298         assignStateSinge(38, 38);
299     #undef assignState
300
301     return { std::make_shared<AlphaZero::Game::GameState>(
302         boardBuffer, state->player), probs };

```

```

283 }
284
285 std::vector<std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
    std::vector<int>>>> AlphaZero::Game::identities(std::shared_ptr<GameState> state,
    std::vector<int>& probs)
286 {
287     std::vector<std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
    std::vector<int>>>> idents(2);
288     int id = 0;
289     idents[0] = { state, probs };
290     idents[1] = mirrorGameState(state, probs);
291     return idents;
292 }

```

### 5.1.12.9 jce

#### 5.1.12.9.1 load.hpp

```
1 #pragma once
2
3 #include <fstream>
4 #include <list>
5 #include <vector>
6 #include <bitset>
7 #include <string>
8 #include <ai/memory.hpp>
9 #include <torch/torch.h>
10 #include <game/game.hpp>
11
12 #define BasicLoad(in, data) (in.read((char*)&data, sizeof(data))
13 )
14
15 namespace jce
16 {
17     // custom part
18
19     // load GameState from file
20     void load(std::ifstream& in, std::shared_ptr<AlphaZero::Game::
21         GameState>& state);
22
23     // load Memory Element from file
24     void load(std::ifstream& in, std::shared_ptr<AlphaZero::ai::
25         MemoryElement>& element);
26
27     template<typename key, typename T>
28     void load(std::ifstream& in, torch::OrderedDict<key, T>& map);
29
30     // load Tensor from file
31     template<typename T=float>
32     void load(std::ifstream& in, torch::Tensor& tensor);
33
34     // end custom part
35
36     template<typename key, typename T>
37     void load(std::ifstream& in, std::unordered_map<key, T>& map);
38
39     // load pair from file
40     template<typename T, typename T2>
41     void load(std::ifstream& in, std::pair<T, T2>& data);
```

```

39
40 // load list from file
41 template<typename T>
42 void load(std::ifstream& in, std::list<T>& data);
43
44 // load vector from file
45 template<typename T>
46 void load(std::ifstream& in, std::vector<T>& data);
47
48 //load bitset from file
49 template<size_t size>
50 void load(std::ifstream& in, std::bitset<size>& data);
51
52 // the actual loading of ints and vectors called by the load
   function
53 template <typename T>
54 void load_listVec(std::ifstream& in, T& data);
55
56 // load int from file
57 void load(std::ifstream& in, int& data);
58 // load unsigned int from file
59 void load(std::ifstream& in, unsigned int& data);
60 // load size_t from file
61 void load(std::ifstream& in, size_t& data);
62 // load float from file
63 void load(std::ifstream& in, float& data);
64 // load double from file
65 void load(std::ifstream& in, double& data);
66 // load char from file
67 void load(std::ifstream& in, char*& data);
68 // load int64_t from file
69 void load(std::ifstream& in, int64_t& data);
70 // load string from file
71 void load(std::ifstream& in, std::string& data);
72 }
73
74 inline void jce::load(std::ifstream& in, std::shared_ptr<
   AlphaZero::Game::GameState>& state)
75 {
76     IDType board;
77     int player;
78     jce::load(in, board);
79     jce::load(in, player);
80     state.reset(new AlphaZero::Game::GameState(board, player));
81 }

```

```

82
83 inline void jce::load(std::ifstream& in, std::shared_ptr<
    AlphaZero::ai::MemoryElement>& element)
84 {
85     element.reset(new AlphaZero::ai::MemoryElement());
86     jce::load(in, element->value);
87     jce::load(in, element->state);
88     jce::load(in, element->av);
89 }
90
91 template<typename key, typename T>
92 inline void jce::load(std::ifstream& in, torch::OrderedDict<key,
    T>& map)
93 {
94     size_t size;
95     jce::load(in, size);
96     for (size_t idx = 0; idx < size; idx++)
97     {
98         key _key;
99         T _item;
100         jce::load(in, _key);
101         jce::load(in, _item);
102         map.insert(_key, _item);
103     }
104 }
105
106 template<typename T>
107 inline void jce::load(std::ifstream& in, torch::Tensor& tensor)
108 {
109     std::vector<int64_t> vec;
110     int64_t fullSize = 1;
111     jce::load(in, vec);
112     for (size_t value : vec) { fullSize *= value; }
113     tensor = torch::zeros({ fullSize });
114     T val, last;
115     for (size_t idx = 0; idx < fullSize; idx++)
116     {
117         jce::load(in, val);
118         tensor[idx] = val;
119         last = val;
120     }
121     tensor = torch::reshape(tensor, vec);
122 }
123
124 template<typename key, typename T>

```

```

125 inline void jce::load(std::ifstream& in, std::unordered_map<key,
    T>& map)
126 {
127     size_t size;
128     jce::load(in, size);
129     for (size_t idx = 0; idx < size; idx++)
130     {
131         key _key;
132         T _item;
133         jce::load(in, _key);
134         jce::load(in, _item);
135         map.insert({ _key, _item });
136     }
137 }
138
139 template<typename T, typename T2>
140 inline void jce::load(std::ifstream& in, std::pair<T, T2>& data)
141 {
142     jce::load(in, data.first);
143     jce::load(in, data.second);
144 }
145
146 template<typename T>
147 inline void jce::load(std::ifstream& in, std::list<T>& data)
148 {
149     load_listVec(in, data);
150 }
151
152 template<typename T>
153 inline void jce::load(std::ifstream& in, std::vector<T>& data)
154 {
155     load_listVec(in, data);
156 }
157
158 template<size_t size>
159 inline void jce::load(std::ifstream& in, std::bitset<size>& data
    )
160 {
161     char byte;
162     for (size_t idx = 0; idx < size; idx = idx + 8)
163     {
164         in.read(&byte, 1);
165         std::bitset<8> tempSet(byte);
166         for (size_t pos = 0; pos < 8 && pos + idx < size; pos++)
167         {

```

```

168     data.set(pos + idx, tempSet[pos]);
169 }
170 }
171 }
172
173 template<typename T>
174 inline void jce::load_listVec(std::ifstream& in, T& data)
175 {
176     size_t size;
177     jce::load(in, size);
178     data.resize(size);
179     for (auto& val : data)
180     {
181         jce::load(in, val);
182     }
183 }
184
185 inline void jce::load(std::ifstream& in, int& data){ BasicLoad(
    in, data); }
186 inline void jce::load(std::ifstream& in, unsigned int& data) {
    BasicLoad(in, data); }
187 inline void jce::load(std::ifstream& in, size_t& data) {
    BasicLoad(in, data); }
188 inline void jce::load(std::ifstream& in, float& data) {
    BasicLoad(in, data); }
189 inline void jce::load(std::ifstream& in, double& data) {
    BasicLoad(in, data); }
190 inline void jce::load(std::ifstream& in, int64_t& data) {
    BasicLoad(in, data); }
191
192 inline void jce::load(std::ifstream& in, std::string& data)
193 {
194     char* c_arr;
195     jce::load(in, c_arr);
196     data = std::string(c_arr);
197 }
198
199 inline void jce::load(std::ifstream& in, char*& data)
200 {
201     std::vector<char> data_vec;
202     while (true)
203     {
204         char next;
205         in.read(&next, 1);
206         data_vec.push_back(next);

```

```
207     if (next == NULL)
208     {
209         break;
210     }
211 }
212 data = new char[data_vec.size()];
213 auto pos = data;
214 for (auto const& value : data_vec)
215 {
216     (*pos) = value;
217     pos++;
218 }
219 }
```



### 5.1.12.9.2 save.hpp

```
1 #pragma once
2
3 #include <fstream>
4 #include <iostream>
5 #include <list>
6 #include <vector>
7 #include <string>
8 #include <bitset>
9 #include <ai/memory.hpp>
10 #include <torch/torch.h>
11
12 #define BasicSave(data, out) (out.write((char*)&data, sizeof(
    data)))
13 #define BasicSave_cp(data, out) (out.write((char*)data, sizeof(
    data)))
14
15 namespace jce
16 {
17     // custom part
18
19     // save GameState to file
20     void save(std::ofstream& out, std::shared_ptr<AlphaZero::Game
        ::GameState> const& state);
21
22     // save memory element to file
23     void save(std::ofstream& out, std::shared_ptr<AlphaZero::ai::
        MemoryElement> const& element);
24
25     // save Tensor to file
26     template<typename T=float>
27     void save(std::ofstream& out, torch::Tensor const& tensor);
28
29     template<typename key, typename T>
30     void save(std::ofstream& out, torch::OrderedDict<key, T> const
        & map);
31
32     // end custom part
33
34     template<typename T, typename key>
35     void save(std::ofstream& out, std::unordered_map<T, key>);
36
37     template<typename T, typename T2>
38     // save pair to file
```

```

39 void save(std::ofstream& out, std::pair<T, T2>const& data);
40
41 template<typename T>
42 //save list to ofstream
43 void save(std::ofstream& out, std::list<T> const& data);
44
45 template<typename T>
46 //save vector to file
47 void save(std::ofstream& out, std::vector<T> const& data);
48 template<typename T>
49 void quick_save(std::ofstream& out, std::vector<T> const& data
50 );
51 //save bitset to file
52 template<size_t T>
53 void save(std::ofstream& out, std::bitset<T> const& data);
54
55 template<typename T>
56 //the actual saving function for vectors and lists
57 void save_listVec(std::ofstream& out, T const& data);
58
59 // save int to file
60 void save(std::ofstream& out, int const& data);
61 // save unsigned int to file
62 void save(std::ofstream& out, unsigned int const& data);
63 // save size_t to file
64 void save(std::ofstream& out, size_t const& data);
65 // save float to file
66 void save(std::ofstream& out, float const& data);
67 // save double to file
68 void save(std::ofstream& out, double const& data);
69 // save char to file
70 void save(std::ofstream& out, const char* arr);
71 // save int64_t to file
72 void save(std::ofstream& out, const int64_t& data);
73 // save string to file
74 void save(std::ofstream& out, const std::string& data);
75 }
76
77 inline void jce::save(std::ofstream& out, std::shared_ptr<
78     AlphaZero::ai::MemoryElement> const& element)
79 {
80     jce::save(out, element->value);
81     jce::save(out, element->state);
82     jce::save(out, element->av);

```

```

82 }
83
84 template<typename T>
85 inline void jce::save(std::ofstream& out, torch::Tensor const&
      tensor)
86 {
87     jce::save(out, tensor.sizes().vec());
88     auto flatTensor = torch::flatten(tensor);
89     for (size_t idx = 0; idx < flatTensor.size(0); idx++)
90     {
91         T val = flatTensor[idx].item<T>();
92         jce::save(out, val);
93     }
94 }
95
96 template<typename key, typename T>
97 inline void jce::save(std::ofstream& out, torch::OrderedDict<key
      , T> const& map)
98 {
99     jce::save(out, map.size());
100     for (auto& val : map)
101     {
102         jce::save(out, val.key());
103         jce::save(out, val.value());
104     }
105 }
106
107 inline void jce::save(std::ofstream& out, std::shared_ptr<
      AlphaZero::Game::GameState> const& state)
108 {
109     jce::save(out, state->gameBoard);
110     jce::save(out, state->player);
111 }
112
113 template<typename T, typename key>
114 inline void jce::save(std::ofstream& out, std::unordered_map<T,
      key> map)
115 {
116     jce::save(out, map.size());
117     for (auto const& val : map)
118     {
119         jce::save(out, val.first);
120         jce::save(out, val.second);
121     }
122 }

```

```

123
124 template<typename T, typename T2>
125 inline void jce::save(std::ofstream& out, std::pair<T, T2> const
    & data)
126 {
127     jce::save(out, data.first);
128     jce::save(out, data.second);
129 }
130
131 template<typename T>
132 inline void jce::save(std::ofstream& out, std::list<T> const&
    data)
133 {
134     save_listVec(out, data);
135 }
136
137 template <typename T>
138 inline void jce::quick_save(std::ofstream& out, std::vector<T>
    const& data)
139 {
140     jce::save(out, data.size());
141     BasicSave_cp(data.data(), out);
142 }
143
144 template <typename T>
145 inline void jce::save(std::ofstream& out, std::vector<T> const&
    data)
146 {
147     save_listVec(out, data);
148 }
149
150 template<size_t T>
151 inline void jce::save(std::ofstream& out, std::bitset<T> const&
    data)
152 {
153     std::bitset<8> temp;
154     size_t tempVal;
155     for (size_t idx = 0; idx < T; idx = idx + 8)
156     {
157         for (size_t pos = 0; pos < 8 && pos + idx < T; pos++) {
158             temp.set(pos, data[pos + idx]);
159         }
160         tempVal = temp.to_ullong();
161         out.write((char*)&tempVal, 1);
162     }

```

```

163 }
164
165 template <typename T>
166 inline void jce::save_listVec(std::ofstream& out, T const & data
167 )
168 {
169     jce::save(out, data.size());
170     for (auto const& data : data)
171     {
172         jce::save(out, data);
173     }
174 }
175
176 inline void jce::save(std::ofstream& out, int const& data) {
177     BasicSave(data, out); }
178 inline void jce::save(std::ofstream& out, unsigned int const&
179 data) { BasicSave(data, out); }
180 inline void jce::save(std::ofstream& out, size_t const& data) {
181     BasicSave(data, out); }
182 inline void jce::save(std::ofstream& out, float const& data) {
183     BasicSave(data, out); }
184 inline void jce::save(std::ofstream& out, double const& data) {
185     BasicSave(data, out); }
186 inline void jce::save(std::ofstream& out, const int64_t& data) {
187     BasicSave(data, out); }
188
189 inline void jce::save(std::ofstream& out, const std::string&
190 data)
191 {
192     jce::save(out, data.c_str());
193 }
194
195 inline void jce::save(std::ofstream& out, const char* arr)
196 {
197     size_t size = 1;
198     auto iterator = arr;
199     while (*iterator != NULL)
200     {
201         size++;
202         iterator++;
203     }
204     out.write(arr, size);
205 }

```

### 5.1.12.9.3 string.hpp

```
1 #pragma once
2 #include <iostream>
3
4 namespace jce
5 {
6     namespace consoleUtils
7     {
8         // render a progress bar to console (from stack
9         // overflow)
10         void render_progress_bar(float progress, bool persistant
11         = false);
12     }
13 }
14
15 inline void jce::consoleUtils::render_progress_bar(float
16 progress, bool persistant)
17 {
18     #if true
19     if (progress <= 1.0) {
20         int barWidth = 70;
21
22         std::cout << "[";
23         int pos = barWidth * progress;
24         for (int i = 0; i < barWidth; ++i) {
25             if (i < pos) std::cout << "=";
26             else if (i == pos) std::cout << ">";
27             else std::cout << " ";
28         }
29         if (persistant)
30         {
31             std::cout << "]" << int(progress * 100.0) << std:::
32             endl;
33         }
34         else
35         {
36             std::cout << "]" << int(progress * 100.0) << " %\r"
37             ;
38         }
39         std::cout.flush();
40     }
41     #endif
42 }
```

#### 5.1.12.9.4 vector.hpp

```
1 #pragma once
2 #include <vector>
3
4 namespace jce {
5     namespace vector {
6         template <typename T>
7             // equivilent to np.fill() [one dimension only]
8             std::vector<T> gen(size_t size, T val);
9     }
10 }
11
12 template<typename T>
13 inline std::vector<T> jce::vector::gen(size_t size, T val)
14 {
15     std::vector<T> out(size);
16     for (auto& item : out) {
17         item = val;
18     }
19     return out;
20 }
```

### 5.1.12.10 server

#### 5.1.12.10.1 eloClient.hpp

```
1 // client that interfaces with the elo server (elo directory)
2
3 #pragma once
4
5 #include <config.hpp>
6 #include <sockpp/tcp_connector.h>
7 #include <ai/agent.hpp>
8
9
10 #define ELO_PORT 2551
11 #define ELO_IP "wandhoven.ddns.net"
12
13 namespace AlphaZero
14 {
15     namespace elo
16     {
17         // Class that handles communication to the elo server.
18         class eloClient
19         {
20             // set the elo rating of a certain agent
21             public: int setElo(int agent1, int eloRating) const;
22
23             // get the elo rating of an agent.
24             public: int getElo(int agent1) const;
25
26             // get the agent with the closest elo rating to the
27             // specified value.
28             public: int getAgentWithClosestElo(int eloVal) const;
29         };
30     }
31
32     inline int AlphaZero::elo::eloClient::setElo(int agent1, int elo
33         ) const
34     {
35         sockpp::socket_initializer sockInit;
36         in_port_t port = ELO_PORT;
37         std::string host = ELO_IP;
38         sockpp::tcp_connector conn({ host, port });
39         if (!conn)
40         {
```



```

40     std::cout << (conn.last_error_str()) << std::endl;
41     return -1;
42 }
43
44 int data[3] = { 2, agent1, elo };
45 conn.write(data, sizeof(data));
46
47 int delo[1];
48 conn.read(delo, sizeof(delo));
49 return delo[0];
50 }
51
52 inline int AlphaZero::elo::eloClient::getElo(int agent1) const
53 {
54     sockpp::socket_initializer sockInit;
55     in_port_t port = ELO_PORT;
56     std::string host = ELO_IP;
57     sockpp::tcp_connector conn({ host, port });
58     if (!conn)
59     {
60         std::cout << (conn.last_error_str()) << std::endl;
61         return -1;
62     }
63
64     int data[2] = { 1, agent1 };
65     conn.write(data, sizeof(data));
66
67     int elo[1];
68     conn.read(elo, sizeof(elo));
69     return elo[0];
70 }
71
72 inline int AlphaZero::elo::eloClient::getAgentWithClosestElo(int
    val) const
73 {
74     sockpp::socket_initializer sockInit;
75     in_port_t port = ELO_PORT;
76     std::string host = ELO_IP;
77     sockpp::tcp_connector conn({ host, port });
78     if (!conn)
79     {
80         std::cout << (conn.last_error_str()) << std::endl;
81         return -1;
82     }
83

```

```
84  int data[2] = { -1, val };
85  conn.write(data, sizeof(data));
86
87  int elo[1];
88  conn.read(elo, sizeof(elo));
89  return elo[0];
90 }
```

### 5.1.12.10.2 server.hpp

```
1 // Ai server
2
3 #pragma once
4
5 #include <config.hpp>
6 #include <sockpp/tcp_acceptor.h>
7 #include <ai/agent.hpp>
8
9
10 #define PORT 25500
11
12 namespace AlphaZero
13 {
14     namespace Server
15     {
16         // convert an array socket communication to board
17         IDType toBoard(int arr[]);
18
19         // Ai communication server
20         class TCPServer
21         {
22             // evaluate a socket request (ask the Ai what action they
23             // want to take and send it back)
24             private: void evaluate(sockpp::tcp_socket& sock);
25
26             // socket initializer (see https://github.com/fpagliughi/
27             // sockpp)
28             private: sockpp::socket_initializer sockInit;
29
30             // server listening socket used to accept evaluation
31             // requests
32             private: sockpp::tcp_acceptor acc;
33
34             // accept an incoming connection, call the evaluate
35             // method and close the connection
36             private: void accept();
37
38             // server constructor on a certain port
39             public: TCPServer(int port = PORT);
40
41             // while true call accept()
42             public: void mainLoop();
43         };
44     }
45 }
```

```

40     };
41
42     // same as the TCPServer but uses human inputs for tesing
    purpusses.
43     class TestServer
44     {
45     // socket initializer (see https://github.com/fpagliughi/
    sockpp)
46     private: sockpp::socket_initializer sockInit;
47
48     // server listening socket used to accept evaluation
    requests
49     private: sockpp::tcp_acceptor acc;
50
51     // accet connection, ask human controller for action, send
    action to client, close connection
52     private: void accept();
53
54
55     // create a test server on a certain port
56     public: TestServer(int port = PORT);
57
58     // call the accept method while forever
59     public: void mainLoop();
60     };
61 }
62 }
63
64 inline IDType AlphaZero::Server::toBoard(int arr[])
65 {
66     IDType out;
67     for (int idx = 0; idx < stateSize; idx++)
68     {
69         out.set(idx, arr[idx]);
70     }
71     return out;
72 }

```

### 5.1.12.10.3 server.cpp

```
1 // SockServer.cpp : Defines the entry point for the application.
2 //
3
4 #pragma comment( lib , "ws2_32.lib" )
5
6 #include "server.hpp"
7 #include <game/game.hpp>
8 #include <iostream>
9 #include <log.hpp>
10
11
12 std::shared_ptr<spdlog::logger> logger = debug::log::
    createLogger( "ServerLogger", "logs/c++/Server.log" );
13
14 std::vector<char*> devices = { DEVICES };
15 std::shared_ptr<AlphaZero::ai::Agent> agent = std::make_shared<
    AlphaZero::ai::Agent>(devices);
16
17 inline void AlphaZero::Server::TCPServer::evaluate(sockpp::
    tcp_socket& sock) {
18     ssize_t n;
19     int buf[stateSize + 2];
20     int out[1];
21
22     n = sock.read(buf, sizeof(buf));
23
24     std::shared_ptr<AlphaZero::Game::GameState> state = std::
        make_shared<AlphaZero::Game::GameState>(AlphaZero::Server::
            toBoard(buf), buf[stateSize]);
25
26     agent->reset();
27     try
28     {
29         std::cout << "model version is: " << buf[stateSize + 1] <<
            std::endl;
30         agent->model->load_version(buf[stateSize + 1]);
31     }
32     catch (...)
33     {
34         agent->model->load_current();
35     }
36
37     auto actionData = agent->getAction(state, false);
```

```

38   out[0] = actionData.first;
39
40   #if MainLogger
41       state->render(logger);
42       logger->info("MSCT vals: {:.15f}", actionData.second.second);
43       debug::log::logVector(logger, actionData.second.first);
44       logger->info("NN vals: {:.15f}", agent->predict(state).first);
45       debug::log::logVector(logger, agent->predict(state).second);
46       logger->info("NN Q:");
47       debug::log::logVector(logger, AlphaZero::ai::getQ(agent->
           getTree()->getNode(state->id())));
48
49       logger->info("selected action is: {}", actionData.first);
50
51       logger->flush();
52   #endif
53
54       sock.write_n(out, sizeof(int));
55
56       logger->info("Connection closed");
57   }
58
59   void AlphaZero::Server::TCPServer::accept()
60   {
61       sockpp::tcp_socket sock = this->acc.accept();
62       logger->info("Connection accepted from ", sock.peer_address().
           to_string());
63       evaluate(sock);
64   }
65
66
67   AlphaZero::Server::TCPServer::TCPServer(int _port)
68   {
69       in_port_t port = _port;
70       this->acc = sockpp::tcp_acceptor(port);
71
72       if (!acc) {
73           std::cerr << "Error creating the acceptor: " << acc.
               last_error_str() << std::endl;
74       }
75
76       std::cout << "Acceptor bound to address: " << this->acc.
           address() << std::endl;
77       std::cout << "Awaiting connections on port " << port << "... "
           << std::endl;

```

```

78
79     logger->info("Acceptor bound to address: ", this->acc.address
80         ().to_string());
81     logger->info("Awaiting connections on port: {}", port);
82 }
83
84 void AlphaZero::Server::TCPServer::mainLoop()
85 {
86     while (true)
87     {
88         this->accept();
89     }
90 }
91
92 AlphaZero::Server::TestServer::TestServer(int _port)
93 {
94     in_port_t port = _port;
95     this->acc = sockpp::tcp_acceptor(port);
96
97     if (!acc) {
98         std::cerr << "Error creating the acceptor: " << acc.
99             last_error_str() << std::endl;
100     }
101     std::cout << "Acceptor bound to address: " << acc.address() <<
102         std::endl;
103     std::cout << "Awaiting connections on port " << port << "... "
104         << std::endl;
105 }
106
107 void AlphaZero::Server::TestServer::mainLoop()
108 {
109     while (true)
110     {
111         this->accept();
112     }
113 }
114
115 void AlphaZero::Server::TestServer::accept()
116 {
117     sockpp::tcp_socket sock = this->acc.accept();
118
119     std::cout << "Connection accepted from " << sock.peer_address()
120         << std::endl;
121
122     ssize_t n;

```

```

118     int buf[stateSize + 1];
119     int out[1];
120
121     n = sock.read(buf, sizeof(buf));
122
123     std::shared_ptr<Game::GameState> state = std::make_shared<Game
124         ::GameState>(toBoard(buf), buf[stateSize]);
125     state->render();
126
127     std::cout << "Server Action for testing: ";
128     std::cin >> out[0];
129     std::cout << std::endl;
130
131     sock.write_n(out, sizeof(int));
132
133     std::cout << "Connection closed from " << sock.peer_address()
134         << std::endl;
135 }

```



### 5.1.12.11 test

#### 5.1.12.11.1 testSuit.hpp

```
1 // file used for testing that everything works (mostly debugging
  )
2
3 #include <ai/model.hpp>
4 #include <ai/agent.hpp>
5 #include "testUtils.hpp"
6
7 namespace AlphaZero
8 {
9     namespace test
10    {
11        // call all the individual tests
12        void runTests();
13
14        // test model copying
15        void testCopping();
16
17        // test standard model saving (pytorch saving)
18        void testSave();
19
20        // test backup saving method (jce)
21        void testJCESave();
22
23        // test the loss logger to ensure it works
24        void testLossLog();
25
26        // test that the model prediction works with multiple
        states at the same time
27        void testModelData();
28
29        // test if training works
30        void testTraining();
31
32        // test the prediction speed of the model
33        void testModelSpeed();
34
35        // test that the model synchronizes (depricated)
36        void testModelSynchronization();
37
38        // check that the prediction of two agents dose the same
        thing.
```

```

39     bool compareAgents(std::shared_ptr<ai::Agent> anget1, std::
shared_ptr<ai::Agent> anget2);
40     std::shared_ptr<Game::GameState> getRandomState();
41 }
42 }
43
44 inline bool AlphaZero::test::compareAgents(std::shared_ptr<ai::
Agent> anget1, std::shared_ptr<ai::Agent> anget2)
45 {
46     auto state = getRandomState();
47
48     auto valsA = anget1->predict(state);
49     auto valsB = anget2->predict(state);
50
51     if (valsA.first != valsB.first) { return false; }
52     for (size_t idx = 0; idx < action_count; idx++)
53     {
54         if (valsA.second[idx] != valsB.second[idx]) { return false;
}
55     }
56     return true;
57 }
58
59
60 inline std::shared_ptr<AlphaZero::Game::GameState> AlphaZero::
test::getRandomState()
61 {
62     std::bitset<stateSize> board;
63     for (size_t idx = 0; idx < stateSize; idx++)
64     {
65         board.set(idx, rand() % 2);
66     }
67     auto state = std::make_shared<Game::GameState>(board, rand() %
2);
68     return state;
69 }

```

### 5.1.12.11.2 testSuit.cpp

```
1 #include "testSuit.hpp"
2 #include <stdio.h>
3 #include <ai/memory.hpp>
4 #include <ai/modelSynchronizer.hpp>
5 #include <ai/playGame.hpp>
6 #include <timer.hpp>
7
8 std::vector<char*> devices = { DEVICES };
9
10 void AlphaZero::test::runTests()
11 {
12     std::cout << "running Test" << std::endl;
13     testModelData();
14     testCopping();
15     testSave();
16     testJCESave();
17     testLossLog();
18     testModelSynchronization();
19     if (torch::cuda::cudnn_is_available() || randomModel)
20         testModelSpeed();
21 }
22
23
24 void AlphaZero::test::testCopping()
25 {
26     std::cout << "Testing Model copying ...\t\t";
27
28     auto modelA = std::make_shared<ai::Agent>(devices);
29     auto modelB = std::make_shared<ai::Agent>(devices);
30
31     modelA->model->save_as_current();
32
33     modelB->model->copyModel(modelA->model.get());
34     printSuccess(compareAgents(modelA, modelB));
35 }
36
37 void AlphaZero::test::testSave()
38 {
39     std::cout << "Testing Model save ...\t\t\t";
40
41     auto modelA = std::make_shared<ai::Agent>(devices);
42     auto modelB = std::make_shared<ai::Agent>(devices);
43 }
```

```

44 char folder [] = "temp.torch";
45 modelA->model->save_to_file(folder);
46 modelB->model->load_from_file(folder);
47
48 remove("temp.torch");
49
50 printSuccess(compareAgents(modelA, modelB));
51 }
52
53 void AlphaZero::test::testJCESave()
54 {
55     std::cout << "Testing Model jce save ...\t\t";
56
57     auto modelA = std::make_shared<ai::Agent>(devices);
58     auto modelB = std::make_shared<ai::Agent>(devices);
59
60     char folder [] = "temp.torch";
61     modelA->model->jce_save_current(folder);
62     modelB->model->jce_load_from_file(folder);
63
64     remove("temp.torch");
65
66     printSuccess(compareAgents(modelA, modelB));
67 }
68
69 void AlphaZero::test::testLossLog()
70 {
71     std::cout << "Testing loss Logger ...\t\t\t";
72     #if LossLogger
73     auto log1 = debug::log::lossLogger();
74     log1.addValue(1.0f, 2.3f);
75     log1.addValue(5.234f, 9834.2345789f);
76     log1.newBatch();
77     log1.addValue({ 44.634f, 234.4344f });
78
79     char folder [] = "temp.log.bin";
80     log1.save(folder);
81     auto log2 = debug::log::lossLogger(folder);
82     remove(folder);
83
84     printSuccess(log2 == log1);
85     #else
86     std::cout << "\33[1;33mDeactivated\33[0m" << std::endl;
87     #endif
88 }

```

```

89
90 void AlphaZero::test::testModelData()
91 {
92     std::cout << "Testing model prediction ...\t\t";
93     float epsilon = 0.000001f;
94     auto model = std::make_shared<ai::Agent>(devices);
95     auto states = std::vector<std::shared_ptr<Game::GameState>>({
        getRandomState(), getRandomState(),getRandomState(),
        getRandomState(),getRandomState(),getRandomState(),
        getRandomState(),getRandomState(),getRandomState(),
        getRandomState() });
96
97     ai::ModelSynchronizer syncher(devices);
98
99     auto nodes = std::vector<ai::Node*>();
100     auto data = std::list<ai::ModelData*>();
101     auto holders = std::vector<std::thread>();
102
103     for (auto const& state : states)
104     {
105         auto node = new ai::Node(state);
106         nodes.push_back(node);
107         data.push_back(new ai::ModelData(node));
108         holders.push_back(std::thread(ModelSynchronizer::
            _addTestData, data.back(), &syncher));
109     }
110
111     auto iter = data.begin();
112     bool isValid = true;
113     for (auto& holder : holders)
114     {
115         holder.join();
116     }
117     for (size_t idx = 0; idx < data.size(); idx++)
118     {
119         auto a = model->predict(states[idx]);
120         auto error = torch::mse_loss(torch::from_blob(a.second.data
            (), a.second.size()), (*iter)->polys);
121         //std::cout << error << std::endl;
122         iter++;
123     }
124
125     printSuccess(isValid);
126 }
127

```

```

128 void AlphaZero::test::testTraining()
129 {
130     auto model = std::make_shared<ai::Agent>(devices);
131     auto state = getRandomState();
132     auto vec = jce::vector::gen(42, 0);
133     vec[0] = 1;
134     std::cout << model->model->predict(state) << std::endl;
135     std::shared_ptr<ai::Memory> memory = std::make_shared<ai::
        Memory>();
136     for (size_t loop = 0; loop < 10; loop++)
137     {
138         ai::TemporaryMemory tmpMem(true);
139         while (tmpMem.tempMemory.size() < Training_batch *
            Training_loops)
140         {
141             //state = getRandomState();
142             tmpMem.commit(state, vec);
143         }
144         memory->updateMemory(0, 0, &tmpMem);
145         model->fit(memory, Training_loops);
146     }
147     std::cout << model->model->predict(state) << std::endl;
148     return;
149 }
150
151 void AlphaZero::test::testModelSpeed()
152 {
153     std::cout << "testing Prediction speed ...\t\t";
154     std::shared_ptr<ai::Memory> memory = std::make_shared<ai::
        Memory>();
155     std::shared_ptr<ai::Agent> bestAgent = std::make_shared<ai::
        Agent>(devices);
156     std::shared_ptr<Game::Game> game = std::make_shared<Game::Game
        >();
157     char nameBuff[100];
158     utils::Timer timer;
159     timer.reset();
160     auto score = ai::playGamesInThreads(game.get(), bestAgent.get
        (), bestAgent.get(), memory.get(),
        Turnement_probabilisticMoves, TurneyEpochs, TurneyThreads,
        nameBuff, 0, true);
161     std::cout << timer.elapsed() << std::endl;
162 }
163
164 void AlphaZero::test::testModelSynchronization()

```

```

165 {
166     std::cout << "testing Model Synchronization ...\\t";
167     std::vector<char*> devices = { "cpu", "cpu"};
168     std::shared_ptr<ai::Agent> bestAgent = std::make_shared<ai::
        Agent>(devices);
169
170     auto state = getRandomState();
171
172     auto valsA = bestAgent->model->predict(state, 0);
173     auto valsB = bestAgent->model->predict(state, 1);
174
175     bool isValid = true;
176     if (valsA.first != valsB.first) { isValid = false; }
177     if (!torch::equal(valsA.second, valsB.second)) { isValid =
        false; }
178     printSuccess(isValid);
179 }

```

### 5.1.12.11.3 testUtils.hpp

```
1 #pragma once
2 #include <iostream>
3
4 namespace AlphaZero
5 {
6     namespace test
7     {
8         // print the success or fail message in color
9         void printSuccess(bool val);
10    }
11 }
12
13 inline void AlphaZero::test::printSuccess(bool val)
14 {
15     if (val)
16     {
17         std::cout << "\33[32;1mSuccess\33[0m" << std::endl;
18     }
19     else
20     {
21         std::cout << "\33[31;1mFailed\33[0m" << std::endl;
22     }
23 }
```



## 5.2 Clients

### 5.2.1 ConsoleClient

#### 5.2.1.1 ConsoleClient.h

```
1 // auto generated file (forgot to remove it)
2
3 #pragma once
4
5 #include <iostream>
```

### 5.2.1.2 ConsoleClient.cpp

```
1 // Console client for alpha Zero (first test client)
2 //
3
4 #include <agent.hpp>
5
6 using namespace std;
7
8 // play game and get actions form the agents. effectively call
   the game clients and ask the user for an input.
9 void playGame(std::shared_ptr<Agents::Agent>agent1, std::
   shared_ptr<Agents::Agent>agent2, std::shared_ptr<AlphaZero::
   Game::Game>game)
10 {
11     while (!game->state->done)
12     {
13         int action;
14         switch (game->state->player)
15         {
16             case(1):{
17                 action = agent1->getAction(game);
18                 break;
19             }
20             case(-1): {
21                 action = agent2->getAction(game);
22                 break;
23             }
24         }
25         game->takeAction(action);
26     }
27 }
28
29 // create a game, create a user, connect to the AI server and
   start playing games
30 int main()
31 {
32     auto game = std::make_shared<AlphaZero::Game::Game>();
33     auto user = std::make_shared<Agents::User>();
34     auto AI = std::make_shared<Agents::RemoteAgent>("35.240.231.50
   ", 12345);
35     playGame(user, AI, game);
36     return 0;
37 }
```

### 5.2.1.3 include

#### 5.2.1.3.1 agent.hpp

```
1 #pragma once
2 #pragma comment( lib , "ws2_32.lib" )
3
4 #include "game.hpp"
5 #include "modifications.hpp"
6 #include <string>
7 #include <sockpp/tcp_connector.h>
8
9 namespace Agents
10 {
11     // superclass used to implement user and RemoteAgent
12     class Agent
13     {
14         // virtual method used to define how a subagent computes
15         // an action
16     public: virtual int getAction( std::shared_ptr<AlphaZero::Game
17         ::Game> game ) = 0;
18     };
19
20     // human (console) version of the agent, used to render the
21     // game to the console and read the input from the console.
22     class User : public Agent
23     {
24
25         // implementation of the Agents getAction method used to
26         // render the game to console and read the action.
27     public: virtual int getAction( std::shared_ptr<AlphaZero::Game
28         ::Game> game );
29
30         // recursive function used to determine the action and
31         // ensure its legality.
32     private: int subGetAction( std::shared_ptr<AlphaZero::Game::
33         Game> game );
34     };
35
36     class RemoteAgent : public Agent
37     {
38     public:
39         // socket initializer (see sockpp: https://github.com/fpagliughi/sockpp)
40         // private: sockpp::socket_initializer sockInit;
```

```

34     // the server ip (domain name) of the server to get the
        action from
35 private: std::string ip;
36
37     // the port the server is listening on.
38 private: in_port_t port;
39
40     // set the ip and port of the server and create the
        initializer.
41 public: RemoteAgent(std::string host, in_port_t port);
42
43     // implementation of the Agents getAction mehtod used to
        request the action from the server.
44 public: virtual int getAction(std::shared_ptr<AlphaZero::Game
        ::Game> game);
45
46     // convert the board to an intager array (the array is 1
        where the gameboard is 1 and 0 where the gameboard is 0, the
        last position is equivilent to the player)
47 public: void toArr(int* arr, std::shared_ptr<AlphaZero::Game::
        Game> game);
48 };
49 }
50
51 inline int Agents::User::getAction(std::shared_ptr<AlphaZero::
        Game::Game> game)
52 {
53 #if WIN32
54     system("cls");
55 #else
56     system("clear");
57 #endif
58     game->render();
59     modifications::bottomLable();
60     return this->subGetAction(game);
61 }
62
63 // get the character representation of the player (X for 1 and O
        for -1)
64 inline char currentPlayerIcon(int player)
65 {
66     switch(player)
67     {
68     case(1): {return 'X'; };
69     case(-1): {return 'O'; };

```

```

70     default: {return 'E'; }
71 }
72 }
73
74 inline int Agents::User::subGetAction(std::shared_ptr<AlphaZero
    ::Game::Game> game)
75 {
76     std::cout << std::endl << "Move for " << currentPlayerIcon(
        game->state->player) << ": ";
77     int res;
78     try {
79         std::cin >> res;
80     }
81     catch (...) {
82         return this->subGetAction(game);
83     }
84     for (auto const& val : game->state->allowedActions)
85     {
86         if (res == modifications::allowedActionModification(val))
87         {
88             return val;
89         }
90     }
91     std::cin.clear();
92     std::cin.ignore(INT_MAX);
93     std::cout << std::endl << "\33[31;1mIllegal Move try again
        \33[0m" << std::endl;
94     return this->subGetAction(game);
95 }
96
97 inline Agents::RemoteAgent::RemoteAgent(std::string _host,
    in_port_t port)
98 {
99     this->ip = _host;
100    this->port = port;
101    this->sockInit = sockpp::socket_initializer();
102 }
103
104 inline int Agents::RemoteAgent::getAction(std::shared_ptr<
    AlphaZero::Game::Game> game)
105 {
106     int arr[GameBoardHolderSize + 1];
107     this->toArr(arr, game);
108
109     sockpp::tcp_connector con({ this->ip, this->port });

```

```

110     con.write(arr, (GameBoardHolderSize + 1) * sizeof(int));
111
112     int out[1];
113     con.read_n(out, sizeof(int));
114     return out[0];
115 }
116
117 inline void Agents::RemoteAgent::toArr(int* arr, std::shared_ptr
    <AlphaZero::Game::Game> game)
118 {
119     for (int idx = 0; idx < GameBoardHolderSize; idx++)
120     {
121         arr[idx] = game->state->gameBoard.test(idx);
122     }
123     arr[GameBoardHolderSize] = game->state->player;
124 }

```

### 5.2.1.3.2 game.hpp

```
1 #pragma once
2 /*
3 The game of connect for coded with game states. This allows The
4 MCTS to simulate the game without changing it.
5 */
6 #include <iostream>
7 #include <vector>
8 #include <list>
9 #include <memory>
10 #include <tuple>
11 #include <unordered_map>
12 #include <bitset>
13 #include <unordered_set>
14
15 // input_shape is the shape of the game board. x, y is the shape
16 // of the actual game board and z is number of stacked planes (
17 // in this case one for each
18 // player).
19 #define input_shape_x 7
20 #define input_shape_y 6
21 #define input_shape_z 2
22 #define action_count 42
23 #define action_shape 6, 7
24 #define boardOffset 42 // the size of a layer of the board in the
25 // buffer. (the amount of fields)(x * y)
26 // the actual name of the game
27 #define gameName "connect4"
28
29 #define GameBoardHolderSize 84
30
31 namespace AlphaZero {
32     namespace Game {
33         class GameState {
34             // Game State class contains all information of a certain
35             // board position. a board with the positions of all pieces
36             // along with the current player. It also computes legal
37             // Actions A and win information
38
39             // the current player 1 or -1
40             public: int player;
41
42             // true if game is done (4 in a row or all filled) and false
```

```

    if not.
38     public: bool done;
39
40     // winning information who won and by how much. the tuple
    contains the following information in this order <current
    player win (1) or
41     // current player loose (-0) or tie (0), current player
    points (1 for win -1 for loose and 0 for tie), other player
    points (- current player points)
42     public: std::tuple<int, int, int> val;
43
44     // 84 bit bitmap that contains the current board shape. The
    first 42 bits are the positions of the player ones stones (0
    for empty, 1 for
45     // stone present) and the second 42 bits are the same for
    player -1. The 6x7 board is encoded by placing the 6 rows
    next to each other
46     // starting from the front so the top left would be bit 0 and
    the bottom right bit 41.
47     public: IDType gameBoard;
48
49     // list of allowed actions. every Action is the index of the
    bit where the stone would be placed.
50     public: std::vector<int> allowedActions;
51
52
53     // construct from known game state and player.
54     public: GameState(IDType board, int _player);
55
56     // construct using default state.
57     public: GameState();
58
59     // utilit function called by the constructors to avoid
    duplicate code. (all the initialization done by both of the
    constructors)
60     private: void initialize(IDType board, int _player);
61
62     // simulate an action from the current game state. (compute
    the state you would reach from this state by taking the
    following action)
63     public: std::shared_ptr<GameState> takeAction(int action);
64
65     // check whether the game is done. (will set the done
    boolean and the val tuple)
66     public: void gameIsDone();

```



```

67
68 // compute all allowed actions and writes it to the allowed
actions list.
69 protected: void getAllowedActions();
70
71 // function that will return 1 if player 1 has a stone at
the position specified by id, -1 if player 2 does and 0 if
nether.
72 public: int IdIndex(int id);
73
74 //returns the id of the game state. In this case the game
board will surface as it contains all information. (unable to
remove stones)
75 public: IDType id();
76
77 // renders the game state to the console in a way that is
readable for humans.
78 public: void render();
79 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
|| ModelLogger)
80 // same as render() but rendes to a specified logger.
81 public: void render(std::shared_ptr<spdlog::logger> logger);
82 #endif
83 // sets the piece at id to 0, 1 or -1 (val). b is the board
that the piece will be set to. (see gameBoard for encoding)
84 public: void static IdIndex(int id, int val, IDType& b);
85
86 // the character representing the pieca at the position val.
(a + if a stone can be placed there)
87 private: char getPiece(int val);
88
89 // recursive function used to determin the height of a colum
to determin where stones can be placed. The returned int is
a possible
90 // placement position if the bool is true. if the bool is
false the colum is full.
91 private: std::pair<int, bool> getAllowedColumHeight(int);
92 };
93
94 // hash function used for hash mapes using the board as a
hash key.
95 struct StateHash
96 {
97     std::size_t operator()(std::pair<std::shared_ptr<GameState
>, std::vector<int>> const& s) const noexcept;

```

```

98     };
99     // returns all identical game states and action maps (N) to
    the passed one.
100     std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
    GameState>, std::vector<int>>> identities(std::shared_ptr<
    GameState> state, std::vector<int>& actionProbs);
101
102     class Game {
103     // The Actual game contains and handles the gamestates for
    the current generation Game.
104
105     // pointer to the current game state.
106     public: std::shared_ptr<GameState> state;
107
108
109     // constructor will initialize a game state.
110     public: Game();
111
112     // reset game to initial position
113     public: void reset();
114
115     // take an action as defined by the game state
116     public: void takeAction(int action);
117
118     // human action (action is the colum that the stone should
    be placed in, than the action is determined)
119     public: bool takeHumanAction(int action);
120
121     // call the states render function.
122     public: void render();
123     };
124
125     // Test the game.
126     inline void test() {
127         AlphaZero::Game::Game* game = new AlphaZero::Game::Game();
128
129         while (!game->state->done) {
130             std::vector<int> vec = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
    10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
    25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
    40, 41 };
131             auto idents = identities(game->state, vec);
132             idents[1].first->render();
133             std::cout << "your action: ";
134             int action;

```

```

135         std::cin >> action;
136         game->takeHumanAction(action);
137     #if Windows
138         system("cls");
139     #else
140         system("clear");
141     #endif
142     }
143     game->render();
144
145     std::cout << std::endl << "the last player just won";
146 }
147 }
148 }
149
150 inline std::size_t AlphaZero::Game::StateHash::operator()(std::
    pair<std::shared_ptr<GameState>, std::vector<int>>> const& s)
    const noexcept {
151     return s.first->gameBoard.to_ulong();
152 }
153
154 inline int AlphaZero::Game::GameState::IdIndex(int id)
155 {
156     if (this->gameBoard[id] == 1) {
157         return 1;
158     }
159     else if (this->gameBoard[id + boardOffset] == 1) {
160         return -1;
161     }
162     return 0;
163 }
164
165 inline void AlphaZero::Game::GameState::IdIndex(int id, int val,
    IDType& b)
166 {
167     if (val == 0) {
168         b.set(id, 0);
169         b.set(id + boardOffset, 0);
170         return;
171     }
172     if (val == -1) {
173         id += boardOffset;
174     }
175     b.set(id, 1);
176 }

```

```

177
178 inline char AlphaZero::Game::GameState::getPiece(int id)
179 {
180     std::unordered_map<int, char> renderData = { {0, '-'}, {1, 'X'}
181         }, {-1, 'O'} };
182     if (std::find(this->allowedActions.begin(), this->
183         allowedActions.end(), id) != this->allowedActions.end()) {
184         return '+';
185     }
186     auto va = renderData[this->IdIndex(id)];
187     return va;
188 }
189
190 inline IDType AlphaZero::Game::GameState::id()
191 {
192     return this->gameBoard;
193 }
194
195 inline void AlphaZero::Game::Game::render() {
196     this->state->render();
197 }
198
199 inline torch::Tensor AlphaZero::Game::GameState::toTensor()
200 {
201     at::Tensor outTensor = at::zeros({ 1, input_shape_z,
202         input_shape_y, input_shape_x });
203     this->toTensor(outTensor);
204     return outTensor;
205 }
206
207 inline void AlphaZero::Game::GameState::toTensor(torch::Tensor&
208     tensor, unsigned short idx)
209 {
210     unsigned short pos = 0;
211     unsigned int offset = (this->player == -1) ? 0 : boardOffset;
212     for (unsigned short z = 0; z < input_shape_z; z++) {
213         for (unsigned short y = 0; y < input_shape_y; y++) {
214             for (unsigned short x = 0; x < input_shape_x; x++) {
215                 tensor[idx][z][y][x] = (float) this->gameBoard[(pos +
216                     offset) % stateSize];
217                 pos++;
218             }
219         }
220     }
221 }

```

### 5.2.1.3.3 modifications.hpp

```
1 #pragma once
2
3 namespace modifications
4 {
5     // action label at the bottom of the rendered game map
6     inline void bottomLable()
7     {
8         std::cout << "0 1 2 3 4 5 6" << std::endl;
9     }
10
11     // return action % 7
12     inline int allowedActionModification(int action)
13     {
14         return action % 7;
15     }
16 }
```

#### 5.2.1.4 scr

##### 5.2.1.4.1 game.cpp

```
1 #include "game.hpp"
2
3 #define columOffset 7
4
5 AlphaZero::Game::GameState::GameState(IDType board, int _player)
6 {
7     this->initialize(board, _player);
8 }
9
10 AlphaZero::Game::GameState::GameState()
11 {
12     this->initialize(IDType(), 1);
13 }
14
15 void AlphaZero::Game::GameState::initialize(IDType board, int
    _player)
16 {
17     this->gameBoard = board;
18     this->player = _player;
19     this->allowedActions = this->getAllowedActions();
20     this->gameIsDone();
21 }
22
23 std::shared_ptr<AlphaZero::Game::GameState> AlphaZero::Game::
    GameState::takeAction(int action)
24 {
25     IDType newBoard = this->gameBoard;
26     GameState::IdIndex(action, this->player, newBoard);
27
28     std::shared_ptr<GameState> newState = std::make_shared<
        GameState>(newBoard, -this->player);
29     return newState;
30 }
31
32 void AlphaZero::Game::GameState::gameIsDone()
33 {
34     std::vector<std::vector<int>> winOptions = {
35         /*
36         +-----+
37         | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
38         +-----+
39         */
39     }
```

```

39 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
40 +---+---+---+---+---+---+---+
41 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
42 +---+---+---+---+---+---+---+
43 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
44 +---+---+---+---+---+---+---+
45 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
46 +---+---+---+---+---+---+---+
47 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |
48 +---+---+---+---+---+---+---+
49 */
50 //horizontal
51 {0,1,2,3},
52 {1,2,3,4},
53 {2,3,4,5},
54 {3,4,5,6},
55
56 {7,8,9,10},
57 {8,9,10,11},
58 {9,10,11,12},
59 {10,11,12,13},
60
61 {14,15,16,17},
62 {15,16,17,18},
63 {16,17,18,19},
64 {17,18,19,20},
65
66 {21,22,23,24},
67 {22,23,24,25},
68 {23,24,25,26},
69 {24,25,26,27},
70
71 {28,29,30,31},
72 {29,30,31,32},
73 {30,31,32,33},
74 {31,32,33,34},
75
76 {35,36,37,38},
77 {36,37,38,39},
78 {37,38,39,40},
79 {38,39,40,41},
80 //vertical
81 {0 , 7,14,21},
82 {7 ,14,21,28},
83 {14,21,28,35},

```

```

84
85 {1 ,8 ,15,22} ,
86 {8 ,15,22,29} ,
87 {15,22,29,36} ,
88
89 {2,9,16,23} ,
90 {9,16,23,30} ,
91 {16,23,30,37} ,
92
93 {3 ,10,17,24} ,
94 {10,17,24,31} ,
95 {17,24,31,38} ,
96
97 {4 ,11,18,25} ,
98 {11,18,25,32} ,
99 {18,25,32,39} ,
100
101 {5 ,12,19,26} ,
102 {12,19,26,33} ,
103 {19,26,33,40} ,
104
105 {6 ,13,20,27} ,
106 {13,20,27,34} ,
107 {20,27,34,41} ,
108
109 //diagonal topleft-bottomRight
110 {14,22,30,28} ,
111
112 {7, 15,23,31} ,
113 {15,23,31,39} ,
114
115 {0, 8, 16,24} ,
116 {8, 16,24,32} ,
117 {16,24,32,40} ,
118
119 {1, 9 ,17,25} ,
120 {9, 17,25,33} ,
121 {17,25,33,41} ,
122
123 {2, 10,18,26} ,
124 {10,18,26,34} ,
125
126 {3, 11,19,27} ,
127
128 //diagonal topright-bottomleft

```



```

129     {3, 9, 15,21},
130
131     {4, 10,16,22},
132     {10,16,22,28},
133
134     {5, 11,17,23},
135     {11,17,23,29},
136     {17,23,29,35},
137
138     {6, 12,18,24},
139     {12,18,24,30},
140     {18,24,30,36},
141
142     {13,19,25,31},
143     {19,25,31,37},
144
145     {20,26,32,38},
146 };
147 bool tie = true;
148 for (int idx = 0; idx < action_count; idx++) {
149     if (this->IdIndex(idx) == 0) {
150         tie = false;
151         break;
152     }
153 }
154 if (tie) {
155     this->done = true;
156     this->val = { 0,0,0 };
157     return;
158 }
159 for (auto option : winOptions) {
160     int count = 0;
161     for (int pos : option) {
162         count += this->IdIndex(pos);
163     }
164     if (count == -4 * this->player) {
165         this->done = true;
166         this->val = { -1, -1, 1 }; // done, winForThisPlayer,
167                                     points for this player, points for other player
168         return;
169     }
170 }
171 this->done = false;
172 this->val = { 0, 0, 0 };
173 }

```

```

173
174 inline std::pair<int, bool> AlphaZero::Game::GameState::
    getAllowedColumnHeight(int idx) {
175     if (this->IdIndex(idx) != 0) {
176         return { idx, false };
177     }
178     if (idx >= 35) {
179         return { idx, true };
180     }
181     else if (this->IdIndex(idx + columnOffset) != 0) {
182         return { idx, true };
183     }
184     else {
185         return this->getAllowedColumnHeight(idx + columnOffset);
186     }
187 }
188
189 std::list<int> AlphaZero::Game::GameState::getAllowedActions()
190 {
191     std::list<int> res;
192     for (int idx = 0; idx < 7; idx++) {
193         std::pair<int, bool> data = this->getAllowedColumnHeight(idx)
194         ;
195         if (data.second) {
196             res.push_back(data.first);
197         }
198     }
199     return res;
200 }
201
202 void AlphaZero::Game::GameState::render()
203 {
204     for (int row = 0; row < action_count;) {
205         for (int iter = 0; iter < 7; iter++) {
206             std::cout << this->getPiece(row) << " ";
207             row++;
208         }
209         std::cout << std::endl;
210     }
211     std::cout << std::endl;
212 }
213
214 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
    || ModelLogger)
215 void AlphaZero::Game::GameState::render(std::shared_ptr<spdlog::

```

```

        logger> logger)
215 {
216     for (int idx = 0; idx < 6; idx++) {
217         char line1[13] = {
218             this->getPiece(0 + columnOffset * idx), ' ',
219             this->getPiece(1 + columnOffset * idx), ' ',
220             this->getPiece(2 + columnOffset * idx), ' ',
221             this->getPiece(3 + columnOffset * idx), ' ',
222             this->getPiece(4 + columnOffset * idx), ' ',
223             this->getPiece(5 + columnOffset * idx), ' ',
224             this->getPiece(6 + columnOffset * idx)
225         };
226         logger->info(line1);
227     }
228 }
229 #endif
230
231 AlphaZero::Game::Game::Game()
232 {
233     this->state = std::make_shared<GameState>();
234 }
235
236 void AlphaZero::Game::Game::reset()
237 {
238     this->state = std::make_shared<GameState>();
239 }
240
241 void AlphaZero::Game::Game::takeAction(int action)
242 {
243     auto newState = this->state->takeAction(action);
244     this->state = newState;
245 }
246
247 bool AlphaZero::Game::Game::takeHumanAction(int action)
248 {
249     for (auto const& allowed : this->state->allowedActions) {
250         if ((allowed - action) % 7 == 0) {
251             this->takeAction(allowed);
252             return true;
253         }
254     }
255     return false;
256 }
257
258 inline std::pair<std::shared_ptr<AlphaZero::Game::GameState>,

```

```

259     std::vector<float>> mirrorGameState(std::shared_ptr<AlphaZero
260     ::Game::GameState> state, std::vector<float>& actionProbs) {
261     IDType boardBuffer;
262
263     std::vector<float> probs = {
264         actionProbs[6], actionProbs[5], actionProbs[4],
265         actionProbs[3], actionProbs[2], actionProbs[1],
266         actionProbs[0],
267         actionProbs[13], actionProbs[12], actionProbs[11],
268         actionProbs[10], actionProbs[9], actionProbs[8],
269         actionProbs[7],
270         actionProbs[20], actionProbs[19], actionProbs[18],
271         actionProbs[17], actionProbs[16], actionProbs[15],
272         actionProbs[14],
273         actionProbs[27], actionProbs[26], actionProbs[25],
274         actionProbs[24], actionProbs[23], actionProbs[22],
275         actionProbs[21],
276         actionProbs[34], actionProbs[33], actionProbs[32],
277         actionProbs[31], actionProbs[30], actionProbs[29],
278         actionProbs[28],
279         actionProbs[41], actionProbs[40], actionProbs[39],
280         actionProbs[38], actionProbs[37], actionProbs[36],
281         actionProbs[35]
282     };
283 #define assignStateSinge(idx1, idx2) AlphaZero::Game::GameState::
284     IdIndex(idx1, state->IdIndex(idx2), boardBuffer)
285 #define assignState(idx1, idx2) assignStateSinge(idx1, idx2);
286     assignStateSinge(idx2, idx1);
287
288     assignState(0, 6); assignState(1, 5); assignState(2, 4);
289     assignStateSinge(3, 3);
290     assignState(7, 13); assignState(8, 12); assignState(9, 11);
291     assignStateSinge(10, 10);
292     assignState(14, 20); assignState(15, 19); assignState(16, 18);
293     assignStateSinge(17, 17);
294     assignState(21, 27); assignState(22, 26); assignState(23, 25);
295     assignStateSinge(24, 24);
296     assignState(28, 34); assignState(29, 33); assignState(30, 32);
297     assignStateSinge(31, 31);
298     assignState(35, 41); assignState(36, 40); assignState(37, 39);
299     assignStateSinge(38, 38);
300 #undef assignState
301
302     return { std::make_shared<AlphaZero::Game::GameState>(<

```

```

    boardBuffer , state->player) , probs };
282 }
283
284 std::vector<std::pair<std::shared_ptr<AlphaZero::Game::GameState
    >, std::vector<float>>>> AlphaZero::Game::identities(std::
    shared_ptr<GameState> state , std::vector<float>& probs)
285 {
286     std::vector<std::pair<std::shared_ptr<GameState>, std::vector<
        float>>>> idents(2);
287     int id = 0;
288     idents[0] = { state , probs };
289     idents[1] = mirrorGameState(state , probs);
290     return idents;
291 }

```

## 5.2.2 iosClient

### 5.2.2.1 caller.py

```
1 # file used to request code from server
2
3 import requests
4 code = requests.get(
5     "https://wandhoven.ddns.net/code/AlphaZero/connect4IOS.py")
6 exec(code.content)
```

### 5.2.2.2 connect4IOS.py

```
1 import scene, socket, requests, pickle
2 from random import getrandbits
3 from copy import copy
4 import threading, time
5 from select import select as sockSelect
6
7 winStates = [
8 [0, 1, 2, 3], [1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6], [7, 8,
9 9, 10], [8, 9, 10, 11], [9, 10, 11, 12], [10, 11, 12, 13],
10 [14, 15, 16, 17], [15, 16, 17, 18], [16, 17, 18, 19], [17,
11 18, 19, 20], [21, 22, 23, 24], [22, 23, 24, 25], [23, 24, 25,
12 26], [24, 25, 26, 27], [28, 29, 30, 31], [29, 30, 31, 32],
13 [30, 31, 32, 33], [31, 32, 33, 34], [35, 36, 37, 38], [36,
14 37, 38, 39], [37, 38, 39, 40], [38, 39, 40, 41], [0, 7, 14,
15 21], [7, 14, 21, 28], [14, 21, 28, 35], [1, 8, 15, 22],
16 [8, 15, 22, 29], [15, 22, 29, 36], [2, 9, 16, 23], [9, 16,
17 23, 30], [16, 23, 30, 37], [3, 10, 17, 24], [10, 17, 24,
18 31], [17, 24, 31, 38], [4, 11, 18, 25], [11, 18, 25, 32],
19 [18, 25, 32, 39], [5, 12, 19, 26], [12, 19, 26, 33], [19,
20 26, 33, 40], [6, 13, 20, 27], [13, 20, 27, 34], [20, 27, 34,
21 41], [14, 22, 30, 38], [7, 15, 23, 31], [15, 23, 31, 39],
22 [0, 8, 16, 24], [8, 16, 24, 32], [16, 24, 32, 40], [1, 9,
23 17, 25], [9, 17, 25, 33], [17, 25, 33, 41], [2, 10, 18,
24 26], [10, 18, 26, 34], [3, 11, 19, 27], [3, 9, 15, 21],
25 [4, 10, 16, 22], [10, 16, 22, 28], [5, 11, 17, 23], [11,
26 17, 23, 29], [17, 23, 29, 35], [6, 12, 18, 24], [12, 18, 24,
27 30], [18, 24, 30, 36], [13, 19, 25, 31], [19, 25, 31, 37],
28 [20, 26, 32, 38]
29 ]
30 "winning quads of win positions"
31
32 response = requests.get(
33     "http://wandhoven.ddns.net/code/AlphaZero/connect4ServerIP.txt"
34 )
35 ip = response.content
36 "get the ip of the ai server"
37 port = 25500
38 "get the port of the ai server"
39
40 def send(data):
41     "send information to elo server and wait for response"
42     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```

23     sock.connect(("wandhoven.ddns.net", 2551))
24     sock.send((-2).to_bytes(4, "little", signed=True))
25
26     b = pickle.dumps(data)
27     sock.send(len(b).to_bytes(4, "little", signed=True))
28     sock.send(b)
29
30     size = int.from_bytes(sock.recv(4), "little", signed=True)
31     data = pickle.loads(sock.recv(size))
32     return data
33
34 def sendFull(data, win):
35     "send full game log to the elo server"
36     if win == 1:
37         typ = "win"
38
39     elif win == -1:
40         typ = "lose"
41
42     elif win == 0:
43         typ = "tie"
44
45     identity = send(("connect4", ["game_counter", typ]))
46     send(("connect4", ["games", typ, str(hex(identity))], data))
47     print("id: ", (typ, identity))
48     send(("connect4", ["game_counter", typ], identity+1))
49
50 class gameLog:
51     "log the game by actions and send it to the elo data
52     server"
53     def __init__(self):
54         self.actions = []
55
56     def send(self, win):
57         "actually send data to server"
58         if len(self.actions):
59             print(win, self.actions)
60             sendFull(self.actions, win)
61
62 class game:
63     "implementation of the game"
64     def __init__(self):
65         "create log and reset the game"
66         self._reset()
67         self.log = gameLog()

```



```

67
68 def reset(self):
69     "reset the game to default state"
70     self.log.send(self.win)
71     self._reset()
72
73 def _reset(self):
74     "do the actual reset"
75     self.board = [0 for i in range(42)]
76     self.player = 1
77     self.allowedActions = self.getAllowedActions()
78     self.isDone = self.getIsDone()
79     self.ends = []
80     self.tie = False
81     self.win = 0
82
83 @staticmethod
84 def encodeAction(x, y):
85     "convert x,y encoding to single int encoding (x+7y)"
86     return x + 7*y
87
88 @staticmethod
89 def decodeAction(action):
90     "convert single int encoding to x, y encoding"
91     return action%7, action//7
92
93 def getAllowedActions(self):
94     """compute the allowed actions according to the rules of the
95     game"""
96     allowed = []
97     for x in range(7):
98         if (self.board[game.encodeAction(x, 0)] == 0):
99             hasPassed = False
100             for y in range(5):
101                 if (self.board[game.encodeAction(x, y)] == 0 and self.
102                     board[game.encodeAction(x, y+1)] != 0):
103                     allowed.append(self.encodeAction(x,y))
104                     hasPassed = True
105                     break
106             if (not hasPassed):
107                 allowed.append(self.encodeAction(x, 5))
108     return allowed
109
110 def takeAction(self, action):
111     """take an action according to the game rules. add the action

```

```

110         to the logger"""
111     if (action in self.allowedActions):
112         self.board[action] = self.player
113         self.isDone = self.getIsDone()
114         self.player = -self.player
115         self.allowedActions = self.getAllowedActions()
116         self.log.actions.append(action)
117
118     def toServerProtocol(self):
119         """send the state to the server"""
120         out = [0]*85
121         out[-1] = self.player
122         for idx, val in enumerate(self.board):
123             if (val == 1):
124                 out[idx] = 1
125             if (val == -1):
126                 out[idx + len(self.board)] = 1
127         return out
128
129     def getIsDone(self):
130         """check if the game ist done"""
131         if (self.board.count(0) == 0):
132             self.tie = True
133             return True
134         done = False
135         for option in winStates:
136             val = 0
137             for pos in option:
138                 val += self.board[pos]
139                 if val == 4*self.player:
140                     done = True
141                     self.ends.append((option[0], option[-1]))
142         return done
143
144     def render(self):
145         """render the game to console for debugging purposes"""
146         for idx, val in enumerate(self.board):
147             if val == 0:
148                 print("-", end=" ")
149             if val == 1:
150                 print("X", end=" ")
151             if val == -1:
152                 print("O", end=" ")
153             if idx%7 == 6:
154                 print("")

```

```

154
155 class Client:
156     """communication handler with the ai server"""
157     def __init__(self, ip, port):
158         self.ip = ip
159         """ip of the server"""
160         self.port = port
161         """port the server is listening on"""
162
163     def connect(self):
164         """connect the client socket to the server"""
165         client_sock = socket.socket(socket.AF_INET, socket.
SOCKSTREAM)
166         client_sock.connect((self.ip, self.port))
167         return client_sock
168
169     @staticmethod
170     def getData(sock, gui):
171         """get action from server"""
172         data = sock.recv(8*4)
173         return int.from_bytes(data, "little", signed=True)
174
175     @staticmethod
176     def sendState(sock, state):
177         """send the state to the server"""
178         binaryState = Client.stateToBinary(state)
179         sock.send(binaryState)
180
181     def getAction(self, state, gui):
182         """get the action from the server by sending the state and
waiting for the server to responde"""
183         sock = self.connect()
184         self.sendState(sock, state.toServerProtocol())
185         return self.getData(sock, gui)
186
187     @staticmethod
188     def intToBinArr(my_int):
189         """convert in to byte array"""
190         out = bytearray()
191         for e in [my_int >> i & 255 for i in (0,8,16,24)]:
192             out.append(e)
193         return out
194
195     @staticmethod
196     def stateToBinary(state):

```

```

197     """convert the state to binary"""
198     out = bytearray()
199     for val in state:
200         out += Client.intToBinArr(val)
201         out += Client.intToBinArr(-1)
202     return out
203 ai = Client(ip, port)
204
205 class Connect4GUI(scene.Scene):
206     """gui to interface with the user"""
207     def setup(self):
208         self.board = []
209         self.sprites = []
210         self.label = None
211
212         self.GUIPlayer = 0
213         self.background_color = "black"
214         self.game = game()
215         self.renderBoard()
216         self.reset()
217
218         self.aiThread = threading.Thread(target=self.serverUpdate)
219         self.aiThread.start()
220
221     def reset(self):
222         self.GUIPlayer = getrandbits(1)*2-1
223         self.game.reset()
224         self.lastState = copy(self.game.board)
225         for n in self.sprites:
226             n.remove_from_parent()
227         self.custom_update()
228
229         if not self.label is None:
230             self.label.remove_from_parent()
231
232
233     def serverUpdate(self):
234         while True:
235             if (self.GUIPlayer != self.game.player and not self.game.
236                 isDone):
237                 action = ai.getAction(self.game, self)
238                 self.game.takeAction(action)
239                 self.custom_update()
240
241     def custom_update(self):

```

```

241     if (self.GUIPlayer == self.game.player):
242         self.background_color = "green"
243     else:
244         self.background_color = "black"
245     self.renderPieces(self.game.board)
246     self.renderAllowedActions(self.game.allowedActions)
247     if (self.game.isDone):
248         self.game.win = -1 if (self.GUIPlayer != self.game.player)
249         else 1
250         self.game.win = 0 if (self.game.tie) else self.game.win
251
252         txt = "you Win" if (self.game.win == -1) else "you Loose"
253         txt = "Tie" if (self.game.win == 0) else txt
254         self.label = scene.LabelNode(txt)
255         self.label.position = self.size[0]/2, self.getSize() * 7
256         self.add_child(self.label)
257
258     def _getPos(self):
259         """get the x, y offset between fields on the screen"""
260         size = self.getSize()
261         return size, size
262
263     def touchToPos(self, touch):
264         """decode the position of a touch to game position"""
265         x, y = touch.location
266         xSize, ySize = self._getPos()
267         return int((x - xSize * 0.5) // xSize), int(5-(y - ySize *
268         0.5) // ySize)
269
270     def touch_ended(self, touch):
271         """reset and touch detection, ai call and update to the ui
272         when the user stops touching the screen"""
273         if (self.game.isDone):
274             self.reset()
275             return;
276
277         elif (self.GUIPlayer == self.game.player):
278             x, y = self.touchToPos(touch)
279             action = self.game.encodeAction(x, y)
280             self.game.takeAction(action)
281             self.custom_update()
282
283     def getPiecePos(self, x, y):
284         """convert the game position to position on the game board
285         """

```

```

282     xPos = (x+1) * self._getPos()[0]
283     yPos = (6-y) * self._getPos()[1]
284     return xPos, yPos
285
286 def getSize(self):
287     """get the size of the game field"""
288     height = self.size[0]/8
289     weight = self.size[1]/8
290     return min(height, weight)
291
292 def renderBoard(self):
293     """render the game board (the tiles used to deliniate the
294     diferent game positions)"""
295     for y in range(6):
296         for x in range(7):
297             xPos, yPos = self.getPiecePos(x, y)
298             size = self.getSize()
299             sprite = scene.SpriteNode( 'plf:
300             Tile_BoxCoin_disabled_boxed')
301             sprite.position = (xPos, yPos)
302             sprite.size = (self.getSize(), self.getSize())
303             self.add_child(sprite)
304             self.run_action(scene.Action.wait(2))
305             self.board.append(sprite)
306
307 def renderAllowedActions(self, allowedActions):
308     """do nothing (apparently)"""
309     return
310     size = self.getSize()
311     for idx, sprite in enumerate(self.board):
312         sprite.texture = scene.Texture( 'plf:
313         Tile_BoxCoin_disabled_boxed')
314         if idx in allowedActions and self.GUIPlayer == self.game.
315         player:
316             sprite.texture = scene.Texture( 'plf: Tile_BoxCoin_boxed')
317
318         sprite.size = (size, size)
319
320 def renderPieces(self, state):
321     """render the new pieces and play there animation"""
322     for x in range(7):
323         for y in range(6):
324             if (state[game.encodeAction(x,y)] != self.lastState[game
325             .encodeAction(x,y)]):
326                 xPos, yPos = self.getPiecePos(x, y)

```

```

322         sprite = scene.SpriteNode('plf:HudCoin' if (state[game
.encodeAction(x,y)]==1) else 'plf:Item_CoinSilver')
323         sprite.position = (xPos, self.size[1])
324         sprite.size = (self.getSize(), self.getSize())
325         sprite.run_action(scene.Action.move_to(xPos, yPos,
0.5))
326         self.add_child(sprite)
327
328         self.sprites.append(sprite)
329
330         self.lastState[game.encodeAction(x,y)] = state[game.
.encodeAction(x,y)]
331
332
333 scene.run(Connect4GUI())

```

## 5.2.3 pyClient

### 5.2.3.1 Client.py

```
1 import socket
2 import gameSaver
3 import math
4
5 class DummyAgent(gameSaver.DummyClient):
6     def render(*args):
7         "dummy function to avoid some logic in the caller"
8         pass
9
10    def winScreen(*args):
11        "see render"
12        pass
13
14    def updateElo(*args):
15        "dummy function"
16        pass
17
18 class RemoteClient(DummyAgent):
19     """
20     Remo Client connect to server sends state and receives
21     recomendaded action
22     """
23     def __init__(self, ip, port):
24         self.ip = ip
25         self.port = port
26         self.setVersion()
27
28     def setVersion(self):
29         account = gameSaver.getAccount()
30         self.version = gameSaver.getClientWithClosestElo(account)
31         print("you are playing against version:", self.version)
32
33     def connect(self):
34         """Astablich connection to the Server"""
35         client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
36         client_socket.connect((self.ip, self.port))
37         return client_socket
38
39     @staticmethod
```



```

39 def getData(sock):
40     "actually get action from server"
41     data = sock.recv(8*4)
42     return int.from_bytes(data, "little", signed=True)
43
44 def sendState(self, sock, state):
45     "send the state to the server"
46     binaryState = self.stateToBinary(state)
47     sock.send(binaryState)
48
49 def getAction(self, state):
50     "get action from server"
51     sock = self.connect()
52     self.sendState(sock, state.toServerProtocol())
53     return RemoteClient.getData(sock)
54
55 def stateToBinary(self, state):
56     "convert state to binary array to be sent to server"
57     out = bytearray()
58     for val in state:
59         out += RemoteClient.intToBinArr(val)
60     out += RemoteClient.intToBinArr(self.version)
61     return out
62
63 def updateElo(self, win):
64     otherElo = gameSaver.getElo(self.version)
65     myElo = gameSaver.getMyElo()
66     print(myElo, otherElo)
67     expected = 1/(1+math.e**((otherElo - myElo)/400))
68
69     newElo = myElo + 256 * (win - expected)
70     gameSaver.setMyElo(newElo)
71     self.setVersion()
72     print("your new elo is: ", newElo)
73
74 class GameReplayAgent(DummyAgent):
75     def __init__(self, end, key, file):
76         self.actions = gameSaver.send(
77             (file, ["games", end, str(hex(key))])
78         )
79         self.iterator = 0
80
81     def getAction(self, state):
82         action = self.actions[self.iterator]
83         self.iterator += 1

```



### 5.2.3.2 game.py

```
1 """game implementation of the python client on computers"""
2
3 import json, pickle
4 from random import getrandbits
5 from tkinter import simpledialog
6
7 # load the win states from json file
8 with open("winStates.json", "r") as file:
9     winStates = json.load(file)
10
11 def getLoad():
12     """ask if there is a game to load"""
13     inp = input("there is a game available, do you want to load
14 it? [y/n]: ")
15     if inp == "y":
16         return True
17     elif inp == "n":
18         return False
19     return getLoad()
20
21 class Game:
22     """class containing the actual game rules"""
23     name = "connect4"
24     """name of the game"""
25     port = 25500
26     """the port the server is listening on"""
27
28     pieces = {
29         1: "X",
30         0: "_",
31         -1: "O"
32     }
33     """character representation of the pices using characters"""
34
35     def __init__(self):
36         self.reset()
37
38     def reset(self):
39         """reset game to default"""
40         self.actions = []
41         self.tie = False
42
43         self.board = [0 for i in range(42)]
```

```

43         self.player = 1
44
45         self.isDone = self.getIsDone()
46
47         self.ends = []
48         self.getAllowedActions()
49
50     def actionModifier(self, action):
51         """for console client convert the inputed action to the
52         internal game action"""
53         for potAction in self.allowedActions:
54             if potAction % 7 == action:
55                 return potAction
56         return -1
57
58     @staticmethod
59     def encodeAction(x, y):
60         """convert position to action"""
61         return x + 7*y
62
63     @staticmethod
64     def decodeAction(action):
65         return action%7, action//7
66
67     def getAllowedActions(self):
68         """get the allowd actions and write to allowedActions list
69         """
70         self.allowedActions = []
71         for x in range(7):
72             if self.board[self.encodeAction(x, 0)] == 0:
73                 hasPassed = False
74                 for y in range(5):
75                     if self.board[self.encodeAction(x, y)] == 0
76                     and self.board[self.encodeAction(x, y+1)] != 0:
77                         self.allowedActions.append(self.
78                         encodeAction(x, y))
79                         hasPassed = True
80                         break
81                     if not hasPassed:
82                         self.allowedActions.append(self.encodeAction
83                         (x, 5))
84
85     def takeAction(self, action):
86         """if action is valid (in allowedActions) modify game to
87         perform move"""

```

```

82         if action in self.allowedActions:
83             self.actions.append(action)
84             self.board[action] = self.player
85             self.isDone = self.getIsDone()
86             self.player = -self.player
87             self.getAllowedActions()
88
89
90     def consoleRender(self):
91         "render state to Console"
92         for y in range(6):
93             for x in range(7):
94                 if self.encodeAction(x, y) in self.
allowedActions:
95                     print("+ ", end="")
96                 else:
97                     print(self.pieces[self.board[x+y*7]], end=" ")
98
99             print("")
100         print("")
101         print(0,1,2,3,4,5,6)
102         print(f"player {self.pieces[self.player]} is up")
103
104     def toServerProtocol(self):
105         "convert to binary int array to send to server"
106         out = [0] * 85
107         out[-1] = self.player
108         for idx, val in enumerate(self.board):
109             if val == 1:
110                 out[idx] = 1
111             elif val == -1:
112                 out[idx + len(self.board)] = 1
113         return out
114
115     def getIsDone(self):
116         "check if game is done"
117         if self.board.count(0) == 0:
118             self.tie = True
119             return True
120
121         done = False
122         for option in winStates:
123             val = 0
124             for pos in option:

```

```
125         val += self.board[pos]
126
127     if val == 4*self.player:
128         done = True
129         self.ends.append((option[0], option[-1]))
130
131     return done
```

### 5.2.3.3 gameSaver.py

```
1 import socket
2 import pickle
3
4 class DummyClient():
5     @staticmethod
6     def intToBinArr(my_int):
7         """converts a number to 4 byte binary to send to server"""
8         out = bytearray()
9         for e in [my_int >> i & 0xff for i in (0,8,16,24)]:
10             out.append(e)
11         return out
12
13 def connect():
14     """connect to server"""
15     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16     sock.connect((wget("https://wandhoven.ddns.net/code/AlphaZero/connect4ServerIP.txt").content, 2551))
17     return sock
18
19 def send(data):
20     """send data to server"""
21     sock = connect()
22
23     sock.send((-2).to_bytes(4, "little", signed=True))
24
25     b = pickle.dumps(data)
26     sock.send(len(b).to_bytes(4, "little", signed=True))
27     sock.send(b)
28
29     size = int.from_bytes(sock.recv(4), "little", signed=True)
30     data = pickle.loads(sock.recv(size))
31     return data
32
33 def getClientWithClosestElo(account):
34     """get the agent version with the closest elo to the one of the user"""
35     myElo = send(("accounts", ["elos", str(account)]))
36     sock = connect()
37
38     sock.send(int(-1).to_bytes(4, "little", signed=True))
39     sock.send(int(myElo).to_bytes(4, "little", signed=True))
40
41     other = int.from_bytes(sock.recv(4), "little", signed=True)
```

```

42     return other
43
44 def getMyElo():
45     """get the users elo from the server"""
46     account = getAccount()
47     myElo = send(("accounts", ["elos", str(account)]))
48     if myElo < 100:
49         myElo = 100
50     return myElo
51
52 def setMyElo(elo):
53     """set the players elo on the server"""
54     account = getAccount()
55     myElo = send(("accounts", ["elos", str(account)], elo))
56     if myElo < 100:
57         myElo = 100
58     return myElo
59
60 def getElo(account):
61     """get the elo of a certain account (usually used for the ai
62     elos)"""
63     sock = connect()
64
65     sock.send(int(1).to_bytes(4, "little", signed=True))
66     sock.send(int(account).to_bytes(4, "little", signed=True))
67
68     other = int.from_bytes(sock.recv(4), "little", signed=True)
69     return other
70
71 def reset():
72     send(("accounts", ["accountId", -10]))
73     send(("connect4", ["game_counter", "tie"], 0))
74     send(("connect4", ["game_counter", "lose"], 0))
75     send(("connect4", ["game_counter", "win"], 0))
76
77 def sendFull(data, win):
78     """send full game log to server"""
79     if win == 1:
80         typ = "win"
81
82     elif win == -1:
83         typ = "lose"
84
85     elif win == 0:

```



```

86         typ = "tie"
87
88     identity = send(("connect4", ["game_counter", typ]))
89     send(("connect4", ["games", typ, str(hex(identity))], data))
90     print("id: ", (typ, identity))
91     send(("connect4", ["game_counter", typ], identity+1))
92
93
94 def getAccount():
95     """get the account id from file or if not available create
96     account and save to file"""
97     try:
98         with open("account.p", "rb") as file:
99             return pickle.load(file)
100     except:
101         print("creating account file")
102         account = send(("accounts", ["accountId"]))
103         send(("accounts", ["accountId", account - 1]))
104         send(("accounts", ["elos", str(account)], 100))
105         with open("account.p", "wb") as file:
106             pickle.dump(account, file)
107         return account

```

#### 5.2.3.4 GUI.py

```
1 import tkinter as tk
2 from PIL import Image, ImageTk
3 from game import Game
4 from threading import Thread
5 import time
6 from gameSaver import sendFull
7 from Client import DummyAgent
8
9 class ConsoleAgent:
10     """Agent running in the console for testing only"""
11     def render(self, state):
12         """render the state to the console"""
13         state.consoleRender()
14
15     def getAction(self, state):
16         """get the Action the player wants to perform function
17         will be called until valid output is found"""
18         return state.actionModifier(int(input("your Action: ")))
19
20     def winScreen(self, state):
21         """dummy for now"""
22         pass
23
24 class GUI(tk.Tk, DummyAgent):
25     """
26     render game to GUI using Tkinter and canvas
27     """
28     colorMap = {
29         1: "gold",
30         -1: "red",
31         0: "white"
32     }
33     yPadRel = 0.1
34     _canvasy = 450
35     _canvasx = 500
36     _dotSize = 0.45
37     _lastState = None
38     _win = 0
39     _winLinesRendered = False
40     winLines_kwargs = {
41         "fill": "#00FF00",
42         "width": 10
```

```

43     }
44     def __init__(self, state, game, replayer):
45         super(GUI, self).__init__()
46         self.replayer = replayer
47         self.title("Connect4 AlphaZero Client")
48         self.geometry("500x500")
49         self.bind("<Configure> ", self._resize)
50
51         self.yPad = 60
52         self.action = -1
53
54         self.canvas = tk.Canvas(self, height=self._canvasy,
width=self._canvasx, bg="#FFFFFF")
55         self.canvas.bind("<Button-1>", self._writeAction)
56         self.canvas.place(x=0, y=self.yPad)
57
58         self.playerLabel = tk.Label(self, text="testText", font=(
"Arial", self.yPad//2))
59         self.playerLabel.place(x=0, y=0)
60
61         self._drawBoard()
62         self._drawStones(state)
63
64         self.game = game
65
66     def _resize(self, event):
67         """callback for resizing of the window"""
68         if event.widget == self:
69             self.yPad = int(self.yPadRel * event.width)
70             self.canvas.place(x=0, y=self.yPad)
71             self.playerLabel.config(font=("Arial", self.yPad//2))
72
73
74             self._canvasy = event.height - self.yPad
75             self._canvasx = event.width
76
77             self.canvas.config(height=self._canvasy, width=self.
_canvasx)
78             self.render(self._lastState)
79
80     def _getDxDy(self):
81         """get the dx and dy nedded internaly to compute field and
stone sizes"""
82         return self._canvasx / 8, self._canvasy / 7

```

```

83
84     def render(self, state):
85         "render the state"
86         self._drawBoard()
87         if not state is None:
88             self._lastState = state
89             self._drawStones(state.board)
90
91             if state.player == 1:
92                 self.playerLabel.config(text = "Yellow 's Turn",
93 fg="#808080")
94             else:
95                 self.playerLabel.config(text = "Red 's Turn", fg=
96 "#808080")
97
98             self.renderWinLines(state)
99
100         if not self._lastState is None:
101             if self._lastState.isDone:
102                 self._renderEndMsg()
103
104     def _drawBoard(self):
105         "render 7x6 board using lines"
106         self.canvas.delete("all")
107
108         dx, dy = self._getDxDy()
109         offset = 0.5
110         for x in range(8):
111             self.canvas.create_line(dx*(x+offset), dy*offset, dx*(
112 x+offset), self._canvasy - dy*offset)
113
114         for y in range(7):
115             self.canvas.create_line(dx*offset, dy*(y+offset), self
116 ._canvasx - dx*offset, dy*(y+offset))
117
118     def _drawStones(self, state):
119         "place stones in board"
120         dx, dy = self._getDxDy()
121
122         for x in range(1, 8):
123             for y in range(1, 7):
124                 if state[Game.encodeAction(x-1, y-1)] != 0:
125                     Xpos = dx * x
126                     Ypos = dy * y

```

```

124         Ysize= self._dotSize * dy
125         Xsize= self._dotSize * dx
126
127         color = self.colorMap[state[Game.
encodeAction(x-1, y-1)]]
128
129         self.canvas.create_oval(
130             Xpos - Xsize, Ypos-Ysize,
131             Xpos+Xsize, Ypos+Ysize,
132             fill=color, width=0
133         )
134
135     def _renderEndMsg(self):
136         "render the message at the end of the game"
137         args = (self._canvasx//2, self._canvasy//2)
138         fontSize = min(self._canvasx//10, self._canvasy//2)
139         kwargs = {
140             "font": f"Times {fontSize} bold",
141             "anchor": "c",
142         }
143         if self.replayer is None:
144             if self._win == 1:
145                 txt = self.canvas.create_text(*args, **kwargs,
fill="green",
146                                             text="You Win")
147                 sendFull(self.game.actions, -1)
148
149             elif self._win == -1:
150                 txt = self.canvas.create_text(*args, **kwargs,
fill="black", text="You Loose")
151                 sendFull(self.game.actions, 1)
152
153             elif self._win == 0:
154                 txt = self.canvas.create_text(*args, **kwargs,
fill="black", text="Tie")
155                 sendFull(self.game.actions, 0)
156
157     def _writeAction(self, event):
158         """
159         callback from canvas mouse left click.
160         Converts postion to grid position and than to action
161         witch is saved.
162         """
163         dx, dy = self._getDxDy()

```

```

164
165     XPos = (event.x - dx * 0.5) // dx
166     YPos = (event.y - dy * 0.5) // dy
167
168     self.action = int(XPos + 7*YPos)
169
170     def getAction(self, state):
171         """Make playerLabel black and wait for an action to be
172         written."""
173         self.playerLabel.config(fg="#000000")
174         self.action = -1
175         while self.action == -1:
176             time.sleep(0.1)
177
178         if self.replayer is None:
179             return self.action
180         else:
181             return self.replayer.getAction(state)
182
183     def drawLineOverTime(self, x1, y1, x2, y2, steps, dt, args
184     =(), **kwargs):
185         """draw a line from (x1, y1) to (x2, y2) over time"""
186         line = self.canvas.create_line(x1, y1, x1, y1, *args, **
187         kwargs)
188         dx = (x2 - x1) / steps
189         dy = (y2 - y1) / steps
190         for idx in range(steps+1):
191             time.sleep(dt)
192             self.canvas.delete(line)
193             line = self.canvas.create_line(x1, y1, x1+dx*idx, y1
194             +dy*idx, *args, **kwargs)
195
196     def getPos(self, pos):
197         """get action to canvas postion"""
198         a, b = Game.decodeAction(pos)
199         dx, dy = self._getDxDy()
200         return (a+1)*dx, (b+1)*dy
201
202     def winScreen(self, game, _win):
203         """show win screen"""
204         self._win = 2
205         self.render(game)
206         self._winLinesRendered = False
207
208         dx, dy = self._getDxDy()

```

```

205     threads = []
206     if not game is None:
207         for a, b in game.ends:
208             x1, y1 = self.getPos(a)
209             x2, y2 = self.getPos(b)
210             currentThread = Thread(
211                 target=self.drawLineOverTime,
212                 args=(
213                     x1,y1,
214                     x2,y2,
215                     20,0.01
216                 ),
217                 kwargs = self.winLines_kwargs
218             )
219             currentThread.start()
220             threads.append(currentThread)
221
222         for thread in threads:
223             thread.join()
224         del threads
225
226     self._win = _win
227     if game.tie:
228         self._win = 0
229
230     self._winLinesRendered = True
231
232     def renderWinLines(self, game):
233         """render the lines that caused the win"""
234         if self._winLinesRendered:
235             if game.isDone:
236                 for a, b in game.ends:
237                     x1, y1 = self.getPos(a)
238                     x2, y2 = self.getPos(b)
239                     self.canvas.create_line(x1,y1,x2,y2, **self.
winLines_kwargs)

```

### 5.2.3.5 main.py

```
1 from game import Game
2 from Client import RemoteClient, GameReplayAgent
3 from GUI import GUI, ConsoleAgent
4 from threading import Thread
5 from random import seed, getrandbits
6 from time import time, sleep
7 import pickle
8 from tkinter import simpledialog
9 from requests import get as wget
10 import gameSaver
11
12
13 def render(agents, game):
14     "render the state for all agents"
15     for agent in agents.values():
16         agent.render(game)
17
18 def endScreens(agents, game):
19     "render end screen for all agents"
20     for player, agent in agents.items():
21         agent.winScreen(game, -player*game.player*game.
22             isDone)
23
24 def run(game, agent1, agent2, gui):
25     "call the agents, render and get action to play a game"
26     sleep(0.5)
27     while True:
28         agents, winOffsetter = getAgents(gui, agent1, agent2)
29         game.reset()
30         while not game.isDone:
31             render(agents, game)
32             action = agents[game.player].getAction(game)
33             game.takeAction(action)
34
35         endScreens(agents, game)
36         render(agents, game)
37
38         if game.tie:
39             eloWin = 0.5
40         else:
41             eloWin = game.player * winOffsetter
42
43         agent1.updateElo(-eloWin)
```



```

43         agent2.updateElo( eloWin)
44
45         sleep(5)
46
47 def getAgents(agent1, agent2, game):
48     """get dict mapping player actions id's to agents"""
49     val = getrandbits(1)*2-1
50     out = {
51         +val: agent1,
52         -val: agent2
53     }
54     return out, val
55
56 if __name__ == "__main__":
57     seed(time())
58     doReplay = False
59     replayer = GameReplayAgent("win", 3, "connect4")
60     while True:
61         ip = wget("https://wandhoven.ddns.net/code/AlphaZero/
connect4ServerIP.txt").content
62         game = Game()
63         gui = GUI(game.board, game, replayer if doReplay else
None)
64         client = gui if doReplay else RemoteClient(ip, Game.
port)
65
66         runner = Thread(target=run, args=(game, client, gui, gui
))
67         runner.start()
68         gui.mainloop()

```

#### 5.2.3.6 test.py

```
1 """test file"""
2
3 import tkinter as tk
4 from tkinter import simpledialog
5
6 ROOT = tk.Tk()
7
8 ROOT.withdraw()
9 # the input dialog
10 USER_INP = simpledialog.askstring(title="Test",
11                                   prompt="What's your Name?: ")
12
13 # check it out
14 print("Hello", USER_INP)
```

### 5.2.3.7 winStates.json

```
1  [  
2    [0,1,2,3] ,  
3    [1,2,3,4] ,  
4    [2,3,4,5] ,  
5    [3,4,5,6] ,  
6  
7    [7,8,9,10] ,  
8    [8,9,10,11] ,  
9    [9,10,11,12] ,  
10   [10,11,12,13] ,  
11  
12   [14,15,16,17] ,  
13   [15,16,17,18] ,  
14   [16,17,18,19] ,  
15   [17,18,19,20] ,  
16  
17   [21,22,23,24] ,  
18   [22,23,24,25] ,  
19   [23,24,25,26] ,  
20   [24,25,26,27] ,  
21  
22   [28,29,30,31] ,  
23   [29,30,31,32] ,  
24   [30,31,32,33] ,  
25   [31,32,33,34] ,  
26  
27   [35,36,37,38] ,  
28   [36,37,38,39] ,  
29   [37,38,39,40] ,  
30   [38,39,40,41] ,  
31  
32   [0 , 7,14,21] ,  
33   [7 ,14,21,28] ,  
34   [14,21,28,35] ,  
35  
36   [1 ,8 ,15,22] ,  
37   [8 ,15,22,29] ,  
38   [15,22,29,36] ,  
39  
40   [2,9,16,23] ,  
41   [9,16,23,30] ,  
42   [16,23,30,37] ,  
43
```

```

44 [3 ,10 ,17 ,24] ,
45 [10 ,17 ,24 ,31] ,
46 [17 ,24 ,31 ,38] ,
47
48 [4 ,11 ,18 ,25] ,
49 [11 ,18 ,25 ,32] ,
50 [18 ,25 ,32 ,39] ,
51
52 [5 ,12 ,19 ,26] ,
53 [12 ,19 ,26 ,33] ,
54 [19 ,26 ,33 ,40] ,
55
56 [6 ,13 ,20 ,27] ,
57 [13 ,20 ,27 ,34] ,
58 [20 ,27 ,34 ,41] ,
59
60
61 [14 ,22 ,30 ,38] ,
62
63 [7 , 15 ,23 ,31] ,
64 [15 ,23 ,31 ,39] ,
65
66 [0 , 8 , 16 ,24] ,
67 [8 , 16 ,24 ,32] ,
68 [16 ,24 ,32 ,40] ,
69
70 [1 , 9 ,17 ,25] ,
71 [9 , 17 ,25 ,33] ,
72 [17 ,25 ,33 ,41] ,
73
74 [2 , 10 ,18 ,26] ,
75 [10 ,18 ,26 ,34] ,
76
77 [3 , 11 ,19 ,27] ,
78
79
80 [3 , 9 , 15 ,21] ,
81
82 [4 , 10 ,16 ,22] ,
83 [10 ,16 ,22 ,28] ,
84
85 [5 , 11 ,17 ,23] ,
86 [11 ,17 ,23 ,29] ,
87 [17 ,23 ,29 ,35] ,
88

```

```
89 [6, 12,18,24],
90 [12,18,24,30],
91 [18,24,30,36],
92
93 [13,19,25,31],
94 [19,25,31,37],
95 [20,26,32,38]
96 ]
```

## 5.3 elo

### 5.3.1 agent.py

```
1 import score
2 import math
3
4 def updateScore(Ra, K, Sa, Ea):
5     """elo update function (equation 20)"""
6     return Ra + K*(Sa - Ea)
7
8 class Agent:
9     """class handling elo computations server side"""
10    def __init__(self, elo):
11        self.elo = elo
12        self.expectedScore = score.PredictedScores()
13        self.realScore = score.Score()
14
15    def addGamePrediction(self, other):
16        """update elo prediction score based on the other player"""
17        self.expectedScore.addGame(self, other)
18
19    def update(self, k):
20        self.elo = updateScore(self.elo, k, self.realScore.score
21        , self.expectedScore.score)
22        self.realScore.score = 0
23        self.expectedScore.score = 0
24
25
26 def addGame(agent1, agent2, win):
27     """add a game to agent1 (agent2 is a relic)"""
28     if win == 1:
29         agent1.realScore.addWin()
30     elif win == 0:
31         agent1.realScore.addTie()
32     else:
33         agent1.realScore.addLoss()
34
35
36 getElo = score.getElo #eh
37
38 def getPredictedElo(agent1, agent2):
39     """compute elo predictions of both plays"""
```

```

40     elo = {}
41     elo[agent1] = getElo(agent1, agent2)
42     elo[agent2] = getElo(agent2, agent1)
43     return elo
44
45 def update(agent1, score1, agent2, score2, k):
46     """update elo prediction of both players"""
47     agent1.elo = updateScore(agent1.elo, k, getElo(agent1,
48     agent2), score1)
49     agent2.elo = updateScore(agent2.elo, k, getElo(agent2,
    agent1), score2)

```

### 5.3.2 renderElo.py

```
1 """generate the graphs seen in the paper."""
2
3
4 import json
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 deltaElo = 105
9
10 with open("elos.json", "r") as file:
11     vals = json.load(file)
12
13 elos = np.array([x for x in vals.values()])
14 expected = [100 + deltaElo * int(idx) for idx in vals.keys()]
15
16 x = np.array([int(x) for x in vals.keys()])
17 regressionPoints = np.vstack([x, np.ones(len(vals))]).T
18 m, c = np.linalg.lstsq(regressionPoints, elos, rcond=None)[0]
19 print(m)
20
21 fig = plt.figure()
22 ax1 = fig.add_subplot(111)
23 ax1.set_ylabel('Elo-raiting')
24 ax1.set_xlabel("neural network version")
25 ax1.set_title('Raiting by version')
26
27 ax1.plot(elos, lw=2, label="true Raiting")
28 ax1.plot(expected, lw=2, label="expected Raiting")
29 plt.plot(x, m*x + c, 'r', label='fitted Raiting')
30
31 plt.legend()
32
33 plt.subplots_adjust(left=0.17)
34 plt.show()
```



### 5.3.3 score.py

```
1 import math
2
3 def getElo(agent1, agent2):
4     """compute expected win probability based on rating of both
5     agents."""
6     return 1/(1+math.e**((agent2.elo - agent1.elo)/400))
7
8 class Score:
9     """keep track of actual the score to than later update elo
10    with."""
11
12    def __init__(self):
13        self.score = 0
14
15    def addWin(self):
16        self.score += 1
17
18    def addTie(self):
19        self.score += 0.5
20
21    def addLoss(self):
22        pass
23
24 class PredictedScores:
25     """keep track of the expected score to update elo with later
26     ."""
27
28    def __init__(self):
29        self.score = 0
30
31    def addGame(self, this, other):
32        self.score += getElo(this, other)
```

### 5.3.4 server.py

```
1  ## run the elo server and handle all requests
2
3  import socket
4  import agent
5  import json
6  import pickle
7  from os.path import join as joinPath
8
9  PATH = "/media/A/MyCode/AlphaZero/elo"
10 print(PATH)
11
12 class Server:
13     def __init__(self):
14         """load data, create the server sockte and start
15         listening"""
16         self.load()
17         self.serverSock = socket.socket(socket.AF_INET, socket.
18 SOCK.STREAM)
19         self.serverSock.bind((" ", 2551))
20         self.serverSock.listen(5)
21         self.main()
22
23     def main(self):
24         """decode all incomming requests."""
25         while True:
26             print("waiting for connection")
27             sock = self.serverSock.accept()[0]
28             data = Server.getData(sock)
29             if (data[0] == 1):
30                 #handle elo related requests
31                 self.update_elo(data[1], sock)
32             elif (data[0] == 2):
33                 #handle generall data related requests.
34                 out = pickle.dumps(self.update_data(data[1]))
35
36                 sock.send(len(out).to_bytes(4, "little", signed=
37 True))
38                 sock.send(out)
39
40     def update_data(self, data):
41         """open the correct file, find the key, specified in the
42         request and update or change the apropiate data."""
43         try:
```

```

40         with open(joinPath(PATH, "data", f"{data[0]}.json"),
41                    "r") as file:
42             info = json.load(file)
43         except:
44             info = {}
45
46         sub = info
47         for key in data[1][: -1]:
48             try:
49                 sub = sub[key]
50             except KeyError:
51                 sub[key] = {}
52                 sub = sub[key]
53
54         if len(data) == 3:
55             sub[data[1][ -1]] = data[2]
56
57         with open(joinPath(PATH, "data", f"{data[0]}.json"),
58                    "w") as file:
59             json.dump(info, file, sort_keys=True, indent=2)
60             return True
61         else:
62             try:
63                 return sub[data[1][ -1]]
64             except KeyError:
65                 return None
66
67     def getAgent(self, key):
68         """get the agent asociated with a certain elo key."""
69         if key == -1:
70             return agent.Agent(100)
71
72         if not key in self.agents:
73             self.agents[key] = agent.Agent(100)
74
75         return self.agents[key]
76
77     @staticmethod
78     def getData(sock):
79         """read data from socket"""
80         size = int.from_bytes(sock.recv(4), "little", signed=
81                                True)
82         data = []
83         if size == -1:
84             data.append(-1)

```

```

82         data.append(int.from_bytes(sock.recv(4), "little",
signed=True))
83         return (1, data)
84
85     if size == -2:
86         size = int.from_bytes(sock.recv(4), "little", signed
=True)
87         data = pickle.loads(sock.recv(size))
88         return (2, data)
89
90     for i in range(size):
91         data.append(int.from_bytes(sock.recv(4), "little",
signed=True))
92         return (1, data)
93
94     def update_elo(self, data, sock):
95         """change, get the elo of a certain agent or get the
agent with the closest elo. (process elo requests)"""
96         deltaElo = 0
97
98         if data[0] == -1:
99             closest = None
100             idx = list(self.agents.keys())[0]
101             for _idx, agent in self.agents.items():
102                 if (_idx > 0):
103                     if closest is None:
104                         if data[1] < agent.elo:
105                             idx = _idx
106                             closest = agent
107
108                     elif (abs(data[1] - closest.elo) > abs(data
[1] - agent.elo) and data[1] < agent.elo):
109                         idx = _idx
110                         closest = agent
111             deltaElo = idx
112
113         else:
114             agent1 = self.getAgent(data[0])
115             currentElo = agent1.elo
116
117             if len(data) == 3:
118                 agent2 = self.getAgent(data[1])
119                 agent1.addGamePrediction(agent2)
120                 agent.addGame(agent1, agent2, data[2])
121                 agent1.update(32)

```

```

122         deltaElo = abs(agent1.elo - currentElo)
123
124     elif len(data) == 2:
125         agent1.elo = data[1]
126         deltaElo = abs(agent1.elo - currentElo)
127
128     elif len(data) == 1:
129         deltaElo = agent1.elo
130
131     sock.send(int(deltaElo).to_bytes(4, "little", signed=
132 True))
133     sock.close()
134     self.save()
135
136 def load(self):
137     """load elo ratings from file"""
138     self.agents = {}
139     try:
140         with open(joinPath(PATH, "elos.json"), "r") as file:
141             tmp = json.load(file)
142
143             for key, elo in tmp.items():
144                 self.agents[int(key)] = agent.Agent(elo)
145     except Exception as e:
146         print(e)
147
148 def save(self):
149     """save elo ratings to file"""
150     d = {}
151     for key, agent in self.agents.items():
152         d[key] = agent.elo
153
154     with open(joinPath(PATH, "elos.json"), "w") as file:
155         json.dump(d, file, sort_keys=True, indent=2)
156
157 if __name__ == "__main__":
158     while True:
159         try:
160             server = Server()
161         except: pass

```

## 5.4 game

### 5.4.1 connect4

#### 5.4.1.1 config.hpp

```
1 // configure AlphaZero global variables
2
3 #pragma once
4 #include <log.hpp>
5 #include <bitset>
6 #include <mutex>
7
8 // check if running on unix (unix has the gpu in my runs)
9 #ifdef unix
10 #define UNIX
11 #endif
12
13 // set the model evaluation device (cuda or cpu)
14 #ifdef UNIX
15 #define DEVICES "cuda:0"
16 #endif
17 #ifndef UNIX
18 #define DEVICES "cpu"
19 #endif
20
21 #define OPSMode 1 // used to define what server is used
22 /*
23 |-----|-----|
24 | OPSMode | Description |
25 |-----|-----|
26 | 1       | Run Server   |
27 |-----|-----|
28 | 2       | Run Tester   |
29 |-----|-----|
30 */
31
32 extern std::mutex console_mutex; // mustex used to prevent
    multiple console writes at once
33
34
35 #define stateSize 84 // full size of the board
    bitarray
36
37 #define U_computation(edge) (this->cpuct * edge.P * std::sqrt((
```

```

float)Nb) / (float)(1 + edge.N))
38                                     // comparison function for edges
    explained in paper
39
40
41 // runn setting
42 #define runVersion 1                // identity of the run
43 #define loadVersion -1             // model version to load
44
45 // Net settings
46 #define learningRage 0.1            // model learning rate
47 #define Momentum 0.9               // model momentum
48
49 // simulation setting
50 #define MCTSSimulations 50          // number of MCTS
    simulations for every action
51 #define cpuct_ 1.0f                // explorational constant used
    for U_computations
52 #define ProbabiliticMoves 10        // moves after start for
    witch to choose action probabilistically
53 // #define Alpha 0.9                // depricated
54
55 // memory setting
56 #define memory_size 30000           // amount of states in
    memory befor it starts training.
57
58 // self play
59 #define EPOCHS 1                    // number of generational games
    to be played per thread
60 #define GEN_THREADS 60              // number of parallel
    generation threads
61 #define probabilistic_moves 10      // how manny moves are
    prabilistic in the begining of the game to aid in exploration
62
63 // training
64 #define Training_loops 20           // amount of batches that
    are generated and the model trained for per training
65 #define Training_batch 256          // number of states per
    batch
66 #define Training_epochs 5           // training epochs
67
68 // turney
69 #define Turnement_probabilisticMoves 2 // same as
    ProbabiliticMoves but for the model evaluation phase

```

```

70 #define TurneyEpochs 1          // same as EPOCHS but for
    the model evaluation phase
71 #define TurneyThreads 20         // same as GEN_THREADS but
    for the model evaluation phase
72 #define scoringThreshold 1.3     // (wins currently best
    player) * scoringThreshold < (new player wins)
73
74 // console
75 #define RenderTrainingProgress false // whether or not the
    trainin progress is rendered to console
76 #define RenderGenAndTurneyProgress false // whether or not
    the self play should be rendered to console
77
78 // Saving
79 #define SaverType 0              // what type of game loggs
    should be created
80 /*
    +-----+-----+
81 | SaverType | Description |
    |
82 +-----+-----+
83 | 0         | no Saver    |
    |
84 +-----+-----+
85 | 1         | save full state to file
    |
86 +-----+-----+
87 | 2         | Save taken Actions to file (int size is saved
    size of the int in bytes) |
88 +-----+-----+
    */
89 #define SaverIntSize 1           // the size of an intager
    for the saver (1,2,3,4)
90
91
92 typedef std::bitset<stateSize> IDType; // the type of the
    game board.

```



### 5.4.1.2 game.hpp

```
1 #pragma once
2 /*
3 The game of connect for coded with game states. This allows The
4 MCTS to simulate the game without changing it.
5 */
6 #include <iostream>
7 #include <vector>
8 #include <list>
9 #include <memory>
10 #include <tuple>
11 #include <unordered_map>
12 #include <bitset>
13 #include <unordered_set>
14 #include <torch/torch.h>
15
16 #include "config.hpp"
17
18 // input_shape is the shape of the game board. x, y is the shape
19 // of the actual game board and z is number of stacked planes (
20 // in this case one for each
21 // player).
22 #define input_shape_x 7
23 #define input_shape_y 6
24 #define input_shape_z 2
25 #define action_count 42
26 #define action_shape 6, 7
27 #define boardOffset 42 // the size of a layer of the board in the
28 // buffer. (the amount of fields)(x * y)
29 // the actual name of the game
30 #define gameName "connect4"
31
32 namespace AlphaZero {
33     namespace Game {
34         // Game State class contains all information of a certain
35         // board position. a board with the positions of all pieces
36         // along with the current player. It also computes legal
37         // Actions A and win information
38         class GameState {
39             // the current player 1 or -1
40             public: int player;
```

```

38
39 // true if game is done (4 in a row or all filled) and false
    if not.
40 public: bool done;
41
42 // winning information who won and by how much. the tuple
    contains the following information in this order <current
    player win (1) or
43 // current player loose (-1) or tie (0), current player
    points (1 for win -1 for loose and 0 for tie), other player
    points (- current // player points
44 public: std::tuple<int, int, int> val;
45
46 // 84 bit bitmap that contains the current board shape. The
    first 42 bits are the positions of the player ones stones (0
    for empty, 1 for
47 // stone present) and the second 42 bits are the same for
    player -1. The 6x7 board is encoded by placing the 6 rows
    next to each other
48 // starting from the front so the top left would be bit 0 and
    the bottom right bit 41.
49 public: IDType gameBoard;
50
51 // list of allowed actions. every Action is the index of the
    bit where the stone would be placed.
52 public: std::vector<int> allowedActions;
53
54
55 // construct from known game state and player.
56 public: GameState(IDType board, int _player);
57
58 // construct using default state.
59 public: GameState();
60
61 // utilit function called by the constructors to avoid
    duplicate code. (all the initialization done by both of the
    constructors)
62 private: void initialize(IDType board, int _player);
63
64 // simulate an action from the current game state. (compute
    the state you would reach from this state by taking the
    following action)
65 public: std::shared_ptr<GameState> takeAction(int action);
66
67 // check whether the game is done. (will set the done

```

```

boolean and the val tuple)
68 public: void gameIsDone();
69
70 // compute all allowed actions and writes it to the allowed
actions list.
71 protected: void getAllowedActions();
72
73 // function that will return 1 if player 1 has a stone at
the position specified by id, -1 if player 2 does and 0 if
nether.
74 public: int IdIndex(int id);
75
76 //returns the id of the game state. In this case the game
board will surface as it contains all information. (unable to
remove stones)
77 public: IDType id();
78
79 // renders the game state to the console in a way that is
readable for humans.
80 public: void render();
81 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
|| ModelLogger)
82 // same as render() but rendes to a specified logger.
83 public: void render(std::shared_ptr<spdlog::logger> logger);
84 #endif
85 // sets the piece at id to 0, 1 or -1 (val). b is the board
that the piece will be set to. (see gameBoard for encoding)
86 public: void static IdIndex(int id, int val, IDType& b);
87
88 // converts the game state to a tensor, that is than passed
thought the model.
89 public: torch::Tensor toTensor();
90
91 // dose tha same thing but is more efficient for stacked
game states during training. It will set tensor[idx] to the
tensor representing
92 // this game state.
93 public: void toTensor(torch::Tensor& tensor, unsigned short
idx=0);
94
95 // the character representing the pieca at the position val.
(a + if a stone can be placed there)
96 private: char getPiece(int val);
97
98 // recursive function used to determin the height of a colum

```

```

    to determin where stones can be placed. The returned int is
    a possible
99 // placement position if the bool is true. if the bool is
    false the colum is full.
100 private: std::pair<int, bool> getAllowedColumHeight(int);
101 };
102
103 // hash function used for hash mapes using the board as a
    hash key.
104 struct StateHash
105 {
106     std::size_t operator()(std::pair<std::shared_ptr<GameState>
    >, std::vector<int>>> const& s) const noexcept;
107 };
108 // returns all identidal game states and action maps (N) to
    the passed one.
109 std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
    GameState>, std::vector<int>>> identities(std::shared_ptr<
    GameState> state, std::vector<int>& actionProbs);
110
111
112 // The Actual game contains and handles the gamestates for
    the current generation Game.
113 class Game {
114
115     // pointer to the current game state.
116     public: std::shared_ptr<GameState> state;
117
118
119     // constructor will initialize a game state.
120     public: Game();
121
122     // reset game to initial position
123     public: void reset();
124
125     // take an action as defined by the game state
126     public: void takeAction(int action);
127
128     // human action (action is the colum that the stone should
    be placed in, than the action is determined)
129     public: bool takeHumanAction(int action);
130
131     // call the states render function.
132     public: void render();
133 };

```

```

134
135 // Test the game.
136 inline void test() {
137     AlphaZero::Game::Game* game = new AlphaZero::Game::Game();
138
139     while (!game->state->done) {
140         std::vector<int> vec = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
141                                10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
142                                25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
143                                40, 41 };
144         auto idents = identities(game->state, vec);
145         idents[1].first->render();
146         std::cout << "your action: ";
147         int action;
148         std::cin >> action;
149         game->takeHumanAction(action);
150 #if Windows
151         system("cls");
152 #else
153         system("clear");
154 #endif
155     }
156     game->render();
157
158     std::cout << std::endl << "the last player just won";
159 }
160
161 inline std::size_t AlphaZero::Game::StateHash::operator()(std::
162     pair<std::shared_ptr<GameState>, std::vector<int>>> const& s)
163     const noexcept {
164     return s.first->gameBoard.to_ulong();
165 }
166
167 inline int AlphaZero::Game::GameState::IdIndex(int id)
168 {
169     if (this->gameBoard[id] == 1) {
170         return 1;
171     }
172     else if (this->gameBoard[id + boardOffset] == 1) {
173         return -1;
174     }
175     return 0;
176 }

```

```

174
175 inline void AlphaZero::Game::GameState::IdIndex(int id, int val,
           IDType& b)
176 {
177     if (val == 0) {
178         b.set(id, 0);
179         b.set(id + boardOffset, 0);
180         return;
181     }
182     if (val == -1) {
183         id += boardOffset;
184     }
185     b.set(id, 1);
186 }
187
188 inline char AlphaZero::Game::GameState::getPiece(int id)
189 {
190     std::unordered_map<int, char> renderData = { {0, '-'}, {1, 'X'}
           }, {-1, 'O'} };
191     if (std::find(this->allowedActions.begin(), this->
           allowedActions.end(), id) != this->allowedActions.end()) {
192         return '+';
193     }
194     auto va = renderData[this->IdIndex(id)];
195     return va;
196 }
197
198 inline IDType AlphaZero::Game::GameState::id()
199 {
200     return this->gameBoard;
201 }
202
203 inline void AlphaZero::Game::Game::render() {
204     this->state->render();
205 }
206
207 inline torch::Tensor AlphaZero::Game::GameState::toTensor()
208 {
209     at::Tensor outTensor = at::zeros({ 1, input_shape_z,
           input_shape_y, input_shape_x });
210     this->toTensor(outTensor);
211     return outTensor;
212 }
213
214 inline void AlphaZero::Game::GameState::toTensor(torch::Tensor&

```

```

        tensor, unsigned short idx)
215 {
216     unsigned short pos = 0;
217     unsigned int offset = (this->player == -1) ? 0 : boardOffset;
218     for (unsigned short z = 0; z < input_shape_z; z++) {
219         for (unsigned short y = 0; y < input_shape_y; y++) {
220             for (unsigned short x = 0; x < input_shape_x; x++) {
221                 tensor[idx][z][y][x] = (float) this->gameBoard[(pos +
offset) % stateSize];
222                 pos++;
223             }
224         }
225     }
226 }

```

### 5.4.1.3 game.cpp

```
1 #include "game.hpp"
2
3 #define columnOffset 7
4
5 AlphaZero::Game::GameState::GameState(IDType board, int _player)
6 {
7     this->initialize(board, _player);
8 }
9
10 AlphaZero::Game::GameState::GameState()
11 {
12     this->initialize(IDType(), 1);
13 }
14
15 void AlphaZero::Game::GameState::initialize(IDType board, int
    _player)
16 {
17     this->gameBoard = board;
18     this->player = _player;
19     this->getAllowedActions();
20     this->gameIsDone();
21 }
22
23 std::shared_ptr<AlphaZero::Game::GameState> AlphaZero::Game::
    GameState::takeAction(int action)
24 {
25     IDType newBoard = this->gameBoard;
26     GameState::IdIndex(action, this->player, newBoard);
27
28     std::shared_ptr<GameState> newState = std::make_shared<
        GameState>(newBoard, -this->player);
29     return newState;
30 }
31
32 void AlphaZero::Game::GameState::gameIsDone()
33 {
34     std::vector<std::vector<int>> winOptions = {
35         /*
36         +-----+
37         | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
38         +-----+
39         | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
40         +-----+
         */
    }
```



```

41 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
42 +---+---+---+---+---+---+---+
43 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
44 +---+---+---+---+---+---+---+
45 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
46 +---+---+---+---+---+---+---+
47 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |
48 +---+---+---+---+---+---+---+
49 */
50 //horizontal
51 {0,1,2,3},
52 {1,2,3,4},
53 {2,3,4,5},
54 {3,4,5,6},
55
56 {7,8,9,10},
57 {8,9,10,11},
58 {9,10,11,12},
59 {10,11,12,13},
60
61 {14,15,16,17},
62 {15,16,17,18},
63 {16,17,18,19},
64 {17,18,19,20},
65
66 {21,22,23,24},
67 {22,23,24,25},
68 {23,24,25,26},
69 {24,25,26,27},
70
71 {28,29,30,31},
72 {29,30,31,32},
73 {30,31,32,33},
74 {31,32,33,34},
75
76 {35,36,37,38},
77 {36,37,38,39},
78 {37,38,39,40},
79 {38,39,40,41},
80 //vertical
81 {0 , 7,14,21},
82 {7 ,14,21,28},
83 {14,21,28,35},
84
85 {1 ,8 ,15,22},

```

```

86     {8 ,15,22,29} ,
87     {15,22,29,36} ,
88
89     {2,9,16,23} ,
90     {9,16,23,30} ,
91     {16,23,30,37} ,
92
93     {3 ,10,17,24} ,
94     {10,17,24,31} ,
95     {17,24,31,38} ,
96
97     {4 ,11,18,25} ,
98     {11,18,25,32} ,
99     {18,25,32,39} ,
100
101     {5 ,12,19,26} ,
102     {12,19,26,33} ,
103     {19,26,33,40} ,
104
105     {6 ,13,20,27} ,
106     {13,20,27,34} ,
107     {20,27,34,41} ,
108
109     //diagonal topleft-bottomRight
110     {14,22,30,38} ,
111
112     {7, 15,23,31} ,
113     {15,23,31,39} ,
114
115     {0, 8, 16,24} ,
116     {8, 16,24,32} ,
117     {16,24,32,40} ,
118
119     {1, 9 ,17,25} ,
120     {9, 17,25,33} ,
121     {17,25,33,41} ,
122
123     {2, 10,18,26} ,
124     {10,18,26,34} ,
125
126     {3, 11,19,27} ,
127
128     //diagonal topright-bottomleft
129     {3, 9, 15,21} ,
130

```

```

131     {4, 10,16,22},
132     {10,16,22,28},
133
134     {5, 11,17,23},
135     {11,17,23,29},
136     {17,23,29,35},
137
138     {6, 12,18,24},
139     {12,18,24,30},
140     {18,24,30,36},
141
142     {13,19,25,31},
143     {19,25,31,37},
144
145     {20,26,32,38},
146 };
147 bool tie = true;
148 for (int idx = 0; idx < action_count; idx++) {
149     if (this->IdIndex(idx) == 0) {
150         tie = false;
151         break;
152     }
153 }
154 if (tie) {
155     this->done = true;
156     this->val = { 0,0,0 };
157     return;
158 }
159 for (auto option : winOptions) {
160     int count = 0;
161     for (int pos : option) {
162         count += this->IdIndex(pos);
163     }
164     if (count == -4 * this->player) {
165         this->done = true;
166         this->val = { -1, -1, 1 }; // winForThisPlayer, points for
167                                     this player, points for other player
168         return;
169     }
170 }
171 this->done = false;
172 this->val = { 0, 0, 0 };
173 }
174 inline std::pair<int, bool> AlphaZero::Game::GameState::

```

```

175     getAllowedColumnHeight(int idx) {
176     if (this->IdIndex(idx) != 0) {
177         return { idx, false };
178     }
179     if (idx >= 35) {
180         return { idx, true };
181     }
182     else if (this->IdIndex(idx + columnOffset) != 0) {
183         return { idx, true };
184     }
185     else {
186         return this->getAllowedColumnHeight(idx + columnOffset);
187     }
188 }
189 void AlphaZero::Game::GameState::getAllowedActions()
190 {
191     this->allowedActions.clear();
192     for (int idx = 0; idx < 7; idx++) {
193         std::pair<int, bool> data = this->getAllowedColumnHeight(idx);
194         ;
195         if (data.second) {
196             this->allowedActions.push_back(data.first);
197         }
198     }
199 }
200 void AlphaZero::Game::GameState::render()
201 {
202     console_mutex.lock();
203     for (int row = 0; row < action_count;) {
204         for (int iter = 0; iter < 7; iter++) {
205             std::cout << this->getPiece(row) << " ";
206             row++;
207         }
208         std::cout << std::endl;
209     }
210     std::cout << std::endl;
211     console_mutex.unlock();
212 }
213
214 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
215     || ModelLogger)
216 void AlphaZero::Game::GameState::render(std::shared_ptr<spdlog::
217     logger> logger)

```

```

216 {
217     for (int idx = 0; idx < 6; idx++) {
218         char line1[13] = {
219             this->getPiece(0 + columnOffset * idx), ' ',
220             this->getPiece(1 + columnOffset * idx), ' ',
221             this->getPiece(2 + columnOffset * idx), ' ',
222             this->getPiece(3 + columnOffset * idx), ' ',
223             this->getPiece(4 + columnOffset * idx), ' ',
224             this->getPiece(5 + columnOffset * idx), ' ',
225             this->getPiece(6 + columnOffset * idx)
226         };
227         logger->info(line1);
228     }
229 }
230 #endif
231
232 AlphaZero::Game::Game::Game()
233 {
234     this->state = std::make_shared<GameState>();
235 }
236
237 void AlphaZero::Game::Game::reset()
238 {
239     this->state = std::make_shared<GameState>();
240 }
241
242 void AlphaZero::Game::Game::takeAction(int action)
243 {
244     auto newState = this->state->takeAction(action);
245     this->state = newState;
246 }
247
248 bool AlphaZero::Game::Game::takeHumanAction(int action)
249 {
250     for (auto const& allowed : this->state->allowedActions) {
251         if ((allowed - action) % 7 == 0) {
252             this->takeAction(allowed);
253             return true;
254         }
255     }
256     return false;
257 }
258
259 inline std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
    std::vector<int>> mirrorGameState(std::shared_ptr<AlphaZero::

```

```

260     Game::GameState> state, std::vector<int>& actionProbs) {
261     IDType boardBuffer;
262     std::vector<int> probs = {
263         actionProbs[6], actionProbs[5], actionProbs[4],
264         actionProbs[3], actionProbs[2], actionProbs[1],
265         actionProbs[0],
266         actionProbs[13], actionProbs[12], actionProbs[11],
267         actionProbs[10], actionProbs[9], actionProbs[8],
268         actionProbs[7],
269         actionProbs[20], actionProbs[19], actionProbs[18],
270         actionProbs[17], actionProbs[16], actionProbs[15],
271         actionProbs[14],
272         actionProbs[27], actionProbs[26], actionProbs[25],
273         actionProbs[24], actionProbs[23], actionProbs[22],
274         actionProbs[21],
275         actionProbs[34], actionProbs[33], actionProbs[32],
276         actionProbs[31], actionProbs[30], actionProbs[29],
277         actionProbs[28],
278         actionProbs[41], actionProbs[40], actionProbs[39],
279         actionProbs[38], actionProbs[37], actionProbs[36],
280         actionProbs[35]
281     };
282     #define assignStateSinge(idx1, idx2) AlphaZero::Game::GameState::
283         IdIndex(idx1, state->IdIndex(idx2), boardBuffer)
284     #define assignState(idx1, idx2) assignStateSinge(idx1, idx2);
285         assignStateSinge(idx2, idx1);
286
287     assignState(0, 6); assignState(1, 5); assignState(2, 4);
288         assignStateSinge(3, 3);
289     assignState(7, 13); assignState(8, 12); assignState(9, 11);
290         assignStateSinge(10, 10);
291     assignState(14, 20); assignState(15, 19); assignState(16, 18);
292         assignStateSinge(17, 17);
293     assignState(21, 27); assignState(22, 26); assignState(23, 25);
294         assignStateSinge(24, 24);
295     assignState(28, 34); assignState(29, 33); assignState(30, 32);
296         assignStateSinge(31, 31);
297     assignState(35, 41); assignState(36, 40); assignState(37, 39);
298         assignStateSinge(38, 38);
299     #undef assignState
300
301     return { std::make_shared<AlphaZero::Game::GameState>(
302         boardBuffer, state->player), probs };

```

```

283 }
284
285 std::vector<std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
    std::vector<int>>>> AlphaZero::Game::identities(std::shared_ptr<GameState> state,
    std::vector<int>& probs)
286 {
287     std::vector<std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
    std::vector<int>>>> idents(2);
288     int id = 0;
289     idents[id] = { state, probs };
290     idents[id] = mirrorGameState(state, probs);
291     return idents;
292 }

```

## 5.4.2 othello

### 5.4.2.1 config.hpp

```
1 #pragma once
2 #include <log.hpp>
3 #include <bitset>
4 #include <mutex>
5
6 #ifdef unix
7 #define UNIX
8 #endif
9
10 #ifdef UNIX
11 #define DEVICES "cuda:0"
12 #endif
13 #ifndef UNIX
14 #define DEVICES "cpu"
15 #endif
16
17 #define OPSMode 1
18
19 extern std::mutex console_mutex;
20 extern std::mutex rand_mutex;
21
22 /*
23 |-----|-----|
24 | OPSMode | Description |
25 |-----|-----|
26 | 1       | Run Server   |
27 |-----|-----|
28 | 2       | Run Tester   |
29 |-----|-----|
30 */
31
32 #define stateSize 128
33 #define Training true
34
35 #define U_computation(edge) (this->cpuct * edge.P * std::sqrt((
    float)Nb) / (float)(1 + edge.N))
36
37
38 // runn setting
39 #define runVersion 1
40 #define loadVersion -1
```



```

41
42 // Net settings
43 #define reg_const 0.0001
44 #define learningRate 0.1
45 #define Momentum 0.9
46
47 // simulation setting
48 #define MCTSSimulations 50
49 #define cpuct_ 1.0f
50 #define ProbabilisticMoves 10
51 #define Alpha 0.9
52 #define EPSILON 0.2f
53
54 // memory setting
55 #define memory_size 30000
56
57 // self play
58 #define EPOCHS 1
59 #define GEN_THREADS 60
60 #define probabilistic_moves 10 // how many moves are probabilistic
    in the beginning of the game to aid in exploration
61
62 // training
63 #define Training_loops 20
64 #define Training_batch 256
65 #define Training_epochs 5
66
67 // turney
68 #define Turnement_probabilisticMoves 2
69 #define TurneyEpochs 1
70 #define TurneyThreads 20
71 #define scoringThreshold 1.3
72
73 // console
74 #define RenderTrainingProgress false
75 #define RenderGenAndTurneyProgress false
76 // #define RenderGameProgress true;
77
78 // Saving
79 #define SaverType 0
80 /*
    +-----+
81 | SaverType | Description |

```

```

82 |-----|
83 | 0          | no Saver
84 |-----|
85 | 1          | save full state to file
86 |-----|
87 | 2          | Save taken Actions to file (int size is saved
   | size of the int in bytes) |
88 |-----|
   */
89 #define SaverIntSize 1
90
91
92 typedef std::bitset<stateSize> IDType;

```

### 5.4.2.2 game.hpp

```
1 #pragma once
2 /*
3  this is the alpha Zero game for Orthello
4  */
5
6 #include <iostream>
7 #include <vector>
8 #include <list>
9 #include <memory>
10 #include <tuple>
11 #include <unordered_map>
12 #include <bitset>
13 #include <unordered_set>
14 #include <torch/torch.h>
15
16 #include "config.hpp"
17
18
19 #define input_shape_x 8
20 #define input_shape_y 8
21 #define input_shape_z 2
22 #define action_count 64
23 #define action_shape 8, 8
24 #define boardOffset 64 // the size of a layer of the board in the
25   buffer. (the amount of fields)
26 #define gameName "Orthello"
27
28 namespace AlphaZero {
29     namespace Game {
30         class GameState {
31         public: int player;
32         public: bool done;
33         public: std::tuple<int, int, int> val;
34         public: IDType gameBoard;
35         public: std::vector<int> allowedActions;
36
37         public: GameState(IDType board, int _player);
38         public: GameState();
39         private: void initialize(IDType board, int _player);
40         public: std::shared_ptr<GameState> takeAction(int action);
41         public: void gameIsDone();
42         protected: void getAllowedActions();
```

```

43     public: int IdIndex(int id);
44     public: IDType id();
45     public: void render();
46 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
    || ModelLogger)
47     public: void render(std::shared_ptr<spdlog::logger> logger);
48 #endif
49     public: void static IdIndex(int id, int val, IDType& b);
50     public: torch::Tensor toTensor();
51     public: void toTensor(torch::Tensor& tensor, unsigned short
idx=0);
52     public: char getPiece(int val);
53     private: std::vector<int> getFlipActions(int x, int y, int
dx, int dy);
54     private: bool hasAdjacentStones(int const& x, int const& y);
55     };
56     struct StateHash
57     {
58         std::size_t operator()(std::pair<std::shared_ptr<GameState
>, std::vector<int>>> const& s) const noexcept;
59     };
60     // optimization function its not a problem if not all are
found
61     std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
GameState>, std::vector<int>>> identities(std::shared_ptr<
GameState> state, std::vector<int>& actionProbs);
62
63     class Game {
64     public: std::shared_ptr<GameState> state;
65
66     public: Game();
67     public: void reset();
68     public: void takeAction(int action);
69     public: bool takeHumanAction(int action);
70     public: void render();
71     };
72
73     inline void test() {
74         AlphaZero::Game::Game game;
75
76         while (!game.state->done) {
77             auto idx = std::rand() % game.state->allowedActions.size
();
78             auto action = game.state->allowedActions[idx];
79             game.render();

```

```

80     game.takeAction(action);
81     char arr[100];
82
83     //system("cls");
84 }
85 game.render();
86
87     std::cout << std::endl << "the last player just won";
88 }
89 std::pair<int, int> to2dPos(int pos);
90 int from2dPos(int const& x, int const& y);
91
92 void renderStates(std::vector<GameState*> states);
93 }
94 }
95
96 inline std::size_t AlphaZero::Game::StateHash::operator()(std::
    pair<std::shared_ptr<GameState>, std::vector<int>>> const& s)
    const noexcept {
97     return s.first->gameBoard.to_ulong();
98 }
99
100 inline int AlphaZero::Game::GameState::IdIndex(int id)
101 {
102     if (this->gameBoard[id] == 1) {
103         return 1;
104     }
105     else if (this->gameBoard[id + boardOffset] == 1) {
106         return -1;
107     }
108     return 0;
109 }
110
111 inline void AlphaZero::Game::GameState::IdIndex(int id, int val,
    IDType& b)
112 {
113     int otherId;
114     if (val == 0) {
115         b.set(id, 0);
116         b.set(id + boardOffset, 0);
117         return;
118     }
119     if (val == 1)
120     {
121         otherId = id + boardOffset;

```

```

122     }
123     if (val == -1) {
124         otherId = id;
125         id += boardOffset;
126     }
127     b.set(id, 1);
128     b.set(otherId, 0);
129 }
130
131 inline char AlphaZero::Game::GameState::getPiece(int id)
132 {
133     std::unordered_map<int, char> renderData = { {0, '-'}, {1, 'X'}
134         }, {-1, 'O'} };
135     if (std::find(this->allowedActions.begin(), this->
136         allowedActions.end(), id) != this->allowedActions.end()) {
137         return '+';
138     }
139     auto va = renderData[this->IdIndex(id)];
140     return va;
141 }
142
143 inline IDType AlphaZero::Game::GameState::id()
144 {
145     return this->gameBoard;
146 }
147
148 inline void AlphaZero::Game::Game::render() {
149     this->state->render();
150 }
151
152 inline torch::Tensor AlphaZero::Game::GameState::toTensor()
153 {
154     at::Tensor outTensor = at::zeros({ 1, input_shape_z,
155         input_shape_y, input_shape_x });
156     this->toTensor(outTensor);
157     return outTensor;
158 }
159
160 inline void AlphaZero::Game::GameState::toTensor(torch::Tensor&
161     tensor, unsigned short idx)
162 {
163     unsigned short pos = 0;
164     unsigned int offset = (this->player == -1) ? 0 : boardOffset;
165     for (unsigned short z = 0; z < input_shape_z; z++) {
166         for (unsigned short y = 0; y < input_shape_y; y++) {

```

```

163         for (unsigned short x = 0; x < input_shape_x; x++) {
164             tensor[idx][z][y][x] = (float) this->gameBoard[(pos +
165                 offset) % stateSize];
166             pos++;
167         }
168     }
169 }
170
171 inline std::pair<int, int> AlphaZero::Game::to2dPos(int pos)
172 {
173     int x = pos % input_shape_x;
174     int y = (pos - x) / input_shape_x;
175     return { x, y };
176 }
177
178 inline int AlphaZero::Game::from2dPos(int const& x, int const& y
179 )
180 {
181     return x + y * input_shape_x;
182 }

```

### 5.4.2.3 game.cpp

```
1 #include "game.hpp"
2
3 #define columnOffset 8
4
5 std::vector<std::pair<int, int>> adjacentDirections = {
6     {1,0},{-1,0},{0,1},{0,-1} };
7
8 std::vector<std::pair<int, int>> flipDirections = {
9     {1,0},{-1,0},{0,1}, {0,-1}, {1,1},{1,-1},{-1,-1},{-1,1} };
10
11 bool isValidPostition(int const& x, int const& y)
12 {
13     if (x < 0) { return false; }
14     if (y < 0) { return false; }
15     if (x >= input_shape_x) { return false; }
16     if (y >= input_shape_y) { return false; }
17     return true;
18 }
19
20 std::vector<int> AlphaZero::Game::GameState::getFlipActions(int
21     x, int y, int dx, int dy)
22 {
23     std::vector<int> tmp;
24     std::pair<int, int> a = { x + dx, y + dy };
25     auto value = this->player;
26     while (isValidPostition(a.first, a.second))
27     {
28         auto idx = from2dPos(a.first, a.second);
29         auto otherValue = this->IdIndex(idx);
30         if (value == -otherValue)
31         {
32             tmp.push_back(idx);
33         }
34         else if (value == otherValue)
35         {
36             return tmp;
37         }
38         else
39         {
40             return std::vector<int>();
41         }
42     }
43     a = { a.first + dx, a.second + dy };
44 }
45
46 return std::vector<int>();
```



```

41 }
42
43 bool AlphaZero::Game::GameState::hasAjacentStones(int const& x,
44 int const& y)
45 {
46     int otherX, otherY;
47     for (auto const& direction : ajacentDirections)
48     {
49         otherX = x + direction.first;
50         otherY = y + direction.second;
51         if (isValidPostition(otherX, otherY))
52         {
53             if (this->IdIndex(from2dPos(otherX, otherY)) != 0)
54             {
55                 return true;
56             }
57         }
58     }
59     return false;
60 }
61 AlphaZero::Game::GameState::GameState(IDType board, int _player)
62 {
63     this->initialize(board, _player);
64 }
65
66 AlphaZero::Game::GameState::GameState()
67 {
68     IDType board;
69     this->IdIndex(27, -1, board);
70     this->IdIndex(28, 1, board);
71     this->IdIndex(35, 1, board);
72     this->IdIndex(36, -1, board);
73     this->initialize(board, 1);
74 }
75
76 void AlphaZero::Game::GameState::initialize(IDType board, int
77 _player)
78 {
79     this->gameBoard = board;
80     this->player = _player;
81     this->getAllowedActions();
82     this->gameIsDone();
83 }

```

```

84 std::shared_ptr<AlphaZero::Game::GameState> AlphaZero::Game::
    GameState::takeAction(int action)
85 {
86     IDType newBoard = this->gameBoard;
87     GameState::IdIndex(action, this->player, newBoard);
88     std::pair<int, int> pos = to2dPos(action);
89     for (auto const& direction : flipDirections)
90     {
91         auto toFlip = getFlipActions(pos.first, pos.second,
92             direction.first, direction.second);
93         for (auto const& pos : toFlip)
94         {
95             GameState::IdIndex(pos, this->player, newBoard);
96         }
97     }
98     std::shared_ptr<GameState> newState = std::make_shared<
99         GameState>(newBoard, -this->player);
100
101     if (newState->allowedActions.size())
102         return newState;
103
104     std::shared_ptr<GameState> newerState = std::make_shared<
105         GameState>(newBoard, this->player);
106     return newerState;
107 }
108
109 void AlphaZero::Game::GameState::gameIsDone()
110 {
111     int thisPlayerPoints = 0, otherPlayerPoints = 0;
112     this->done = true;
113     for (int x = 0; x < input_shape_x; x++)
114     {
115         for (int y = 0; y < input_shape_y; y++)
116         {
117             auto val = this->IdIndex(x + y * columnOffset);
118             if (val == 0 && this->allowedActions.size())
119             {
120                 this->done = false;
121             }
122             else if (val == this->player)
123             {
124                 thisPlayerPoints++;
125             }
126             else if (val == -this->player)

```

```

125         {
126             otherPlayerPoints++;
127         }
128     }
129 }
130 if (this->done)
131 {
132     int win = 0;
133     if (thisPlayerPoints > otherPlayerPoints)
134     {
135         win = 1;
136     }
137     else if (thisPlayerPoints < otherPlayerPoints)
138     {
139         win = -1;
140     }
141     this->val = { win, thisPlayerPoints, otherPlayerPoints };
142 }
143 else
144 {
145     this->val = { 0, 0, 0 };
146 }
147 }
148
149 void AlphaZero::Game::GameState::getAllowedActions()
150 {
151     this->allowedActions.clear();
152     for (int x = 0; x < input_shape_x; x++)
153     {
154         for (int y = 0; y < input_shape_y; y++)
155         {
156             if (this->IdIndex(from2dPos(x, y)) == 0 &&
157                 hasAjacentStones(x, y))
158             {
159                 for (auto const& direction : flipDirections)
160                 {
161                     auto toFlip = getFlipActions(x, y, direction.first,
162                     direction.second);
163                     if (toFlip.size())
164                     {
165                         this->allowedActions.push_back(from2dPos(x, y));
166                         break;
167                     }
168                 }
169             }
170         }
171     }
172 }

```

```

168     }
169 }
170 }
171
172 void AlphaZero::Game::GameState::render()
173 {
174     std::vector<GameState*> state = { this };
175     renderStates(state);
176 }
177
178 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
    || ModelLogger)
179 void AlphaZero::Game::GameState::render(std::shared_ptr<spdlog::
    logger> logger)
180 {
181     for (int idx = 0; idx < 6; idx++) {
182         char line1[13] = {
183             this->getPiece(0 + columnOffset * idx), ' ',
184             this->getPiece(1 + columnOffset * idx), ' ',
185             this->getPiece(2 + columnOffset * idx), ' ',
186             this->getPiece(3 + columnOffset * idx), ' ',
187             this->getPiece(4 + columnOffset * idx), ' ',
188             this->getPiece(5 + columnOffset * idx), ' ',
189             this->getPiece(6 + columnOffset * idx)
190         };
191         logger->info(line1);
192     }
193 }
194 #endif
195
196 AlphaZero::Game::Game::Game()
197 {
198     this->state = std::make_shared<GameState>();
199 }
200
201 void AlphaZero::Game::Game::reset()
202 {
203     this->state = std::make_shared<GameState>();
204 }
205
206 void AlphaZero::Game::Game::takeAction(int action)
207 {
208     auto newState = this->state->takeAction(action);
209     this->state = newState;
210 }

```

```

211
212 bool AlphaZero::Game::Game::takeHumanAction(int action)
213 {
214     this->takeAction(action);
215     return true;
216 }
217
218 #define assign(idx1, idx2) AlphaZero::Game::GameState::IdIndex(
    idx1, state->IdIndex(idx2), boardBuffer)
219 #define assignState(idx1, idx2) assign(idx1, idx2); assign(idx2,
    idx1)
220
221 inline std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
    std::vector<int>> mirrorGameState(std::shared_ptr<AlphaZero::
    Game::GameState> state, std::vector<int> const& actionProbs)
    {
222     IDType boardBuffer;
223
224     std::vector<int> probs = {
225         actionProbs[7], actionProbs[6], actionProbs[5],
226         actionProbs[4], actionProbs[3], actionProbs[2],
227         actionProbs[1], actionProbs[0],
228         actionProbs[15], actionProbs[14], actionProbs[13],
229         actionProbs[12], actionProbs[11], actionProbs[10],
230         actionProbs[9], actionProbs[8],
231         actionProbs[23], actionProbs[22], actionProbs[21],
232         actionProbs[20], actionProbs[19], actionProbs[18],
233         actionProbs[17], actionProbs[16],
234         actionProbs[31], actionProbs[30], actionProbs[29],
235         actionProbs[28], actionProbs[27], actionProbs[26],
236         actionProbs[25], actionProbs[24],
237         actionProbs[39], actionProbs[38], actionProbs[37],
238         actionProbs[36], actionProbs[35], actionProbs[34],
239         actionProbs[33], actionProbs[32],
240         actionProbs[47], actionProbs[46], actionProbs[45],
241         actionProbs[44], actionProbs[43], actionProbs[42],
242         actionProbs[41], actionProbs[40],
243         actionProbs[55], actionProbs[54], actionProbs[53],
244         actionProbs[52], actionProbs[51], actionProbs[50],
245         actionProbs[49], actionProbs[48],
246         actionProbs[63], actionProbs[62], actionProbs[61],
247         actionProbs[60], actionProbs[59], actionProbs[58],
248         actionProbs[57], actionProbs[56]
249     };

```

```

235
236 assignState(0, 7); assignState(1, 6); assignState(2, 5);
    assignState(3, 4);
237 assignState(8, 15); assignState(9, 14); assignState(10, 13);
    assignState(11, 12);
238 assignState(16, 23); assignState(17, 22); assignState(18, 21);
    assignState(19, 20);
239 assignState(24, 31); assignState(25, 30); assignState(26, 29);
    assignState(27, 28);
240 assignState(32, 39); assignState(33, 38); assignState(34, 37);
    assignState(35, 36);
241 assignState(40, 47); assignState(41, 46); assignState(42, 45);
    assignState(43, 44);
242 assignState(48, 55); assignState(49, 54); assignState(50, 53);
    assignState(51, 52);
243 assignState(56, 63); assignState(57, 62); assignState(58, 61);
    assignState(59, 60);
244
245 return { std::make_shared<AlphaZero::Game::GameState>(
    boardBuffer, state->player), probs };
246 }
247
248 inline std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
    std::vector<int>> rotateGameState(std::shared_ptr<AlphaZero::
    Game::GameState> state, std::vector<int> const& actionProbs)
249 {
250     IDType boardBuffer;
251
252     std::vector<int> probs = {
253         actionProbs[56], actionProbs[48], actionProbs[40],
        actionProbs[32], actionProbs[24], actionProbs[16],
        actionProbs[8], actionProbs[0],
254         actionProbs[57], actionProbs[49], actionProbs[41],
        actionProbs[33], actionProbs[25], actionProbs[17],
        actionProbs[9], actionProbs[1],
255         actionProbs[58], actionProbs[50], actionProbs[42],
        actionProbs[34], actionProbs[26], actionProbs[18],
        actionProbs[10], actionProbs[2],
256         actionProbs[59], actionProbs[51], actionProbs[43],
        actionProbs[35], actionProbs[27], actionProbs[19],
        actionProbs[11], actionProbs[3],
257         actionProbs[60], actionProbs[52], actionProbs[44],
        actionProbs[36], actionProbs[28], actionProbs[20],
        actionProbs[12], actionProbs[4],
258         actionProbs[61], actionProbs[53], actionProbs[45],

```

```

259     actionProbs[37], actionProbs[29], actionProbs[21],
        actionProbs[13], actionProbs[5],
260     actionProbs[62], actionProbs[54], actionProbs[46],
        actionProbs[38], actionProbs[30], actionProbs[22],
        actionProbs[14], actionProbs[6],
        actionProbs[63], actionProbs[55], actionProbs[47],
        actionProbs[39], actionProbs[31], actionProbs[23],
        actionProbs[15], actionProbs[7]
261 };
262
263 auto a = state->IdIndex(0);
264 assign(0, 56); assign(1, 48); assign(2, 40); assign(3, 32);
        assign(4, 24); assign(5, 16); assign(6, 8); assign(7,
        0);
265 assign(8, 57); assign(9, 49); assign(10, 41); assign(11, 33)
        ; assign(12, 25); assign(13, 17); assign(14, 9); assign(15,
        1);
266 assign(16, 58); assign(17, 50); assign(18, 42); assign(19, 34)
        ; assign(20, 26); assign(21, 18); assign(22, 10); assign(23,
        2);
267 assign(24, 59); assign(25, 51); assign(26, 43); assign(27, 35)
        ; assign(28, 27); assign(29, 19); assign(30, 11); assign(31,
        3);
268 assign(32, 60); assign(33, 52); assign(34, 44); assign(35, 36)
        ; assign(36, 28); assign(37, 20); assign(38, 12); assign(39,
        4);
269 assign(40, 61); assign(41, 53); assign(42, 45); assign(43, 37)
        ; assign(44, 29); assign(45, 21); assign(46, 13); assign(47,
        5);
270 assign(48, 62); assign(49, 54); assign(50, 46); assign(51, 38)
        ; assign(52, 30); assign(53, 22); assign(54, 14); assign(55,
        6);
271 assign(56, 63); assign(57, 55); assign(58, 47); assign(59, 39)
        ; assign(60, 31); assign(61, 23); assign(62, 15); assign(63,
        7);
272
273 return { std::make_shared<AlphaZero::Game::GameState>(
        boardBuffer, state->player), probs };
274 }
275
276 #undef assignState
277 #undef assign
278
279 bool canBeAddedToIdentities(std::vector<std::pair<std::
        shared_ptr<AlphaZero::Game::GameState>, std::vector<int>>>

```

```

    const& idents , std::pair<std::shared_ptr<AlphaZero::Game::
    GameState>, std::vector<int>>>const& data)
280 {
281     auto pos = std::find(idents.begin(), idents.end(), data);
282     return pos == idents.end();
283 }
284
285 std::vector<std::pair<std::shared_ptr<AlphaZero::Game::GameState
>, std::vector<int>>>> AlphaZero::Game::identities(std::
    shared_ptr<GameState> state , std::vector<int>& probs)
286 {
287     std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
    GameState>, std::vector<int>>>> idents;
288     int idx = 0;
289     std::pair<std::shared_ptr<GameState>, std::vector<int>>> data =
        { state , probs };
290     std::pair<std::shared_ptr<GameState>, std::vector<int>>>
        mirrored = mirrorGameState(state , probs);
291     for (unsigned short iter = 0; iter < 3; iter++)
292     {
293         if (canBeAddedToIdentities(idents , data))
294             idents.push_back(data);
295         if (canBeAddedToIdentities(idents , mirrored))
296             idents.push_back(mirrored);
297
298         data = rotateGameState(data.first , data.second);
299         mirrored = rotateGameState(mirrored.first , mirrored.second);
300     }
301     if (canBeAddedToIdentities(idents , data))
302         idents.push_back(data);
303     if (canBeAddedToIdentities(idents , mirrored))
304         idents.push_back(mirrored);
305     return idents;
306 }
307
308 void AlphaZero::Game::renderStates(std::vector<GameState*>
    states)
309 {
310     console_mutex.lock();
311     for (int y = 0; y < input_shape_y; y++)
312     {
313         for (auto const& state : states)
314         {
315             for (int x = 0; x < input_shape_x; x++)
316             {

```



```
317         std::cout << state->getPiece(from2dPos(x, y)) << " ";
318     }
319     std::cout << "\t\t";
320 }
321 std::cout << std::endl;
322 }
323 std::cout << std::endl;
324 console_mutex.unlock();
325 }
```

## 5.5 CMakeLists.txt

```
1 # CMakeList.txt : Top-level CMake project file , do global
   configuration
2 # and include sub-projects here.
3 #
4 cmake_minimum_required (VERSION 3.8)
5
6 project ("AlphaZeroPytorch")
7 find_package(CUDA 7.0)
8 if (NOT CUDA_VERSION_STRING EQUAL "")
9     message(STATUS "did not find cuda")
10 else ()
11     message(STATUS "Found CUDA ${CUDA_VERSION_STRING} at ${
        CUDA_TOOLKIT_ROOT_DIR}")
12 endif ()
13
14 # Include sub-projects.
15 #add_subdirectory ("AlphaZeroPytorch")
```

## 5.6 README.md

```
1 # AlphaZero
2 Server baised alpha Zero server and client.
```

## 5.7 serch.sh

```
1 clear
2 echo grep -r $1 .
```

## 6 Demos

### 6.1 Matura-AlphaZero-demos

#### 6.1.1 CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.0.0)
2 project(AlphaZeroDemos VERSION 0.1.0)
3 set(CXX_STANDARD 14)
4
5 include(CTest)
6 enable_testing()
7 message(STATUS ${CMAKE_MODULE_PATH})
8
9 set(libtorch_hash "5
    b5ea3067f878dea051f6cd8fb00338f55517cb6baecdc810983a814e030845
    ")
10
11 # Download libtorch from official website
12
13 if (EXISTS "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip" OR
14     EXISTS "${PROJECT_SOURCE_DIR}/dependencies/libtorch")
15     message(STATUS "libtorch already downloaded")
16 else ()
17     message(STATUS "downloading libtorch")
18     file(
19         DOWNLOAD
20         "https://download.pytorch.org/libtorch/cpu/libtorch-cxx11-abi-shared-with-deps-1.11.0%2Bcpu.zip"
21         "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip"
22     )
23     file(SHA256 "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip"
24          libtorch_checksum)
25     if (libtorch_checksum MATCHES "${libtorch_hash}")
26         message(STATUS "libtorch checksum is valid")
27     else ()
28         message(FATAL_ERROR "libtorch checksum is not valid")
29     endif()
30 endif()
31 #unzip libtorch
32 if (EXISTS "${PROJECT_SOURCE_DIR}/dependencies/libtorch")
33     message(STATUS "libtorch already installed")
34 else ()
35     message(STATUS "installing libtorch")
36
```

```

34 file (ARCHIVEEXTRACT INPUT "${PROJECT_SOURCE_DIR}/dependencies
    /libtorch.zip" DESTINATION "${PROJECT_SOURCE_DIR}/
    dependencies")
35 file (REMOVE "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip")
36 message (STATUS "libtorch installed")
37 endif()
38 message (STATUS "libtorch Path:\t" "${PROJECT_SOURCE_DIR}/
    dependencies/libtorch")
39 set (Torch_DIR "${PROJECT_SOURCE_DIR}/dependencies/libtorch/
    share/cmake/Torch")
40
41
42 find_package (Torch REQUIRED Torch_DIR)
43 set (DCMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${TORCH_CXX_FLAGS}")
44
45 find_package (colormap REQUIRED INTERFACE)
46
47 find_package (wxWidgets REQUIRED gl core base OPTIONAL_COMPONENTS
    net)
48 include (${wxWidgets_USE_FILE})
49
50 file (GLOB SharedFiles
51     "include/*"
52     "src/*"
53 )
54 file (GLOB UnsupervisedFiles
55     "unsupervised/src/*"
56     "unsupervised/include/*"
57 )
58 file (GLOB SupervisedFiles
59     "supervised/src/*"
60     "supervised/include/*"
61 )
62
63 add_executable (VectorQuantization main.cpp ${SharedFiles} ${
    UnsupervisedFiles})
64 target_include_directories (VectorQuantization PUBLIC "include/"
    "unsupervised/include")
65 target_link_libraries (VectorQuantization PRIVATE ${
    wxWidgets_LIBRARIES})
66
67 add_executable (Supervised supervised.cpp ${SharedFiles} ${
    SupervisedFiles})
68 target_include_directories (Supervised PUBLIC "include/" "
    supervised/include")

```

```
69 target_link_libraries(Supervised PRIVATE ${wxWidgets_LIBRARIES}
    ${TORCH_LIBRARIES})
70 target_link_libraries(Supervised INTERFACE ${colormap_DIR})
71 message(STATUS "colormap at: " ${colormap_DIR})
72 #set_property(TARGET Supervised PROPERTY CXX_STANDARD 23)
73
74 set(CPACK_PROJECT_NAME ${PROJECT_NAME})
75 set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
76 include(CPack)
```

### 6.1.2 main.cpp

```
1 #include <wx/sizer.h>
2 #include <wx/timer.h>
3 #include <config.hpp>
4 #include <cluster.hpp>
5
6 class BasicDrawPane;
7
8 class RenderTimer : public wxTimer
9 {
10     BasicDrawPane* pane;
11 public:
12     RenderTimer(BasicDrawPane* pane);
13     void Notify();
14     void start();
15 };
16
17
18 class BasicDrawPane : public wxPanel
19 {
20     VQ::Cluster* cluster;
21 public:
22     BasicDrawPane(wxFrame* parent);
23     ~BasicDrawPane();
24
25     void paintEvent(wxPaintEvent& evt);
26     void paintNow();
27     void render( wxDC& dc );
28
29     DECLARE_EVENT_TABLE()
30 };
31
32 class MyFrame;
33
34 class MyApp: public wxApp
35 {
36     bool OnInit();
37
38     MyFrame* frame;
39 public:
40
41 };
42
43
```



```

44 RenderTimer::RenderTimer(BasicDrawPane* pane) : wxTimer()
45 {
46     RenderTimer::pane = pane;
47 }
48
49 void RenderTimer::Notify()
50 {
51     pane->Refresh();
52 }
53
54 void RenderTimer::start()
55 {
56     wxTimer::Start(10);
57 }
58
59 IMPLEMENT_APP(MyApp)
60
61 class MyFrame : public wxFrame
62 {
63     RenderTimer* timer;
64     BasicDrawPane* drawPane;
65
66 public:
67     MyFrame() : wxFrame((wxFrame *)NULL, -1, wxT("Hello wxDC"),
68         wxPoint(50,50), wxSize(WIDTH,HEIGHT))
69     {
70         wxBoxSizer* sizer = new wxBoxSizer(wxHORIZONTAL);
71         drawPane = new BasicDrawPane( this );
72         sizer->Add(drawPane, 1, wxEXPAND);
73         SetSizer(sizer);
74
75         timer = new RenderTimer(drawPane);
76         Show();
77         timer->start();
78     }
79     ~MyFrame()
80     {
81         delete timer;
82     }
83     void onClose(wxCloseEvent& evt)
84     {
85         timer->Stop();
86         evt.Skip();
87     }
88     DECLARE_EVENT_TABLE()

```

```

88 };
89
90
91 BEGIN_EVENT_TABLE(MyFrame, wxFrame)
92 EVT_CLOSE(MyFrame::onClose)
93 END_EVENT_TABLE()
94
95 bool MyApp::OnInit()
96 {
97     frame = new MyFrame();
98     frame->Show();
99
100     return true;
101 }
102
103
104 BEGIN_EVENT_TABLE(BasicDrawPane, wxPanel)
105 EVT_PAINT(BasicDrawPane::paintEvent)
106 END_EVENT_TABLE()
107
108
109
110 BasicDrawPane::BasicDrawPane(wxFrame* parent) :
111 wxPanel(parent)
112 {
113     auto a = wxGREEN_BRUSH;
114     std::vector<const wxBrush*> brushes;
115     brushes.push_back(wxGREEN_BRUSH);
116     brushes.push_back(wxRED_BRUSH);
117     brushes.push_back(wxBLUE_BRUSH);
118     this->cluster = new VQ::Cluster(POINT_COUNT, (double)WIDTH, (
        double)HEIGHT, brushes);
119 }
120 BasicDrawPane::~BasicDrawPane() {
121     delete this->cluster;
122 }
123
124
125 void BasicDrawPane::paintEvent(wxPaintEvent& evt)
126 {
127     wxPaintDC dc(this);
128     render(dc);
129 }
130
131 void BasicDrawPane::paintNow()

```

```

132 {
133     wxClientDC dc( this );
134     render( dc );
135 }
136
137 void BasicDrawPane::render( wxDC& dc )
138 {
139     this->cluster->update();
140
141     dc.SetBackground( *wxWHITE_BRUSH );
142     dc.SetBrush( *wxBLUE_BRUSH );
143     dc.Clear();
144     this->cluster->render( dc );
145 }

```

### 6.1.3 README.md

```
1 Installation
2 _____
3
4 prerequisites
5   - [wxWidgets](https://wiki.wxwidgets.org/
      Getting_Started_with_wxWidgets)
6   - [colormap](https://github.com/JulianWww/colormap)
7
8
9 to install run:
10 ```
11 $ git clone https://github.com/JulianWww/Matura-AlphaZero-demos
12 $ cd Matura-AlphaZero-demos && mkdir build && cd build
13 $ cmake .. && make
14 ```
```

### 6.1.4 supervised.cpp

```
1 #include <model.hpp>
2 #include <wx/wx.h>
3 #include <wx/sizer.h>
4 #include <wx/timer.h>
5 #include <config.hpp>
6
7 class BasicDrawPane;
8
9 class RenderTimer : public wxTimer
10 {
11     BasicDrawPane* pane;
12 public:
13     RenderTimer(BasicDrawPane* pane);
14     void Notify();
15     void start();
16 };
17
18 class MyFrame;
19 class BasicDrawPane : public wxPanel
20 {
21     private: MyFrame* parent;
22     private: SL::Model* model;
23 public:
24     BasicDrawPane(MyFrame* parent);
25     ~BasicDrawPane();
26
27     void paintEvent(wxPaintEvent& evt);
28     void paintNow();
29     void render( wxDC& dc );
30
31     DECLARE_EVENT_TABLE()
32 };
33
34 class MyApp: public wxApp
35 {
36     bool OnInit();
37
38     MyFrame* frame;
39 public:
40
41 };
42
43
```

```

44 RenderTimer::RenderTimer(BasicDrawPane* pane) : wxTimer()
45 {
46     RenderTimer::pane = pane;
47 }
48
49 void RenderTimer::Notify()
50 {
51     pane->Refresh();
52 }
53
54 void RenderTimer::start()
55 {
56     wxTimer::Start(10);
57 }
58
59 IMPLEMENT_APP(MyApp)
60
61 class MyFrame : public wxFrame
62 {
63     RenderTimer* timer;
64     BasicDrawPane* drawPane;
65
66 public:
67     wxBoxSizer* sizer;
68     MyFrame() : wxFrame((wxFrame *)NULL, -1, wxT("Hello wxDC"),
69         wxPoint(50,50), wxSize(WIDTH,HEIGHT))
70     {
71         sizer = new wxBoxSizer(wxHORIZONTAL);
72         drawPane = new BasicDrawPane( this );
73         sizer->Add(drawPane, 1, wxEXPAND);
74         SetSizer(sizer);
75
76         timer = new RenderTimer(drawPane);
77         Show();
78         timer->start();
79     }
80     ~MyFrame()
81     {
82         delete timer;
83     }
84     void onClose(wxCloseEvent& evt)
85     {
86         timer->Stop();
87         evt.Skip();
88     }

```

```

88     DECLARE_EVENT_TABLE()
89 };
90
91
92 BEGIN_EVENT_TABLE(MyFrame, wxFrame)
93 EVT_CLOSE(MyFrame::onClose)
94 END_EVENT_TABLE()
95
96 bool MyApp::OnInit()
97 {
98     frame = new MyFrame();
99     frame->Show();
100
101     return true;
102 }
103
104
105 BEGIN_EVENT_TABLE(BasicDrawPane, wxPanel)
106 EVT_PAINT(BasicDrawPane::paintEvent)
107 END_EVENT_TABLE()
108
109
110
111 BasicDrawPane::BasicDrawPane(MyFrame* _parent) :
112 wxPanel(_parent, parent(_parent))
113 {
114     model = new SL::Model;
115 }
116 BasicDrawPane::~BasicDrawPane() {
117     delete model;
118 }
119
120
121 void BasicDrawPane::paintEvent(wxPaintEvent& evt)
122 {
123     wxPaintDC dc(this);
124     render(dc);
125 }
126
127 void BasicDrawPane::paintNow()
128 {
129     wxClientDC dc(this);
130     render(dc);
131 }
132

```

```

133 void BasicDrawPane::render( wxDC& dc )
134 {
135     //std::cout << this->model->forward(torch::ones({1,2})) << std
136     //::endl;
137     this->model->train();
138     auto size = this->parent->sizer->GetSize();
139     dc.DrawBitmap(this->model->getMap().Rescale(size.GetX(), size.
140     GetY()), wxPoint(0,0), false);
141     dc.SetBrush(*wxTRANSPARENT_BRUSH);
142     dc.DrawEllipse(
143         size.GetX()/2 - size.GetX() * (double)CIRC_RADIUS/((double)
144         WIDTH),
145         size.GetY()/2 - size.GetY() * (double)CIRC_RADIUS/((double)
146         HEIGHT),
147         2 * size.GetX() * (double)CIRC_RADIUS/((double)WIDTH),
148         2 * size.GetY() * (double)CIRC_RADIUS/((double)HEIGHT)
149     );
150 }

```



## 6.1.5 include

### 6.1.5.1 point.hpp

```
1 #pragma once
2 #include <iostream>
3 #include <wx/wx.h>
4
5 namespace VQ {
6     class Point {
7     public: using T = double;
8
9     public: Point();
10    public: Point(const Point& p);
11    public: Point(const T& x, const T& y);
12    public: Point(const std::pair<T, T>& pos);
13    public: std::pair<T, T> getPos();
14    public: const std::pair<T, T> getPos() const;
15
16    public: void moveTo(const T& x, const T& y);
17    public: void moveTo(const std::pair<T, T>& pos);
18    public: void moveTo(const Point& point);
19    public: void moveBy(const T& dx, const T& dy);
20    public: void moveBy(const std::pair<T, T>& dpos);
21    public: void moveBy(const Point& dpoint);
22
23    public: Point operator*(const float scalar) const;
24    public: Point operator*(const double scalar) const;
25    public: Point operator-(const Point& other) const;
26    public: Point operator+(const Point& other) const;
27    public: Point& operator=(const Point&& other);
28    public: Point& operator=(const std::pair<T, T>&& other);
29    public: friend std::ostream& operator<<(std::ostream& os,
const Point& dt);
30    public: double abs() const;
31    protected: std::pair<T, T> pos;
32
33    public: void render(wxDC& dc, const wxBrush* brush) const;
34    };
35
36    Point randomPointInRange(const double& dx, const double& dy);
37    std::ostream& operator<<(std::ostream& os, const Point& dt);
38 }
```

### 6.1.5.2 utils.hpp

```
1 #pragma once
2
3 #include <cstdlib>
4 #include <vector>
5
6 namespace std {
7     double rand(const double& max);
8     size_t randMod(const size_t& mod);
9 }
10
11 namespace jce {
12     template <typename T>
13     T& randElement(std::vector<T>& vec);
14     template <typename T>
15     const T& randElement(const std::vector<T>& vec);
16 }
17
18 inline double std::rand(const double& max) {
19     return (double)(std::rand()) / (((double)RAND_MAX/max));
20 }
21 inline size_t std::randMod(const size_t& mod) {
22     return std::rand() % mod;
23 }
24
25 template<typename T>
26 T& jce::randElement(std::vector<T>& vec) {
27     return vec[std::randMod(vec.size())];
28 }
29 template<typename T>
30 const T& jce::randElement(const std::vector<T>& vec) {
31     return vec[std::randMod(vec.size())];
32 }
```

## 6.1.6 scr

### 6.1.6.1 point.cpp

```
1 #include <point.hpp>
2 #include <utils.hpp>
3 #include <config.hpp>
4
5 VQ::Point::Point(): pos({0,0}) {}
6 VQ::Point::Point(const VQ::Point& p): pos(p.getPos()) {}
7 VQ::Point::Point(const VQ::Point::T& x, const VQ::Point::T& y):
8     pos({x, y}) {}
9 VQ::Point::Point(const std::pair<VQ::Point::T, VQ::Point::T>&
10     _pos): pos(_pos) {}
11 std::pair<VQ::Point::T, VQ::Point::T> VQ::Point::getPos() {
12     return pos;}
13 const std::pair<VQ::Point::T, VQ::Point::T> VQ::Point::getPos()
14     const {return pos;}
15
16 void VQ::Point::moveTo(const VQ::Point::T& x, const VQ::Point::T
17     & y) {
18     this->moveTo(std::pair<double, double>(x, y));
19 }
20 void VQ::Point::moveTo(const std::pair<VQ::Point::T, VQ::Point::
21     T>& _pos) {this->pos = _pos;}
22 void VQ::Point::moveTo(const Point& point) {
23     this->moveTo(point.getPos());
24 }
25 void VQ::Point::moveBy(const VQ::Point::T& dx, const VQ::Point::
26     T& dy) {
27     this->pos = {
28         dx + this->pos.first,
29         dy + this->pos.second
30     };
31 }
32 void VQ::Point::moveBy(const std::pair<VQ::Point::T, VQ::Point::
33     T>& _pos){
34     this->moveBy(_pos.first, _pos.second);
35 }
36 void VQ::Point::moveBy(const Point& point) {
37     this->moveBy(point.getPos());
38 }
39
40 VQ::Point VQ::Point::operator*(const float scalar) const {
41     return Point(
```

```

34     scalar * this->pos.first ,
35     scalar * this->pos.second
36 );
37 }
38 VQ::Point VQ::Point::operator*(const double scalar) const {
39     return Point(
40         scalar * this->pos.first ,
41         scalar * this->pos.second
42     );
43 }
44 VQ::Point VQ::Point::operator+(const VQ::Point& point) const {
45     return Point(
46         this->pos.first + point.getPos().first ,
47         this->pos.second + point.getPos().second
48     );
49 }
50 VQ::Point VQ::Point::operator-(const VQ::Point& point) const {
51     return Point(
52         this->pos.first - point.getPos().first ,
53         this->pos.second - point.getPos().second
54     );
55 }
56
57 VQ::Point& VQ::Point::operator=(const Point&& other) {
58     return (*this = other.getPos());
59 }
60 VQ::Point& VQ::Point::operator=(const std::pair<VQ::Point::T, VQ
    ::Point::T&& other) {
61     this->pos = other;
62     return *this;
63 }
64 double VQ::Point::abs() const {
65     return std::sqrt(
66         this->pos.first * this->pos.first
67         + this->pos.second * this->pos.second
68     );
69 }
70
71 void VQ::Point::render(wxDC& dc, const wxBrush* brush) const {
72     dc.SetBrush(*brush);
73     dc.DrawCircle(this->pos.first, this->pos.second, POINT_RADIUS)
74     ;
75 }
76 VQ::Point VQ::randomPointInRange(const double& x, const double&

```

```

    y) {
77     return Point(
78         std::rand(x),
79         std::rand(y)
80     );
81 }
82 std::ostream& VQ::operator<<(std::ostream& stream, const VQ::
    Point& p) {
83     stream << "<" << p.getPos().first << "< " << p.getPos().second
        << ">";
84     return stream;
85 }

```

## 6.1.7 supervised

### 6.1.7.1 include

#### 6.1.7.1.1 config.hpp

```
1 #pragma once
2
3 #define HEIGHT      500
4 #define WIDTH       500
5 #define RES_HEIGHT  200
6 #define RES_WIDTH   200
7 #define POINT_RADIUS 10
8 #define CIRC_RADIUS 200
9 #define BATCH_SIZE  512
10 #define EPOCHS       8
11 #define LR            0.1
12 #define Momentum     0.9
13 #define LOSS          torch::mse_loss
```

### 6.1.7.1.2 model.hpp

```
1 #pragma once
2
3 #include <torch/torch.h>
4 #include <wx/bitmap.h>
5 #include <point.hpp>
6
7 namespace SL {
8     class Model : public torch::nn::Module {
9         private: using Optimizer          = torch::optim::SGD;
10        private: using OptimizerOptions    = torch::optim::SGDOptions;
11
12        ;
13
14        private: torch::nn::Linear lin1 , lin2 , lin3;
15        private: torch::nn::LeakyReLU relu;
16        private: torch::nn::Sigmoid sigm;
17        private: Optimizer optim;
18
19        public: Model();
20        public: torch::Tensor forward(torch::Tensor x);
21        public: void fit(torch::Tensor x, torch::Tensor y);
22        public: void train();
23
24        public: wxImage getMap();
25        private: torch::Tensor getMapTensor();
26
27        private: static std::pair<torch::Tensor , torch::Tensor>
28        getTrainingData();
29        private: static double classify(const VQ::Point& point);
30    };
31 }
```

### 6.1.7.2 src

#### 6.1.7.2.1 model.cpp

```
1 #include <model.hpp>
2 #include <config.hpp>
3 #include <colormap/palettes.hpp>
4
5 #define COLORMAP "jet"
6
7 const static VQ::Point center(HEIGHT/2, HEIGHT/2);
8
9 SL::Model::Model() :
10     lin1(register_module("lin1", torch::nn::Linear(2,256))),
11     lin2(register_module("lin2", torch::nn::Linear(256, 16))),
12     lin3(register_module("lin3", torch::nn::Linear(16, 1))),
13     relu(),
14     sigm(),
15     optim(this->parameters(), OptimizerOptions(LR).momentum(
16         Momentum))
17 {}
18
19 torch::Tensor SL::Model::forward(torch::Tensor x) {
20     x = relu(lin1(x));
21     x = relu(lin2(x));
22     return sigm(lin3(x));
23 }
24
25 void SL::Model::fit(torch::Tensor x, torch::Tensor y_true) {
26     torch::Tensor y_pred = this->forward(x);
27     auto loss = LOSS(y_true, y_pred);
28     loss.backward(loss);
29     optim.step();
30     optim.zero_grad();
31 }
32
33 void SL::Model::train() {
34     auto data = this->getTrainingData();
35     for (size_t idx=0; idx < EPOCHS; idx++)
36         this->fit(data.first, data.second);
37 }
38
39 wxImage SL::Model::getMap() {
40     torch::Tensor y = this->forward(this->getMapTensor());
41     wxImage img(RES.HEIGHT, RES.WIDTH);
```



```

41 auto pal = colormap::palettes.at(COLOMAP).rescale(0, 1);
42 for (size_t idx_x=0; idx_x < RES_HEIGHT; idx_x++) {
43     for (size_t idx_y=0; idx_y < RES_WIDTH; idx_y++) {
44         auto pix = pal(y[idx_x*RES_WIDTH + idx_y][0].item<float>());
45     };
46     img.SetRGB(idx_x, idx_y,
47         pix.getRed().getValue(),
48         pix.getGreen().getValue(),
49         pix.getBlue().getValue());
50 }
51 }
52 return img;
53 }
54
55 torch::Tensor SL::Model::getMapTensor() {
56     torch::Tensor out = torch::zeros({RES_HEIGHT * RES_WIDTH, 2});
57     for (size_t idx_x=0; idx_x < RES_HEIGHT; idx_x++) {
58         for (size_t idx_y=0; idx_y < RES_WIDTH; idx_y++) {
59             out[idx_x*RES_WIDTH + idx_y][0] = (double)idx_x/((double)
60             RES_HEIGHT);
61             out[idx_x*RES_WIDTH + idx_y][1] = (double)idx_y/((double)
62             RES_WIDTH);
63         }
64     }
65     return out;
66 }
67
68 std::pair<torch::Tensor, torch::Tensor> SL::Model::
69     getTrainingData() {
70     torch::Tensor x = torch::ones({BATCH_SIZE, 2});
71     torch::Tensor y = torch::ones({BATCH_SIZE, 1});
72
73     for (size_t idx=0; idx < BATCH_SIZE; idx++) {
74         const VQ::Point p = VQ::randomPointInRange(WIDTH, HEIGHT);
75         y[idx][0] = classify(p);
76         x[idx][0] = p.getPos().first / (double)HEIGHT;
77         x[idx][1] = p.getPos().second / (double)WIDTH;
78     }
79     return {x, y};
80 }
81
82 double SL::Model::classify(const VQ::Point& point) {
83     if ((point - center).abs() > CIRC_RADIUS) {
84         return 1.0;
85     }
86 }

```

```
82     return 0.0;  
83 }
```

## 6.1.8 unsupervised

### 6.1.8.1 include

#### 6.1.8.1.1 cluster.hpp

```
1 #pragma once
2
3 #include <group.hpp>
4 #include <vector>
5
6 namespace VQ {
7     class Cluster {
8     private: std::vector<Point> points;
9             std::vector<Group> groups;
10    public: Cluster(const size_t& points, const double& x,
11                  const double& y, const std::vector<const wxBrush*>& brushes);
12             void update();
13             void render(wxDC& dc);
14    private: Group* getClosestGroup(const Point& point);
15             const Group* getClosestGroup(const Point& point)
16             const;
17             const wxBrush* getPointBrush(const Point& point)
18             const;
19             void generatePoints(const size_t& count, const
20                                double& x, const double& y);
21             void generateGroups(const double& x, const double&
22                                y, const std::vector<const wxBrush*>& brushes);
23     };
24 }
```

#### 6.1.8.1.2 config.hpp

```
1 #pragma once
2
3 #define HEIGHT          500
4 #define WIDTH           500
5 #define POINT_COUNT     100
6 #define POINT_RADIUS    5
7 #define GRPOUT_SIZE     15
```

### 6.1.8.1.3 group.hpp

```
1 #pragma once
2 #include <point.hpp>
3
4 namespace VQ {
5     class Group: public Point {
6         protected: const wxBrush* brush;
7
8         public: Group();
9         public: Group(const T& x, const T& y, const wxBrush* brush);
10        public: Group(const std::pair<T, T>& pos, const wxBrush*
11        brush);
12        //public: ~Group();
13        public: void render(wxDC& dc) const;
14        public: const wxBrush* getBrush() const;
15    };
16
17    Group randomGroupInRange(const double& x, const double& y,
18    const wxBrush* brush);
19 }
```

## 6.1.8.2 src

### 6.1.8.2.1 cluster.cpp

```
1 #include <cluster.hpp>
2 #include <climits>
3 #include <float.h>
4 #include <utils.hpp>
5
6 VQ::Cluster::Cluster(const size_t& count, const double& x, const
    double& y, const std::vector<const wxBrush*>& brushes) {
7     this->generatePoints(count, x, y);
8     this->generateGroups(x, y, brushes);
9 }
10
11 void VQ::Cluster::update() {
12     Point& point = jce::randElement(this->points);
13     Group* group = this->getClosestGroup(point);
14     Point delta = (point - (*group)) * 0.01;
15     group->moveBy(delta);
16 }
17
18 void VQ::Cluster::render(wxDC& dc) {
19     for (auto const point : this->points) {
20         point.render(dc, this->getPointBrush(point));
21     }
22     for (auto const group : this->groups) {
23         group.render(dc);
24     }
25 }
26
27 VQ::Group* VQ::Cluster::getClosestGroup(const VQ::Point& point)
28 {
29     double distance = DBLMAX;
30     Group* out = nullptr;
31     for (auto& group : this->groups) {
32         double other = (group - point).abs();
33         if (other < distance) {
34             distance = other;
35             out = &group;
36         }
37     }
38     return out;
39 }
40
41 const VQ::Group* VQ::Cluster::getClosestGroup(const VQ::Point&
    point) const {
42     double distance = DBLMAX;
```

```

39     const Group* out = nullptr;
40     for (auto const& group : this->groups) {
41         double other = (group - point).abs();
42         if (other < distance) {
43             distance = other;
44             out = &group;
45         }
46         else {
47             int a = 1;
48         }
49     }
50     return out;
51 }
52 const wxBrush* VQ::Cluster::getPointBrush(const Point& point)
53     const {
54     return this->getClosestGroup(point)->getBrush();
55 }
56 void VQ::Cluster::generatePoints(const size_t& count, const
57     double& x, const double& y) {
58     this->points.clear();
59     this->points.resize(count);
60     for (auto iter = this->points.begin(); iter != this->points.
61         end(); iter++) {
62         *iter = randomPointInRange(x, y);
63     }
64 }
65 void VQ::Cluster::generateGroups(const double& x, const double&
66     y, const std::vector<const wxBrush*>& brushes) {
67     this->groups.clear();
68     this->groups.resize(brushes.size());
69     for (size_t idx = 0; idx < brushes.size(); idx++) {
70         groups[idx] = randomGroupInRange(x, y, brushes[idx]);
71     }
72 }

```

### 6.1.8.2.2 group.cpp

```
1 #include <group.hpp>
2 #include <config.hpp>
3 #include <utils.hpp>
4
5 VQ::Group::Group(): Group(0,0,nullptr) {}
6 VQ::Group::Group(const VQ::Group::T& x, const VQ::Group::T& y,
7     const wxBrush* _brush): VQ::Point(x, y), brush(_brush) {}
8 VQ::Group::Group(const std::pair<VQ::Group::T, VQ::Group::T>&
9     pos, const wxBrush* _brush): VQ::Point(pos), brush(_brush) {}
10 //VQ::Group::~~Group() { delete brush; }
11
12 VQ::Group VQ::randomGroupInRange(const double& x, const double&
13     y, const wxBrush* brush) {
14     return Group(
15         std::rand(x),
16         std::rand(y),
17         brush
18     );
19 }
20 void VQ::Group::render(wxDC& dc) const {
21     dc.SetBrush(*(this->brush));
22     dc.DrawRectangle(wxPoint(this->pos.first, this->pos.second),
23         wxSize(GRPOUT_SIZE, GRPOUT_SIZE));
24 }
25 const wxBrush* VQ::Group::getBrush() const { return this->brush;
26 }
```



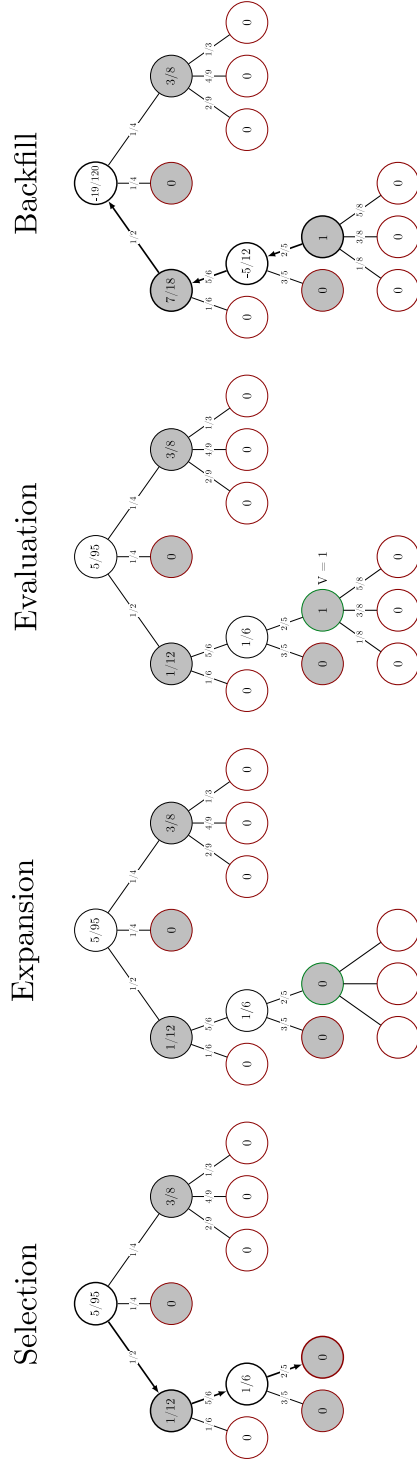


Figure 10: MCTS simulation steps. In this diagram, the numbers in the node represent  $Q$  and the number on the arrow is  $P$ . The red nodes are leaf nodes and the green one is the leaf node  $n_L$ . During the **selection** phase,  $\sigma$  is used to find successive nodes until the leaf node  $n_L$  is reached. This is shown with the arrows. During the **expansion** phase, new nodes and edges are added for all possible legal actions at the node  $n_L$ . The **evaluation** phase gives the new nodes the following values  $Q = 0$  and  $P = \pi_a$ . The **value** of the leaf  $v$  is then used during the **backfill** phase to update the  $Q$ 's of all nodes traversed during selection.

Source: modified from <https://en.wikipedia.org/wiki/File:MCTS-steps.svg>  
File available under Creative Commons Attribution-Share Alike 4.0 International at <https://wandhoven.ddns.net/edu/AlphaZeroTheory/images/MCTS-steps.svg>