Mastering the game of Connect 4 through self-play

Julian Wandhoven
fgz

September 21, 2022

Abstract

Alpha Zero is an AI algorythem, 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 Serch. This document also explains how neural networks work and a short explenation of the infrastructure around the AI to allow for playing on remote devices. [5][6]

Contents

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++. Furthermore all agents have been evaluated against each other using the Elo evaluation system [2]. Additionally, I have added a short introduction on how neural networks work and how they are trained in section 1.4 on page 17.

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. Training an AI by teaching it to prefer advantagious actions is called reinforcement learning.

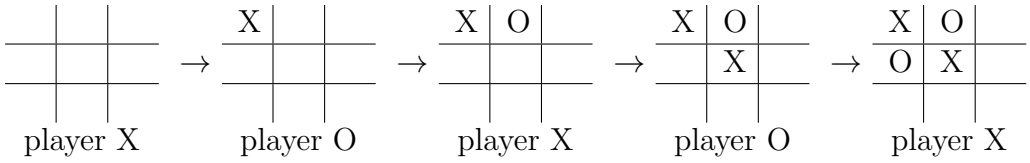
1.1 Reinforcement Learning

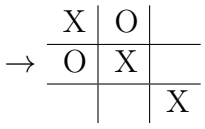
When training neural networks, there are three major possible situations: Supervised learning, unsupervised learning, and reinforcement learning. The first uses predetermined data with known in- and outputs the network is trained to predict. An example of supervised learning is the recognition of handwriting 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 algorythems 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 betwean them. The sample point is selected from the input data [7]. An example of both supervised and unsupervised learning can be seen in the demo[1].

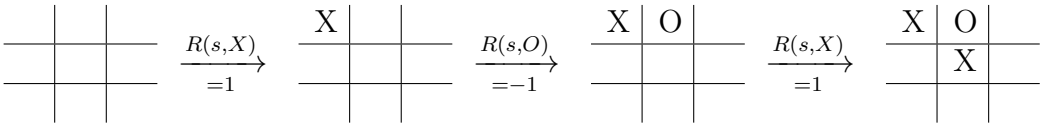
These two methods represent the extreme ends of the spectrum. Reinforcement leaning on the other hand can be thought of as an intermediate form. It uses a predetermined environment which gives positive, neutral and negative feedback. The neural network is then discouraged from taking actions leading 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 agents 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 1.5.1.2 on page 28). When the game reaches an end state and a winner is determined, the feedback is propagated backwards up threw all states used to get to the end state. 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 for that player, 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 $R(s, a_s) = 0$. Every agent p will try to maximize $\sum_{s \in g \cap p} R(s, a_s)$. $g \cap p$ is the set of all states in which the player p takes an action. Let's look at a tic tac toe example of the following game:

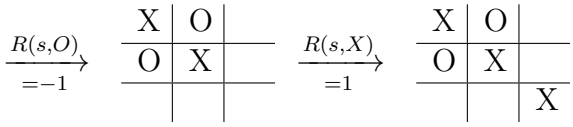
[1] demo is at https://github.com/JulianWww/Matura-AlphaZero-demos





Since player X won the game, the reward for every state $s \in g \cap X$ is $R(s, a_s) = 1$ and the reward for every state $s \in g \cap O$ is $R(s, a_s) = -1$. 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

In order to train a reinforcement learning AI, it must interact with an environment. In Alpha Zero the game is the the environment. The game consists of a series 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, every game ends in an end game state. At an end game state one

player won or the game ended in a tie. Togather the states form a graph. Any possible path thought this graph starting from a root state (the inital state of the board) and ending in any end game state, is a possible game. From an end game state, there is no where left to go. For connect4 an end game state has either four stones in a line or a full game board. Let $\mathbb{G}$ be the set of all legal game states. Let $\mathbb{G}_{done} \subset \mathbb{G}$ be the set of all game states for which the game is done. At every gamestate one player is at turn. This means that that player will take an action next. Let $\phi(s) : \mathbb{G} \to \{1, -1\}$ be the function mapping states to players. In the tick-tack-toe example from earlyer we could say that:

$$\phi(s) = \begin{cases} 1 & s \in g \cap X \\ -1 & s \in g \cap O \end{cases} \tag{1}$$

More generally $\phi(s) = 1$ if the first player (the player that starts the game) is at turn and $\phi(s) = -1$ if the other player is a turn.

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.

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

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 $T_s(s) : \mathbb{G} \to \mathbb{R}^{\cdots}$ is the board's 3-dimensional board tensor. The *vec* function is

defined in section 8.5.1 on page 242.

$$vec\left(\left[\begin{bmatrix} a_{111} & \cdots & a_{11o} \\ \vdots & \ddots & \vdots \\ a_{1n1} & \cdots & a_{1no} \end{bmatrix} \cdots \begin{bmatrix} a_{m11} & \cdots & a_{m1o} \\ \vdots & \ddots & \vdots \\ a_{mn1} & \cdots & a_{mno} \end{bmatrix}\right]\right) = \begin{pmatrix} a_{111} \\ \vdots \\ a_{11o} \\ \vdots \\ a_{1n1} \\ \vdots \\ a_{1no} \\ \vdots \\ a_{m11} \\ \vdots \\ a_{m1o} \\ \vdots \\ a_{mn1} \\ \vdots \\ a_{mno} \end{pmatrix} \tag{2}$$

This operation for the tic tac toe board from before would look like this:

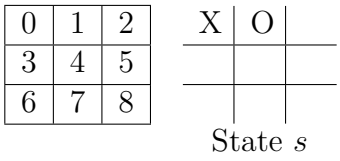
$$vec\left(T_s\left(\begin{array}{c|c|c} & \text{X} & \text{O} \\ \hline \text{O} & \text{X} & \\ \hline \text{O} & & \text{X} \end{array}\right)\right) = [0\,1\,0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,1\,0\,0\,1\,0\,0]^T \tag{3}$$

1.2.2 Actions

Actions are numbers used to identify changes to the game. Every game has a set of all possible actions $\mathbb{A}_{possible} \subset \mathbb{N}_0$. In connect4, the set of all possible actions for the current player is $\mathbb{A}_{possible} = [0, 41] \cap \mathbb{N}$. 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 $\mathcal{A} : \mathbb{G} \times \mathbb{A} \to \mathbb{G}$ 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:



State s

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

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. In Alpha Zero it is used to improve the neural networks prediction. Because a MCTS changes during simulation, idices are used to specify which simulation step the tree is in. A MCTS simulation consists of three phases Leaf Selection, Node Evaluation and Expansion and Backfill.

The MCTS graph consists of nodes and edges. The edges represent connections betwean nodes while the nodes represent game states. Let $\mathbb{M}_{possible} = \{\mathcal{N}(s) \mid s \in \mathbb{G}\}$ be the set of all possible nodes , where $\mathcal{N} : \mathbb{G} \to \mathbb{M}_{possible}$ is a bijective function that maps a game state to a node. $\mathcal{S} : \mathbb{M}_{possible} \to \mathbb{G} = \mathcal{N}^{-1}$ is the inverse of $\mathcal{N}$. For notational simplicity, the set of all allowed actions for any node $n \in \mathbb{M}_{possible}$ is $\mathbb{A}(n) = \mathbb{A}(\mathcal{S}(n))$. The

ammount of nodes and structure of the MCTS is changed by the algorythem itself during simulation. Therefor Let l be the index of the current simulation step. Let $\mathbb{L} = [0, S] \cap \mathbb{N}$ be the set of all l, where S is a constant defined in the algorythems configuration file (section 6.1.11.1 on page 49). During the first simulation, the Tree contains only the root node n_0. The root node is the starting point of the simulation. Let $\mathbb{M}_l \subseteq \mathbb{M}_{possible}$ be the set of all nodes in the tree at step $l \in \mathbb{L}$.

Every node $n_l \in \mathbb{M}_l$ has a set of edges $\mathbb{E}_l(n_l)$ at step l that point from it to another nodes. Let $\mathbb{E}_l$ be the set of all edges in the current graph. $\mathcal{E}_l : \mathbb{M}_l \times \mathbb{A}_{possible} \to \mathbb{E}_l$ is a bijective function used to map nodes and actions to edges. Furthermore for $\mathcal{E}_l(n_l, a)$ to be valid a must be an element of $\mathbb{A}(n_l)$. The function $\mathcal{N}_{to_l} : \mathbb{E}_l \to \mathbb{M}_l$ maps an edge to the node it is pointing to while $\mathcal{N}_{from_l} : \mathbb{E}_l \to \mathbb{M}_l$ is used to find the node an edge is pointing from. Furthermore it is usefull to distingush expanded nodes from leaf nodes. Leaf nodes are nodes that don't have edges leading out of them. Let $\mathbb{M}_{leaf_l} \subseteq \mathbb{M}_l$ be the set of leafnodes. For every node $n_l \in \mathbb{M}_{leaf_l}$, the following is true by definition $\mathbb{E}_l(n_l) = \emptyset$. Expanded nodes are have edges leading out of them. Let $\mathbb{M}_{expanded_l} \subseteq \mathbb{M}_l$ be the set of expanded nodes. For every node $n_l \in \mathbb{M}_{leaf_l}$, the following is true by definition $\mathbb{E}_l(n_l) \neq \emptyset$.

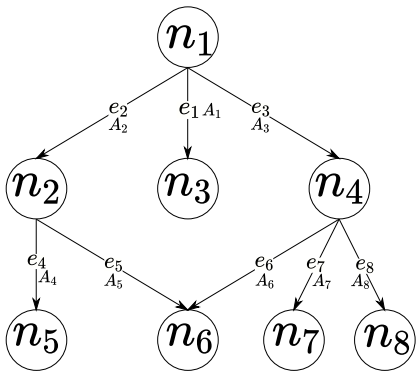


Figure 1: In the following small MCTS tree represents a possible graph at step l, where $n_?$ are nodes, $e_?$ are edges connecting the nodes and $A_?$ are actions associated with the edge they are next to. The questionmark represents the indices. The set of all nodes $\mathbb{M}_l = \{n_1, n_2, \ldots, n_8\}$ and $\mathbb{E}_l = \{e_1, e_2, \ldots, e_5\}$. We can also see that $\mathbb{E}_l(n_1) = \{e_1, e_2, e_3\}$, $\mathbb{E}_l(n_2) = \{e_4, e_5\}$, $\mathbb{E}_l(n_3) = \emptyset$ and so forth. Because e_1 represents action A_1 taken at node n_1, $\mathcal{E}_l(n_1, A_1) = e_1$. By the same logic $\mathcal{E}_l(n_2, A_2) = e_2$ and so forth. Becuase edge e_1 connects node n_1 to n_3, $\mathcal{N}_{from_l}(e_1) = n_1$ and $\mathcal{N}_{to_l}(e_1) = n_3$. The same logic applies to all other nodes. The set of all expanded nodes $\mathbb{M}_{expanded_l} = \{n_1, n_2, n_4\}$ because they have at least one edge leading away from them. On the other hand, the set of all leaf nodes $\mathbb{M}_{leaf_l} = \{n_3, n_5, n_6, n_7, n_8\}$. Finally $\mathbb{M} = \mathbb{M}_{expanded} \cup \mathbb{M}_{leaf}$ (see section 9.1 on page 243 for all data)

1.3.1 Evaluation Basis

The MCTS's goal is to find a good estimations of the reward for a certain action at a certain state. This reward estimation is $Q_l : \mathbb{E}_l \to \mathbb{R}$. To define Q_l, the functions $W_l : \mathbb{E}_l \to \mathbb{R}$ and $N_l : \mathbb{E}_l \to \mathbb{N}_0$ are required. $N_l(e_l)$ is the amount of times an edge $e_l \in \mathbb{E}_l$ has been traversed. This means how many times σ (equation 6 on page 15) has chosen to follow the edge e_l to a new node. $W_l(e_l)$ is the sum of the reward computations from all $N(e_l)$ times the edge has been evaluated. With these two functions Q_l is defined as:

$$Q_l(e_l) = \begin{cases} 0 & N_l(e_l) = 0 \\ \frac{W(e_l)}{N(e_l)} & \end{cases} LeakyReLU \tag{4}$$

The fourth and last of these functions is $P_l : \mathbb{E}_l \to \mathbb{R}$. P_l is the policy function, it's the neural network's preliminary estimation of how valuable the action is. To see what $P_l(e_l)$ is trained to aproximate see equation 46 on page 27. This function is used to guide the search to more promising edges.

1.3.2 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 $n \in \mathbb{M}_l$ the function σ_l is used. To define σ_l we must first define the edge evaluation function $v_l : \mathbb{E}_l \to \mathbb{R}$. v_l is defined as follows:

$$v_l(e) = Q_l(e) + c_{puct} P_l(e) \cdot \frac{\sqrt{\Sigma_{b \in \mathbb{E}(\mathcal{N}_{from_l}(e))} N_l(b)}}{1 + N_l(e)} \tag{5}$$

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. σ_l, for a given node $n \in \mathbb{M}_l$, is then defined as:

$$\sigma_l(n) = \mathcal{N}_{to_l}(\text{argmax}_l(\mathbb{E}_l(n))) \tag{6}$$

argmax_l returns the edge $e \in \mathbb{E}_l$ with the largest $v_l(e)$.
In order to find a leaf node n_{L_l} starting from the root node n_0 and be able to update the tree later on, the following algorithm is run L times, until a leaf node is found. First let $i \in [0, L[\cap \mathbb{N}$ be the index of the iteration of this algorithm. Let $\mathbb{M}_{back,l,i} \subseteq \mathbb{M}$ be the set of nodes traversed by the algorithm at set i during simulation l. By definition $\mathbb{M}_{back,l,0} := \{n_0\}$. Than:

$$n_{L_i+1} = \sigma(n_{L_i}) \tag{7}$$

$$\mathbb{M}_{back,l,i+1} = \mathbb{M}_{back,l,i} \cup \{n_{L_i+1}\} \tag{8}$$

Once a leaf-node $n_{L_l} \in \mathbb{M}_{leaf_l}$ is found that node is evaluated. $\mathbb{M}_{back,l,L}$ will be important in section 1.3.4 on page 17.

1.3.3 Node Evaluation and Expansion

When a leaf node n_{L_l} is reached, there are two possible cases. Ether $\mathcal{S}(n_{L_l})$ is an end game state or not.

1.3.3.1 end game state $\mathcal{S}(n_{L_l}) \in \mathbb{G}_{done}$

In this case the tree is not changed but backfill is still performed. If the player at turn at the end game state is $\phi(\mathcal{S}(n_{L_l}))$. The value $v_l \in \mathbb{R}_{-1,1}$ (76) of n_{L_l} is defined as:

$$v_l = \begin{cases} -\phi(\mathcal{S}(n_{L_l})) & \phi(\mathcal{S}(n_{L_l})) \text{ lost} \\ 0 & \phi(\mathcal{S}(n_{L_l})) \text{ tied} \\ \phi(\mathcal{S}(n_{L_l})) & \phi(\mathcal{S}(n_{L_l})) \text{ won} \end{cases} \tag{9}$$

1.3.3.2 not end game state $\mathcal{S}(n_{L_l}) \notin \mathbb{G}_{done}$

In this case the nodes state is passed to the neural network (section 1.4 on page 17). The neural networks prediction function outputs a policy vector $\pi_l \in \mathbb{R}_{0,1}^{|\mathbb{A}|}$ (76) and a scalar $v_l \in \mathbb{R}_{-1,1}$ (76). The scalar v_l is the neural networks estimation of the expected reward at n_{L_l}. π_l is the estimation of the advantageousness of every action in $\mathbb{A}$. After evaluation, the leaf node is expanded. Let $\mathbb{E}_{new}$ be the next set of all edges pointing away from n_{L_l}. This set contains one edge for every action in $\mathbb{A}(n_{L_l})$. Let $\mathbb{M}_{new}$ be the set of nodes, that the nodes in $\mathbb{E}_{new}$ are pointing to. The various Tree sets and functions are then updated as follows for step $l+1$.

$$\mathbb{M}_{l+1} = \mathbb{M}_l \cup \mathbb{M}_{new} \tag{10}$$

$$\mathbb{E}_{l_+1} = \mathbb{E}_l \cup \mathbb{E}_{new} \tag{11}$$

$$\mathbb{E}_{l+1}(n) = \begin{cases} \mathbb{E}_{new} & n = N_{L_l} \\ \mathbb{E}_l(n) \end{cases} \tag{12}$$

$$\mathcal{E}_{l+1}(n,a) = \begin{cases} \mathcal{E}_l(n,a) & n \neq n_{L_l} \\ e \end{cases} \tag{13}$$

Where $e \in \mathbb{E}_{new}$ is the edge associated with action a at node n_{L_l}.

$$\mathcal{N}_{to_{l+1}}(e) = \begin{cases} \mathcal{N}_{to_l}(e) & e \in \mathbb{E}_l \\ n \end{cases} \tag{14}$$

Where $n \in \mathbb{M}_{new}$ is the node, that edge e is pointing to.

$$\mathcal{N}_{from_{l+1}}(e) = \begin{cases} \mathcal{N}_{from_l}(e) & e \in \mathbb{E}_l \\ n_{L_l} \end{cases} \tag{15}$$

$$\mathbb{M}_{expanded_{l+1}} = \{n \in \mathbb{M}_{l+1} \ : \ \mathbb{E}_{l+1}(n) \neq \emptyset\} \tag{16}$$

$$\mathbb{M}_{leaf_{l+1}} = \{n \in \mathbb{M}_{l+1} \ : \ \mathbb{E}_{l+1}(n) = \emptyset\} \tag{17}$$

$$P_{l+1}(\mathcal{E}_{l+1}(n,a)) = \begin{cases} P_l(\mathcal{E}_{l+1}(n,a)) & e \in \mathbb{E}_l \\ \pi_{l_a} \end{cases} \tag{18}$$

1.3.4 Backfill

The value v_l is used to update the P, W and Q function as follows for every edge $e \in \mathbb{E}_{l+1}$:

$$N_{l+1}(e) = \begin{cases} 0 & e \notin \mathbb{E}_l \\ N_l(e) + 1 & \mathcal{N}_{to_{l+1}}(e) \in \mathbb{M}_{back,l,L} \wedge \mathcal{N}_{from_{l+1}}(e) \in \mathbb{M}_{back,l,L} \\ N_l(e) \end{cases} \tag{19}$$

$$W_{l+1}(e) = \begin{cases} 0 & e \notin \mathbb{E}_l \\ W_l(e) + v_l & \mathcal{N}_{to_{l+1}}(e) \in \mathbb{M}_{back,l,L} \wedge \mathcal{N}_{from_{l+1}}(e) \in \mathbb{M}_{back,l,L} \\ W_l(e) \end{cases} \tag{20}$$

Q_{l+1} is already defined according to (4). $\mathcal{N}_{to_{l+1}}(e) \in \mathbb{M}_{back,l,L} \wedge \mathcal{N}_{from_{l+1}}(e) \in \mathbb{M}_{back,l,L}$ checks if the edge e has been traversed during leaf selection. [1] In constrast to [5], this implementation stores the MCTS simulation values in the edges to allow for the fact that actions from two differant nodes leading to the same node are not identical and therefore should not be treated as such. But nodes of the same game state are identical and should be treated as such.

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 [9]. 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 $1 \times n$ dimentional 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 $\delta_{wb} : \mathbb{R}^n \to \mathbb{R}^m$

$$\delta_{wb}(x) = w \cdot x + b \tag{21}$$

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 $\xi_k : \mathbb{R}^{m\times n\times l} \to \mathbb{R}$ is defined as:

$$\xi_k(i) = \langle w_k, i \rangle_I + b \tag{22}$$

where $\langle \rangle_I$ is the Tensor inner product defined in equation 70 on page 241. The convolutional operation $\Lambda : \mathbb{R}^{i\times j\times l} \to \mathbb{R}^{i-m+1\times j-n+1\times|\mathbb{K}|}$ is an element wise opperation. Given that $I \in \mathbb{R}^{i\times j\times l}$ is the layer input, every element of

$\Lambda(I)_{abc}$ with $a \in [1, i-m+1] \cap \mathbb{N}$, $b \in [1, j-n+1] \cap \mathbb{N}$ and $c \in [1, |\mathbb{K}|] \cap \mathbb{N}$ is defined as:

$$\Lambda(I)_{abc} = \xi_{k_c}(\langle I \rangle_{S_{m \times n,\, ab}}) \tag{23}$$

The submatrix indexing operation $\langle \rangle_s$ is defined in section 8.4 on page 241. For example given the following input tensor $I \in \mathbb{R}^{4\times4\times1}$:

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

and the following kernel weight matrix $w_k \in \mathbb{R}^{3\times3\times1}$ 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 $\Lambda(I) \in R^{2\times2\times1}$. To calculate $\Lambda(I)_{111}$, we compute the kernel operation $\xi_{k_1}(\langle I \rangle_{S_{3\times3,\,11}})$

$$\Lambda_{111}\left(\begin{bmatrix} [3] & [0] & [1] & [5] \\ [2] & [6] & [2] & [4] \\ [2] & [4] & [1] & [0] \\ [3] & [0] & [1] & [5] \end{bmatrix}\right) = \begin{bmatrix} [3] & [0] & [1] \\ [2] & [6] & [2] \\ [2] & [4] & [1] \end{bmatrix} \circ \begin{bmatrix} [-1] & [0] & [1] \\ [-2] & [0] & [2] \\ [-1] & [0] & [1] \end{bmatrix} + 7 \tag{24}$$

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

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 layer evaluation 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. In order to represent non linear functions, a non linear activation function f_a is added between two neural network layers. Thus, the chained function becomes $c(x) = f_1(f_a(f_2(x)))$. In this neural network, three different activation functions are used: $tanh$, $softmax$, and $LeakyReLU$. These functions are defined as follows:

tanh:

$\mathbb{R} \to \mathbb{R}$

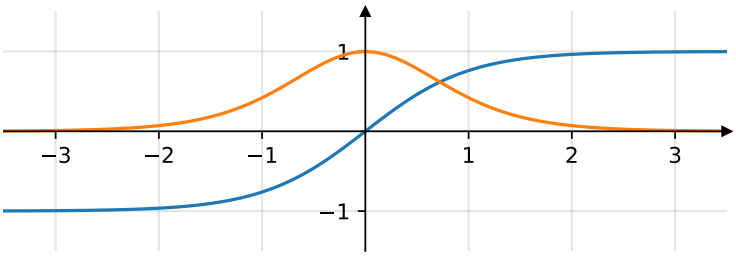


Figure 2: tanh function in blue and the tanh's derivative is in orange

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{25}$$

$$\frac{d}{dx} tanh(x) = sech(x)^2 \tag{26}$$

see section 10.1 on page 244 for proof.
Therefor for a given n dimensional vector $v \in \mathbb{R}^n$, $tanh(v)$ is deffined as:

$$tanh(v) = \langle tanh, x \rangle_E \tag{27}$$

Where $\langle \rangle_E$ is defined in equation 75 on page 242. For simplicity later on the derivative of $tanh(v)$, $tanh'(v) \in \mathbb{R}^{n \times n}$ is a $n \times n$ dimensional matrix. It is defined as

$$tanh'(v)_{ij} = \begin{cases} tanh'(v_{ij}) & i = j \\ 0 \end{cases} \tag{28}$$

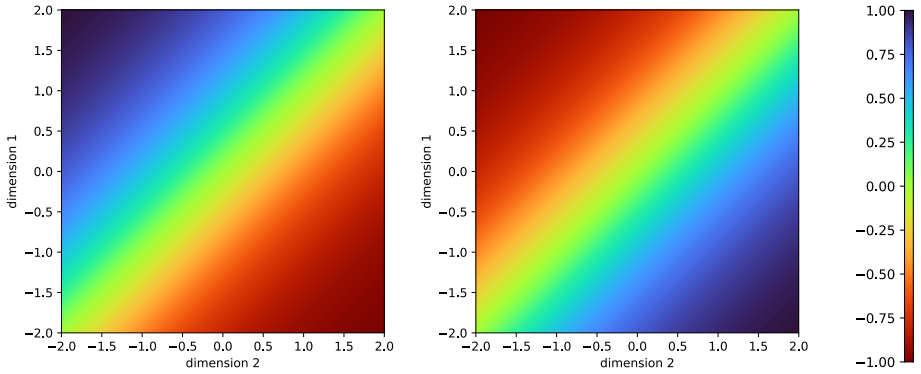
for every element i, j in the derivative matrix.

softmax:

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

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

$$o_i = softmax(v)_i = \frac{e^{v_i}}{\sum_{j \in v} e^j} \tag{29}$$



(a) $i = 1$ (b) $i = 2$ scale

Figure 3: Graph of the softmax function from $\mathbb{R}^2 \to \mathbb{R}^2$. i is the index of the output dimension. Therefore, $i = 1$ referes to the output's first dimension and $i = 2$ referes 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, $\frac{d}{dv_j} softmax(v)_i$, there are two possible cases.

case $j = i$:

$$\frac{d}{dv_j}\left[\frac{e^{v_i}}{\sum_{b \in v} e^b}\right] = softmax(v)_i \cdot (1 - softmax(v)_j) \tag{30}$$

case $j \neq i$:

$$\frac{d}{dv_j}\left[\frac{e^{v_i}}{\sum_{b \in v} e^b}\right] = -softmax(v)_i \cdot softmax(v)_j \tag{31}$$

Therefore, the derivative of the softmax function is:

$$softmax'(v)_{ij} = \begin{cases} softmax(v)_i \cdot (1 - softmax(v)_j) & i = j \\ -softmax(v)_i \cdot softmax(v)_j & i \neq j \end{cases} \tag{32}$$

LeakyReLU:

$\mathbb{R} \to \mathbb{R}$

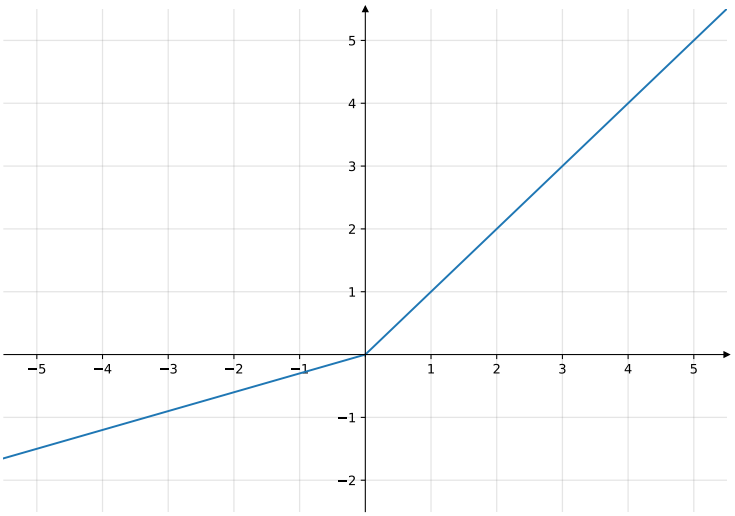


Figure 4: LeakyReLU with $c = 0.3$

$$\text{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ x \cdot c & x < 0 \end{cases} \tag{33}$$

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

$$\text{LeakyReLU}'(x) = \begin{cases} 1 & x > 0 \\ c & x < 0 \end{cases} \tag{34}$$

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 befined for all $x \in \mathbb{R}$. A possible definition that accomplisches the objective is:

$$g(x) = \begin{cases} 1 & x \geq 0 \\ c & x < 0 \end{cases} \tag{35}$$

Therefor for a given n dimensional vector $v \in \mathbb{R}^n$, LeakyReLU(v) is deffined as:

$$\text{LeakyReLU}(v) = \langle \text{LeakyReLU}, x \rangle_E \tag{36}$$

Where $\langle \rangle_E$ is defined in equation 75 on page 242. For simplicity later on the derivative of LeakyReLU(v), LeakyReLU$'(v) \in \mathbb{R}^{n \times n}$ is a $n \times n$ dimensional matrix. It is defined as

$$\text{LeakyReLU}'(v)_{ij} = \begin{cases} g(v_{ij}) & i = j \\ 0 \end{cases} \tag{37}$$

for every element i, j in the derivative matrix.

1.4.1.4 Training

To make this explanation easier, I will use a fully connected neural network. Neural network training or backpropagation can be mathematically expressed as minimizing a loss function $\ell(Y_{pred}, Y_{true})$ describing how inaccurate the network is. In our case, ℓ 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. $mse : \mathbb{R}^\mu \times \mathbb{R}^\mu \to \mathbb{R}$ is defined as:

$$\ell(Y_{pred}, Y_{true}) = \frac{|Y_{pred} - Y_{true}|^2}{\mu} \tag{38}$$

The network then performs gradient descent to find parameters that minimize ℓ. Let $\ell' : \mathbb{R}^\mu \times \mathbb{R}^\mu \to \mathbb{R}^\mu$ be the derivative of ℓ.

$$\ell'(Y_{pred}, Y_{true}) = \frac{2}{\mu} (Y_{pred} - Y_{true}) \tag{39}$$

Given a neural netowrk consisting of n layers, training will start at the last layer and probagate thought the network from back to front. Hence the name backpropagation. To change the values of the parameters of every layer in the network of the network, the algorythem must fist determine the in which direction and by how much, the variables should be moved. Let $j \in \mathbb{N} \cup [0, n-1[$ be the index of the layer. The indeces start at the first layer $(j = 0)$ and count upwards. The layer at any particualer index j is f_j. For

the last layer $j = n - 1$, the change to its output ΔY_j is described as follows using (39):

$$\Delta Y_j = \ell'(Y_{pred}, Y_{true}) \tag{40}$$

In Generall f_j is always followed by an activation function f_{a_j}. Let A_j be the precomputed inputs to f_{a_j}. Let ΔA_j be the desired for change to A_j.

$$\Delta A_j = \langle s, f'_a(A_j) \rangle_E \Delta Y_j \tag{41}$$

Where $s(x) = \frac{1}{x}$. Next comes the update to the weight matrix w_j. Let Δw_j describe the change to w_j and let X_j be the input vector of the layer. Δw_j is than defined as:

$$\Delta w_j = \Delta A_j \cdot X_j^T \tag{42}$$

The layer's bias vector is updated in the direction of ΔA_j:

$$\Delta b_j = \Delta A_j \tag{43}$$

Lastly, the change to the output of the previous layer ΔY_{j-1} is computed.

$$\Delta Y_{j-1} = \Delta A_a \cdot w_j^T \tag{44}$$

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 $i_n = [X, Y] \in \mathbb{R}^{7\times6\times2}$. Consider the following Connect4 board (fig 5 on page 25).

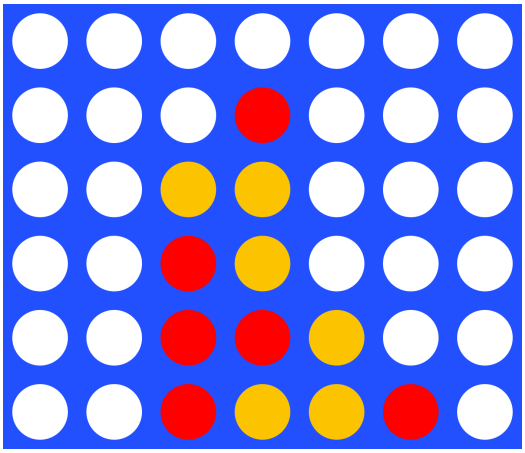


Figure 5

If red is the current player then:

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

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

For clarification, the numbers are coloured in the same colour as the stones

at that position. After stacking X and Y, i_n is:

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

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]
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]
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 [3]
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 throught the softmax function to form π:

$$\pi = \text{softmax}(p_{masked}) \tag{45}$$

1.4.2.3 Training

Training is performed in batches of 256 states. Both heads are updated useing *mse* or Mean Squared Error. In the policy network all non-legal actions are ignored. This avoids unnecessary updating of the neural network. The value of the value head, 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:

$$p_{a_l} = \frac{N(n, a_l)}{\sum_{a \in \mathbb{A}(n)} N(n, a)} \tag{46}$$

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

$$p_{a_n} = p_{pre_{a_n}} \tag{47}$$

Where p_{pre} is defined in section 1.4.2.2 on page 26.

1.5 Training loop

The Alpha zero training loop consists of data generation, training and testing. During data generation, data is generated, that is than used to train the neural network. After every training session, the new model is evaluated against the old one, to determin if it is better. If this is the case, the new model is used to generate date, if not the old one generates data gain and may be replaced after the next round.

1.5.1 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.

1.5.1.1 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 $a = argmax(N(\mathcal{E}(n_t, a \in \mathbb{A}(n_t))))$ 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:

$$\mathcal{P}(X = a_i, n_t) = \frac{N(\mathcal{E}(n_t, a_i))}{\sum_{j \in \mathbb{A}(n_t)} N(\mathcal{E}(n_t, j))} \tag{48}$$

($\mathbb{A}(s)$ 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 loops data generation phase, actions are always selected deterministically.

1.5.1.2 Memory

The memory is used to store game states, after they where generated, but befor they are used to train the mode. It 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:

$$v_a = \begin{cases} \mathcal{P}(X = a, \mathcal{N}(g)) & a \in \mathbb{A}(g) \\ p_{pre_a} \end{cases} \tag{49}$$

$\mathbb{A}(g)$ is the set of all legal actions. p_{pre} is defined in section 1.4.2.2 on page 26, and is used for all non legal actions. P is defined in equation 48 on page 28.

1.5.1.3 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 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 on page 27.

1.5.3 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]

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 [5]:

$$E = \frac{1}{1 + e^{(r_b - r_a)/400}} \tag{50}$$

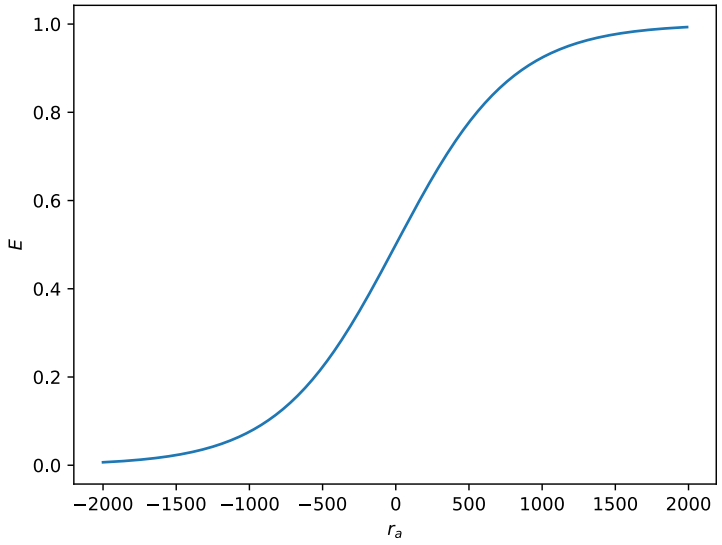


Figure 6: elo-rating win probability for $r_b = 0$

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, a's elo, rating is updated as follows[2]:

$$r_{a_{n+1}} = r_{a_n} + K(W - E) \tag{51}$$

with:

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

r_{a_n} the current rating of the player.

$W = s_a$ which is defined by equation 52 where a is the player to be updated.

E the expected chance of winning, see equation 50.

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 [2]:

$$s_a = \frac{1}{m} \sum \begin{cases} 1 & a \text{ wins} \\ \frac{1}{2} & \text{tie} \\ 0 & a \text{ looses} \end{cases} \tag{52}$$

Assuming that this is the probability of a winning against b, a's elo-rating can be computed by solving equation 50 to r_a (fig 7 on page 32):

$$r_a = r_b - ln\left(\frac{1 - s_a}{s_a}\right) \cdot 400 \tag{53}$$

see section 10.3 on page 246 for how to get from (50) to (53) Since a ranking of all the agents already exists (see section 1.5.3 on page 29), an agent's elo-rating can be computed by playing against an older version and then using equation 53 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.

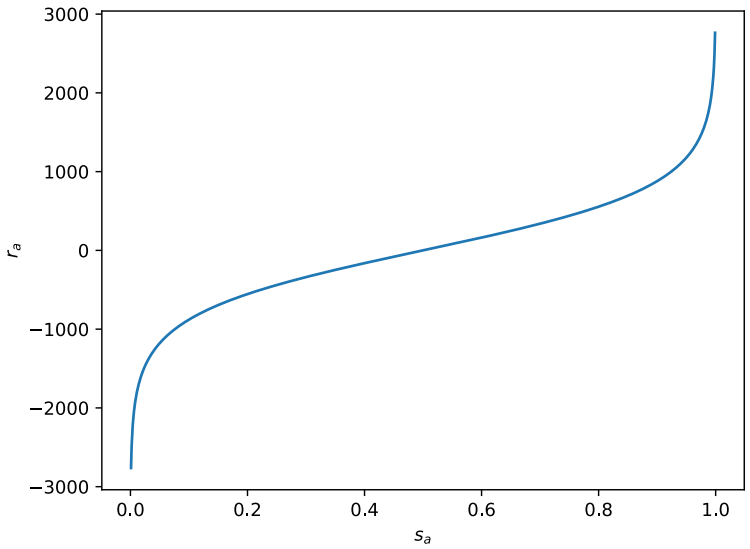


Figure 7: elo inverse function for $r_b = 0$

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:

$$\Delta r_{min} = -ln\left(\frac{1 - \Delta s_i}{\Delta s_i}\right) \cdot 400 \tag{54}$$

Where Δs_i is the minimal expected chance of agent i beating agent $i-1$. 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_i and s_{i-1} are the scores of agents i and $i-1$, that play aginst each other, then by definition:

$$s_i + s_{i-1} = 1 \tag{55}$$

Let w_i be the win count of agent i over game count.
Let d_i be the draw count over game count.
Let l_i be the loose count of agent i over game count.

Than:

$$s_i = w_i + \frac{d_i}{2} \tag{56}$$

$$s_{i-1} = l_i + \frac{d_i}{2} \tag{57}$$

Than due to θ:

$$w_i \geqslant l_i \theta \tag{58}$$

And by extension:

$$s_i \geqslant s_{i-1} \tag{59}$$

Under the assumption, that there are no ties:

$$s_i \geqslant s_{i-1} \cdot \theta \tag{60}$$

Let s_{min_i} be the minimal possible s_i (assuming no ties). Using (60) and (55):

$$\left| \begin{aligned} s_i + s_{i-1} &= 1 \\ s_i &= s_{i-1}\theta \end{aligned} \right| \tag{61}$$

$$\implies s_i = \frac{\theta}{1+\theta} \tag{62}$$

For $\theta = 1.3$ this means that the expected average change in rating Δr using (62) and (54):

$$\Delta r \geqslant -ln\left(\frac{1}{\theta}\right) \cdot 400 = \Delta r_{min} \cong 105 \tag{63}$$

Collected data shows this to be true (fig 8 on page 34). The same data shows that the average Δr is in fact roughly 408, which would equate to a θ of

$$\theta = \frac{1}{e^{\frac{-\Delta r}{400}}} \cong 2.8 \tag{64}$$

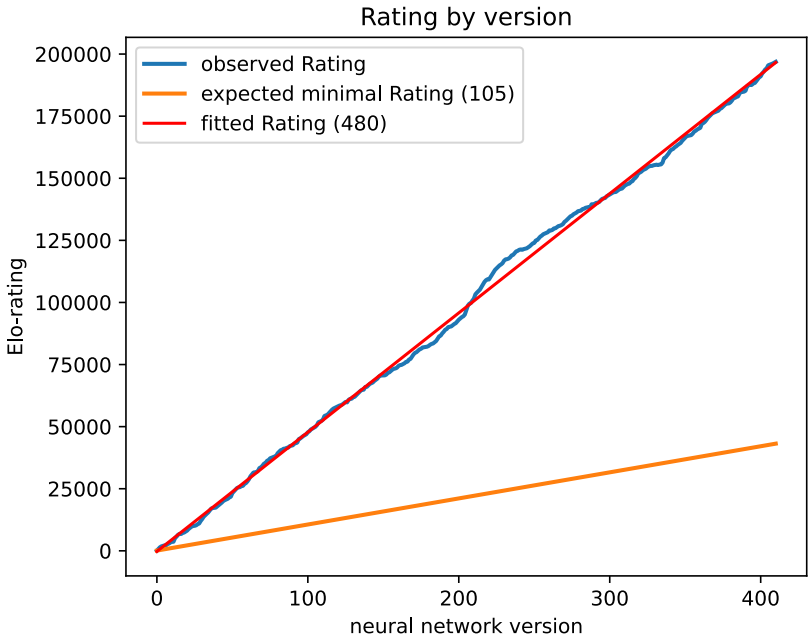


Figure 8: Elo-rating of agents based on their version along with the expected minimal rating Δr_{min} and the best fitted rating Δr.

3 Problem Complexity

Lastly let us look at the complexity of the game connect4. Complexity in this case referes to the amount of possibly game states in the game.

3.1 Trivial Answers

There are two trivial ways of putting an upper bound on the amount of game states, the game has. The first is too simply look at the number of possible ways to place three differant kinds of stones onto a board with 42 locations. This means that there can not be more game states than 3^{42}. Another way is to realize is that it must be less than 42!.

3.2 Combinatorics

Let $a(n) : \mathbb{N} \to \mathbb{N}^+$ be the amount of placements of n stones of colors given by the connect4 rules in a 6×7 board:

$$a(n) = \binom{42}{\lceil \frac{n}{2} \rceil} \binom{42 - \lceil \frac{n}{2} \rceil}{\lfloor \frac{n}{2} \rfloor} \tag{65}$$

The Total amount of possible game states n_s must be less than:

$$n_s < \sum_{n=0}^{42} a(n) = 16'282'402'094'173'127'445 \tag{66}$$

3.3 SBFS based state search

According to calculations made with BDD's, the amount of state in connect4 is $4'531'985'219'092$.[4]

4 Implementation

The Alpha Zero training loop, can be found in "train" function found in section 6.1.11.7.11 on page 99. This function loads the latest neural network and memory if available. It uses then perform data generation by using the "playGames_InThreads" function to have the Ai play a certain amount of games against itself. This is done by distributing the computations over multiple threads, each thread has its own MCTS and temporary memory. Once they are done generating data, the global memory is updated. The "playGames_InThreads" function creates and manages the threads running "playGames". This latter function simulates a certain amount of games betwean two agents (section 6.1.11.7.1 on page 66). A game simmulation consists creating an instance of the Game (section 6.1.11.8.1 on page 111) class, that handles all the game logic, then queriing sed game for what player is a trun, then using the agensts "getAction" to determin the next move and get the data used for training (see section 1.5.1.1 on page 28), which is passed to temporary memory (46), lastly the action is used to move to the next game state, by using the games "takeAction" function. This loop, of getting the agent to sugens an action, and then using the game to take that action is repeated, untill the game reaches an end game state. At that point, the

temporary memory is updated with the correct rewards in accordance with section 1.5.1.3 on page 29.

5 Conclusion

In conclusion the algorythem works, and is able to learn to play connect4 at a level better than any human I could find. Some improvements can still be made, but in generall, Im rather happy with the results.

5.1 Improvements

- The algorythem could be compleatly rewritten to allow for taining on TPU's.
- Process distribution could be improved allow for training of the AI on multiple GPU's.
- It it is possible to remove the need for model testing.
- The algorythem could be tested for differant games, and other configurations.
- Most of the time is spent on running neural network simulations. By maximizing information gane from every neural network simulation. A possible way to do this would be to not only update MCTS Graphs to the current node, but all the way back to the root. Furthermore by not only updateing the path but every possible pass back would increase efficency.

References

[1] Guillaume MJ-B Chaslot, Mark HM Winands, and HJVD Herik. Parallel monte-carlo tree search. In *International Conference on Computers and Games*, pages 60–71. Springer, 2008.

[2] Arpad E. Elo. *The Rating of Chessplayers, Past and Present*. Arco Pub., New York, 1978.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[4] Peter Kissmann. https://web.archive.org/web/20131002122656/http://www.tzi.de/~edelkamp/lectures/ae/slide/AE-SymbolischeSuche.pdf.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[6] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[7] Wikipedia. Vector quantization — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Vector%20quantization&oldid=1082226300, 2022. [Online; accessed 06-June-2022].

[8] Wikipedia contributors. Vectorization (mathematics) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Vectorization_(mathematics)&oldid=1057421500, 2021. [Online; accessed 17-March-2022].

[9] Zhihua Zhang. Artificial neural network. In *Multivariate time series analysis in climate and environmental research*, pages 1–35. Springer, 2018.

Acknowledgements

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

6 Code

6.1 AlphaZeroPytorch

6.1.1 AlphaZeroPytorch.h

```
// AlphaZeroPytorch.h : Include file for standard system include files,
// or project specific include files.

#pragma once

#include <iostream>

// TODO: Reference additional headers your program requires here.
```

6.1.2 AlphaZeroPytorch.cpp

```
// AlphaZeroPytorch.cpp : Defines the entry point for the application.
//

#include "AlphaZeroPytorch.h"
#include <ai/playGame.hpp>
#include <io.hpp>
#include <chrono>
#include <thread>
#include "makeFiles.hpp"


int main(int argc, char ** argv)
{
  /*std::ofstream out("out.txt");
  std::streambuf* coutbuf = std::cout.rdbuf(); //save old buf
  std::cout.rdbuf(out.rdbuf()); //redirect std::cout to out.txt!
   */

  if (torch::cuda::cudnn_is_available())
  {
    std::cout << "\33[1;32mcuDNN is available\33[0m" << std::endl;
  }
  else if (torch::hasXLA())
```

```
  {
    std::cout << "\33[1;32mXLa is available\33[0m" << std::endl;
  }
  else
  {
    std::cout << "\33[1;31mWarning: cuDNN is unavailable,
    consider using a CUDA enabled GPU\33[0m" << std::endl;
  }
  std::vector<char*> devices = { DEVICES };
  for (auto const& device : devices)
  {
    std::cout << device << ", ";
  }

  std::cout << std::endl << "started training" << std::endl;
  createFolders();
  AlphaZero::ai::train(-1);
#if ProfileLogger
  debug::Profiler::profiler.log();
#endif
  return 0;
}
```

6.1.3 CMakeLists.txt

```
# CMakeList.txt : CMake project for AlphaZeroPytorch, include
    source and define
# project specific logic here.
#
project(AlphaZero)

set (CMAKE_CXX_STANDARD 17)
cmake_minimum_required (VERSION 3.8)
set(CMAKE_BUILD_TYPE Debug)

message(STATUS "searching for Pytorch ...")
find_package(Torch REQUIRED)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${TORCH_CXX_FLAGS}")

message(STATUS "adding source files ...")
file(GLOB SOURCES_Files
  "include/*.cpp"
  "include/ai/*.cpp"
  "include/game/*.cpp"
)
file(GLOB HEADER_Files
```

```
  "include/*.hpp"
  "include/ai/*.hpp"
  "include/game/*.hpp"
)
file(GLOB SERVER_Files
  "include/Server/*.cpp"
)

file(GLOB OUTER_Files
  "include/*.cpp"
)

file(GLOB GAME_Files
  "include/game/*.cpp"
)

file(GLOB TEST_SOURCE
  "include/test/*.cpp"
  "include/ai/*.cpp"
  "include/game/*.cpp"
  "include/*.cpp"
)

find_library(spdlog_location spdlog "/" REQUIRED)
find_library(sockpp_location sockpp "/" REQUIRED)


# Add source to this project's executable.
add_executable (train "AlphaZeroPytorch.cpp" "AlphaZeroPytorch.h" ${SOURCES_Files} ${HEADER_Files})
add_executable (GameReplayer "Replay.cpp" "include/game/game.cpp" "include/config.cpp")
add_executable (runTest "test.cpp" ${TEST_SOURCE} ${SOURCES_Files})
add_executable (runServer "runServer.cpp" ${SOURCES_Files} ${SERVER_Files})
add_executable (convert "convertToJceFormat.cpp" "include/ai/modelWorker.cpp" "${OUTER_Files}")
add_executable (eloRaiting "doEloRaiting.cpp" ${SOURCES_Files} ${SERVER_Files})

target_compile_definitions(train PRIVATE cpuct_=2.0f)
target_compile_definitions(GameReplayer PRIVATE cpuct_=2.0f)
target_compile_definitions(runTest PRIVATE cpuct_=2.0f)
target_compile_definitions(runServer PRIVATE cpuct_=1.0f)
```

```
target_compile_definitions(convert PRIVATE cpuct_=2.0f)
target_compile_definitions(eloRaiting PRIVATE cpuct_=2.0f)


message(STATUS "linking libs")

message(STATUS "sockpp is at: ${sockpp_location} ${spdlog_location}")
target_link_libraries(train         PRIVATE "${TORCH_LIBRARIES}" ${spdlog_location})
target_link_libraries(runServer     PRIVATE "${TORCH_LIBRARIES}" ${spdlog_location} ${sockpp_location})
target_link_libraries(eloRaiting    PRIVATE "${TORCH_LIBRARIES}" ${spdlog_location} ${sockpp_location})
target_link_libraries(GameReplayer PRIVATE "${TORCH_LIBRARIES}" ${spdlog_location})
target_link_libraries(convert       PRIVATE "${TORCH_LIBRARIES}" ${spdlog_location})
target_link_libraries(runTest     PRIVATE "${TORCH_LIBRARIES}" ${spdlog_location})


target_include_directories(train PUBLIC "include/")
target_include_directories(runServer PUBLIC "include/")
target_include_directories(GameReplayer PUBLIC "include/")
target_include_directories(runTest PUBLIC "include/")
target_include_directories(convert PUBLIC "include/")
target_include_directories(eloRaiting PUBLIC "include/")


message(STATUS "done")
```

6.1.4 convertToJceFormat.cpp

```
#include <iostream>
#include <ai/model.hpp>
#include "makeFiles.hpp"

#define FILENAME "model.jce.bin"

int main(int argc, char** argv)
{
  createFolders();
  int version=-2;
  std::cout << "What version do you want to convert -1 for current, -2 for inverse conversion: ";
```

```
12    std::cin >> version;
13
14
15    AlphaZero::ai::Model model("cpu");
16
17    if (version >= 0)
18    {
19      model.load_version(version);
20      model.jce_save_current(FILENAME);
21    }
22    else if (version == -1)
23    {
24      model.load_current();
25      model.jce_save_current(FILENAME);
26    }
27    else if (version = -2)
28    {
29      model.jce_load_from_file(FILENAME);
30      model.save_as_current();
31    }
32    else
33    {
34      return -1;
35    }
36
37    return 1;
38  }
```

6.1.5 doEloRaiting.cpp

```
1  #include <Server/eloClient.hpp>
2  #include <ai/playGame.hpp>
3  #include <math.h>
4
5  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
       )
6  {
7    std::cout << "evaluating elo for: " << agent << std::endl;
8
9    int othersElo = elo.getElo(agent - 1);
10   std::cout << "others elo is: " << othersElo << std::endl;
11   AlphaZero::Game::Game* game = new AlphaZero::Game::Game();
12   AlphaZero::ai::Memory* memory = new AlphaZero::ai::Memory();
13
```

```
14    if (agent -1 > 0)
15    {
16      lastAgent->model->load_version(agent - 1);
17    }
18    if (agent > 0)
19    {
20      currentAgent->model->load_version(agent);
21    }
22    auto data = AlphaZero::ai::playGames_inThreads(game, lastAgent
        .get(), currentAgent.get(), memory, 2, 1, games, "
        eloEvaluation");
23
24    int wins = data[currentAgent.get()];
25    int losses = data[lastAgent.get()];
26    int ties = games - wins - losses;
27
28    float score = ((float)wins + 0.5f * (float)ties)/((float)games
        );
29    if (score > 0.99)
30    {
31      score = 0.99f;
32    }
33    float Relo = (float)othersElo - log((1 - score) / score) *
        400;
34    elo.setElo(agent, (int)Relo);
35
36    std::cout << wins << " wins, " << ties << " ties and " <<
        losses << " losses" << std::endl;
37    std::cout << "win Ratio is : " << score << std::endl;
38    std::cout << "new rating is: " << Relo << std::endl << std::
        endl;
39
40    delete game;
41    delete memory;
42  }
43
44  int main()
45  {
46    std::vector<char*> devices = { DEVICES };
47    std::shared_ptr<AlphaZero::ai::Agent> lastAgent = std::
        make_shared<AlphaZero::ai::Agent>(devices);
48    std::shared_ptr<AlphaZero::ai::Agent> currentAgent = std::
        make_shared<AlphaZero::ai::Agent>(devices);
49
50    AlphaZero::elo::eloClient elo;
```

```
51    std::cout << elo.setElo(0, 100) << std::endl;
52
53    int agent = 409;
54    while (true)
55    {
56      evaluateAgent(agent, 40, elo, lastAgent, currentAgent);
57      agent++;
58    }
59    return 1;
60  }
```

6.1.6 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  }
```

6.1.7 Replay.cpp

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

6.1.8 runServer.cpp

```
1  #include <config.hpp>
2  #include <Server/server.hpp>
3  #include <game/game.hpp>
4
5
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
        for server if not testing\33[0m" << std::endl;
15    AlphaZero::Server::TestServer server(PORT);
16    server.mainLoop();
17 #endif
18 }
```

6.1.9 showLoss.cpp

```
1  #include <iostream>
2  #include <log.hpp>
3  #include <python.h>
4
5  int main(int argc, char** argv)
6  {
7    debug::log::lossLogger log("logs/games/loss.bin");
8
9    return 1;
10 }
```

6.1.10 test.cpp

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

6.1.11 include

6.1.11.1 config.hpp

```
#pragma once
#include <log.hpp>
#include <bitset>
#include <mutex>

#ifdef unix
#define UNIX
#endif

#define DEVICES "cuda:0"

#define OPSMode 1

extern std::mutex console_mutex;
extern std::mutex rand_mutex;

/*
+---------+-------------+
| OPSMode | Description |
+---------+-------------+
| 1       | Run Server  |
+---------+-------------+
| 2       | Run Tester  |
+---------+-------------+
*/

#define GameChecksLegalMoved true // the game will check if a move is legal not neded for training
#define stateSize 84
#define Training true
#define DEBUG false

#define U_computation(edge) (this->cpuct * edge.P * std::sqrt((float)Nb) / (float)(1 + edge.N))


// runn setting
#define runVersion 1
#define loadVersion -1

// Net settings
```

```
#define MaxQuDefault -99999
#define reg_const 0.0001
#define learningRage 0.1
#define Momentum 0.9

// simulation setting
#define MCTSSimulations 50
//#define cpuct_ 2.0f
#define ProbabiliticMoves 10
#define Alpha 0.9
#define EPSILON 0.2f

// memory setting
#define memory_size 30000

// self play
#define EPOCHS 1
#define GEN_THREADS 60
#define probabilitic_moves 10 // how manny moves are prabilistic
    in the begining of the game to aid in exploration

// training
#define Training_loops 20
#define Training_batch 256
#define Training_epochs 5

// turney
#define Turnement_probabiliticMoves 2
#define TurneyEpochs 1
#define TurneyThreads 20
#define scoringThreshold 1.3

// console
#define RenderTrainingProgress false
#define RenderGenAndTurneyProgress false

// Saving
#define SaverType 0
/*
    +-----------+------------------------------------------------------------

  | SaverType | Description
                              |

    +-----------+------------------------------------------------------------
```

```
80    | 0          | no Saver
                                |
81
      +------------+-------------------------------------------------------------------------------
82    | 1          | save full state to file
                                |
83
      +------------+-------------------------------------------------------------------------------
84    | 2          | Save taken Actions to file (int size is saved
      size of the int in bytes) |
85
      +------------+-------------------------------------------------------------------------------
      */
86  #define SaverIntSize 1
87
88
89  typedef std::bitset<stateSize> IDType;
```

6.1.11.2 config.cpp

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

6.1.11.3 io.hpp

```
1   #pragma once
2   #include <jce/load.hpp>
3   #include <jce/save.hpp>
4   #include "config.hpp"
5
6
7   namespace AlphaZero
8   {
9       namespace io
10      {
11          namespace FullState
12          {
13              class GameSaver
14              {
15              public: std::list<std::list<std::shared_ptr<Game::
      GameState>>> states;
```

```
16              public: void addState(std::shared_ptr<Game::
    GameState> state);
17              public: void addGame();
18              public: void clear();
19              public: void save(char filename[]);
20              public: void load(char filename[]);
21
22              // replay game in console (debug)
23              public: void ConsoleReplay(int itx);
24              };
25          }
26          namespace ActionsOnly
27          {
28              class GameSaver
29              {
30              public: std::list<std::list<unsigned int>> states;
31              public: void addState(int);
32              public: void addGame();
33              public: void clear();
34              public: void save(char filename[]);
35              public: void load(char filename[]);
36
37              // replay game in console (debug)
38              public: void ConsoleReplay(int idx);
39              };
40          }
41      }
42  }
43
44  inline void AlphaZero::io::FullState::GameSaver::addState(std::
    shared_ptr<Game::GameState> state)
45  {
46      this->states.back().push_back(state);
47  }
48
49  inline void AlphaZero::io::FullState::GameSaver::addGame()
50  {
51      this->states.push_back(std::list<std::shared_ptr<Game::
    GameState>>());
52  }
53
54  inline void AlphaZero::io::FullState::GameSaver::clear()
55  {
56      this->states.clear();
57  }
```

```
58
59 inline void AlphaZero::io::FullState::GameSaver::save(char
       filename[])
60 {
61     std::ofstream fout;
62     fout.open(filename, std::ios::binary);
63     if (fout.is_open())
64     {
65         jce::save(fout, this->states);
66         fout.close();
67     }
68     else
69     {
70         throw "Game saver file not opend.";
71     }
72 }
73
74 inline void AlphaZero::io::FullState::GameSaver::load(char
       filename[])
75 {
76     std::ifstream infile(filename, std::ios::binary);
77
78     if (infile.is_open())
79     {
80         jce::load(infile, this->states);
81         infile.close();
82     }
83     else
84     {
85     throw "Game saver file not opend.";
86     }
87 }
88
89 inline void AlphaZero::io::FullState::GameSaver::ConsoleReplay(
       int idx)
90 {
91     for (auto const& state : *std::next(this->states.begin(),
       idx))
92     {
93         state->render();
94     }
95 }
96
97
98 inline void AlphaZero::io::ActionsOnly::GameSaver::addState(int
```

```
    actions)
{
    this->states.back().push_back(actions);
}

inline void AlphaZero::io::ActionsOnly::GameSaver::addGame()
{
    this->states.push_back(std::list<unsigned int>());
}

inline void AlphaZero::io::ActionsOnly::GameSaver::clear()
{
    this->states.clear();
}

inline void AlphaZero::io::ActionsOnly::GameSaver::save(char
    filename[])
{
    std::ofstream file(filename, std::ios::binary);
    if (file.is_open())
    {
        jce::save(file, this->states);
        file.close();
    }
    else
    {
        throw "Game saver file not opend.";
    }
}

inline void AlphaZero::io::ActionsOnly::GameSaver::load(char
    filename[])
{
    std::ifstream file(filename, std::ios::binary);
    if (file.is_open())
    {
        jce::load(file, this->states);
        file.close();
    }
    else
    {
        throw "Game saver file not opend.";
    }
}

```

```
141 inline void AlphaZero::io::ActionsOnly::GameSaver::ConsoleReplay
    (int idx)
142 {
143     Game::Game game = Game::Game();
144 
145     for (int action : *std::next(this->states.begin(), idx))
146     {
147         game.render();
148         game.takeAction(action);
149     }
150     game.render();
151 }
```

6.1.11.4 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 
24 
25 namespace debug {
26 #if ProfileLogger
27   namespace Profiler {
28     class MCTSProfiler {
29     private: utils::Timer timer;
30     public: std::unordered_map<unsigned int, double> times;
```

```
31      private: double rest;
32
33      private: bool first = true;
34      private: bool toRest = false;
35      private: unsigned int currentTime;
36
37      public: void switchOperation(unsigned int id);
38      public: void stop();
39      public: void log();
40      public: void log(std::shared_ptr<spdlog::logger> logger);
41      };
42      extern debug::Profiler::MCTSProfiler profiler;
43    };
44  #endif
45    namespace log {
46      std::shared_ptr<spdlog::logger> createLogger(const char*
       name, const char* file);
47      template<typename T>
48      void logVector(std::shared_ptr<spdlog::logger> logger, std::
       vector<T>, char out[]);
49      void logVector(std::shared_ptr<spdlog::logger> logger, std::
       vector<int>);
50      void logVector(std::shared_ptr<spdlog::logger> logger, std::
       vector<float>);
51
52      class lossLogger
53      {
54      public: lossLogger();
55      public: lossLogger(const char file[]);
56
57      protected: std::vector<std::vector<std::pair<float, float>>>
        vals;
58
59      public: void addValue(const float val, const float poly);
60      public: void addValue(const std::pair<float, float>& val);
61      public: void newBatch();
62      public: void save(const char file[]);
63
64      public: std::pair<float, float> operator[](std::pair<size_t,
        size_t> idx) const;
65      public: std::vector<std::pair<float, float>> operator[](
       size_t idx) const;
66      public: bool operator==(const lossLogger&);
67      };
68
```

```
69 #if MainLogger
70     extern std::shared_ptr<spdlog::logger> mainLogger;
71 #endif
72 #if MCTSLogger
73     extern std::shared_ptr<spdlog::logger> MCTS_Logger;
74 #endif
75 #if MemoryLogger
76     extern std::shared_ptr<spdlog::logger> memoryLogger;
77 #endif
78 #if ProfileLogger
79     extern std::shared_ptr<spdlog::logger> profileLogger;
80 #endif
81 #if ModelLogger
82     extern std::shared_ptr<spdlog::logger> modelLogger;
83 #endif
84 #if LossLogger
85     extern lossLogger _lossLogger;
86 #endif
87   }
88 }
89
90 inline std::shared_ptr<spdlog::logger> debug::log::createLogger(
       const char* name, const char* file) {
91   // Create a daily logger - a new file is created every day on
       2:30am
92   int success = remove(file);
93   auto logger = spdlog::basic_logger_mt(name, file);
94   return logger;
95 }
96
97 inline debug::log::lossLogger::lossLogger()
98 {
99   this->newBatch();
100 }
101
102 inline void debug::log::lossLogger::addValue(const float a,
       const float b)
103 {
104   std::pair<float, float> data = { a, b };
105   this->addValue(data);
106 }
107
108 inline void debug::log::lossLogger::addValue(const std::pair<
       float, float>& val)
109 {
```

```
110   this->vals.back().push_back(val);
111 }
112
113 inline void debug::log::lossLogger::newBatch()
114 {
115   this->vals.push_back({});
116 }
117
118 inline std::pair<float, float> debug::log::lossLogger::operator
      [](std::pair<size_t, size_t> idx) const
119 {
120   return this->vals[idx.first][idx.second];
121 }
122
123 inline std::vector<std::pair<float, float>> debug::log::
      lossLogger::operator[](size_t idx) const
124 {
125   return this->vals[idx];
126 }
127
128 template<typename T>
129 inline void debug::log::logVector(std::shared_ptr<spdlog::logger
      > logger, std::vector<T> vec, char out[])
130 {
131   logger->info(out, (vec[0]), (vec[1]), (vec[2]), (vec[3]),
       (vec[4]), (vec[5]), (vec[6]));
132   logger->info(out, (vec[7]), (vec[8]), (vec[9]), (vec[10]),
       (vec[11]), (vec[12]), (vec[13]));
133   logger->info(out, (vec[14]), (vec[15]), (vec[16]), (vec[17]),
       (vec[18]), (vec[19]), (vec[20]));
134   logger->info(out, (vec[21]), (vec[22]), (vec[23]), (vec[24]),
       (vec[25]), (vec[26]), (vec[27]));
135   logger->info(out, (vec[28]), (vec[29]), (vec[30]), (vec[31]),
       (vec[32]), (vec[33]), (vec[34]));
136   logger->info(out, (vec[35]), (vec[36]), (vec[37]), (vec[38]),
       (vec[39]), (vec[40]), (vec[41]));
137 }
138
139 inline void debug::log::logVector(std::shared_ptr<spdlog::logger
      > logger, std::vector<int> vec)
140 {
141   char out[] = "Action vals are: {:3d}, {:3d}, {:3d}, {:3d}, {:3
       d}, {:3d}, {:3d}";
142   logVector(logger, vec, out);
143 }
```

```
144
145 inline void debug::log::logVector(std::shared_ptr<spdlog::logger
    > logger, std::vector<float> vec)
146 {
147   char out[] = "Action vals are: {:1.2f}, {:1.2f}, {:1.2f},
     {:1.2f}, {:1.2f}, {:1.2f}, {:1.2f}";
148   logVector(logger, vec, out);
149 }
150 #if ProfileLogger
151 inline void debug::Profiler::MCTSProfiler::switchOperation(
    unsigned int id)
152 {
153   this->stop();
154   this->first = false;
155   this->toRest = false;
156   this->timer.reset();
157   this->currentTime = id;
158 }
159
160 inline void debug::Profiler::MCTSProfiler::stop()
161 {
162
163   if (!this->first) {
164     if (toRest) {
165       this->rest = this->rest + this->timer.elapsed();
166     }
167     double currentNum;
168     if (this->times.count(currentTime) == 0) {
169       currentNum = 0.0f;
170     }
171     else {
172       currentNum = this->times.at(this->currentTime);
173     }
174     double res = currentNum + this->timer.elapsed();
175     this->times.insert_or_assign( currentTime, res);
176   }
177   this->toRest = true;
178 }
179
180 inline void debug::Profiler::MCTSProfiler::log() {
181   this->log(debug::log::profileLogger);
182 }
183
184 inline void debug::Profiler::MCTSProfiler::log(std::shared_ptr<
    spdlog::logger> logger)
```

```
185 {
186   logger->info("using {} threads", threads);
187   logger->info("run Info:");
188   logger->info("");
189
190   // this is only true on the computers im using as im just
        entering the infomation here to save time
191 #ifdef WIN32
192   logger->info("os: Windows 10");
193   logger->info("CPU: Intel(R) Core(TM) i5-8350U CPU @ 1.70 GHz "
        );
194   logger->info("GPU: None");
195   logger->info("memory: 8 GB");
196 #else
197   logger->info("OS: Ubuntu 18.04");
198   logger->info("CPU: ??");
199   logger->info("GPU: Nvidia P4 cuda 11.4");
200   logger->info("memory: 7.8 Gb");
201 #endif
202
203   logger->info("
        ==========================================================
        ");
204   for (auto const& pair : this->times) {
205     logger->info("Profiler time id {} took {} s", pair.first,
        pair.second);
206   }
207   logger->info("everything else took: {}", this->rest);
208
209   logger->info("");
210   logger->info("0 : MCTS and NN forward");
211   logger->info("3 : Game Stuff");
212   logger->info("4 : Memory shuffeling");
213   logger->info("5 : NN Backward");
214 }
215 #endif
216 #endif
```

6.1.11.5 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::MCTS_Logger = debug
       ::log::createLogger("MCTS_Logger", "logs/c++/MCTS_Logger.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::MCTSProfiler debug::Profiler::profiler = debug
       ::Profiler::MCTSProfiler();
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 " <<
       file << "\33[0m" << std::endl;
39   }
```

```
40    in.close();
41  }
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 " <<
      file << "\33[0m" << std::endl;
53    }
54    out.close();
55  }
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            if (other[idx] != (*this)[idx])
68            {
69              return false;
70            }
71          }
72        }
73        else
74        {
75          return false;
76        }
77      }
78      return true;
79    }
80    return false;
81  }
```

6.1.11.6 timer.hpp

```
#pragma once
#include <iostream>
#include <chrono>

namespace utils {
    class Timer
    {
    public:
        Timer();
        void reset();
        double elapsed() const;

    private:
        typedef std::chrono::high_resolution_clock clock_;
        typedef std::chrono::duration<double, std::ratio<1> >
    second_;
        std::chrono::time_point<clock_> beg_;
    };
};


inline utils::Timer::Timer()
{
    this->beg_ = clock_::now();
}

inline void utils::Timer::reset()
{
    this->beg_ = clock_::now();
}

inline double utils::Timer::elapsed() const
{
        return std::chrono::duration_cast<second_>(clock_::now()
    - beg_).count();
}
```

6.1.11.7 ai

6.1.11.7.1 agent.hpp

```
#pragma once
#include <ai/modelSynchronizer.hpp>
```

```
#include <ai/memory.hpp>
#include <jce/vector.hpp>
#include "utils.hpp"
#include <thread>

namespace AlphaZero {
  namespace ai {
    class Agent {
    public: std::unordered_map<size_t, std::shared_ptr<MCTS>> tree;
    public: std::unique_ptr<AlphaZero::ai::ModelSynchronizer> model;
    public: Agent(std::vector<char*> devices);
    public: int identity;
    public: MCTS* getTree();
    public: void reset();
#if Training
    public: std::pair<int, std::pair<std::vector<int>, float>> getAction(std::shared_ptr<Game::GameState> state, bool proabilistic);
#else
    public: virtual std::pair<int, std::vector<float>> getAction(std::shared_ptr<Game::GameState> state, bool proabilistic);
#endif
    public: void runSimulations(Node*, MCTS* tree);
    private: float evaluateLeaf(Node*, MCTS* tree);
    public: void fit(std::shared_ptr<Memory> memory, unsigned short iteration);
    public: std::pair<float, std::vector<float>> predict(std::shared_ptr<Game::GameState> state);
    public: void predict(ModelData* data);
    public: void predict(std::list<ModelData*> data);
    private: std::pair<int, std::pair<std::vector<int>, float>> derministicAction(Node* node);
    private: std::pair<int, std::pair<std::vector<int>, float>> prabilisticAction(Node* node);
    };
#if not Training
    class User : public Agent {
    public: virtual std::pair<int, std::vector<float>> getAction(std::shared_ptr<Game::Game> game, bool proabilistic);
    };
#endif
    void runSimulationsCaller(AlphaZero::ai::Agent* agent, Node* node, MCTS* tree);
```

```
37    }
38  }
39
40  inline AlphaZero::ai::MCTS* AlphaZero::ai::Agent::getTree()
41  {
42    auto a = std::hash<std::thread::id>{}(std::this_thread::get_id
        ());
43    if (this->tree.count(a))
44    {
45      return this->tree[a].get();
46    }
47    else
48    {
49      auto tree = std::make_shared<MCTS>();
50      this->tree.insert({ a, tree });
51      return tree.get();
52    }
53  }
54
55  inline void AlphaZero::ai::Agent::reset()
56  {
57    this->tree.clear();
58  }
59
60  inline void AlphaZero::ai::Agent::runSimulations(Node* node,
        MCTS* tree)
61  {
62    std::pair<Node*, std::list<Edge*>> serchResults = tree->
        moveToLeaf(node);
63    float val = this->evaluateLeaf(serchResults.first, tree);
64    tree->backFill(serchResults.second, serchResults.first, val);
65    tree->addMCTSIter();
66  }
67
68  inline void AlphaZero::ai::runSimulationsCaller(AlphaZero::ai::
        Agent* agent, Node* node, MCTS* tree)
69  {
70    while (tree->MCTSIter < MCTSSimulations) {
71      agent->runSimulations(node, tree);
72    }
73  }
74
75  inline float AlphaZero::ai::Agent::evaluateLeaf(Node* node, MCTS
        * tree)
76  {
```

```
77    if (!node->state->done){
78      std::shared_ptr<Game::GameState> nextState;
79      Node* nextNode;
80      auto data = ModelData(node);
81      this->predict(&data);
82      for (auto& action : node->state->allowedActions) {
83        nextState = node->state->takeAction(action);
84        nextNode = tree->addNode(nextState);
85        Edge newEdge = Edge(nextNode, node, action, data.polys[
      action].item<float>()); //the last is the prob
86        node->addEdge( action, newEdge);
87      }
88      return data.value;
89    }
90    return (float)std::get<0>(node->state->val);
91  }
92
93  inline void AlphaZero::ai::Agent::fit(std::shared_ptr<Memory>
      memory, unsigned short run)
94  {
95    std::cout << "\33[35;1mretraining\33[0m" << std::endl;
96    for (int idx = 0; idx < Training_loops ; idx++) {
97  #if RenderTrainingProgress
98      jce::consoleUtils::render_progress_bar((float)idx / (float)
      Training_loops);
99  #endif
100     auto batch = Model::getBatch(memory, Training_batch);
101     for (size_t trainingEpoch = 0; trainingEpoch <
      Training_epochs; trainingEpoch++)
102       this->model->fit(batch, run, idx);
103
104 #if LossLogger
105     debug::log::_lossLogger.newBatch();
106 #endif
107   }
108   this->model->synchronizeModels();
109 #if LossLogger
110   debug::log::_lossLogger.save("logs/games/loss.bin");
111 #endif
112 #if RenderTrainingProgress
113   jce::consoleUtils::render_progress_bar(1.0f, true);
114 #endif
115 }
116
117 inline std::pair<float, std::vector<float>> AlphaZero::ai::Agent
```

```
    ::predict(std::shared_ptr<Game::GameState> state)
{
  auto preds = this->model->predict(state);
  float& val = preds.first;
  std::vector<float> polys = std::vector<float>(action_count);

  c10::Device device ("cpu");

  torch::Tensor mask = torch::ones(
    { 1, action_count },
    c10::TensorOptions().device(c10::Device("cpu")).dtype(at::
    kBool)
  );

  for (auto idx : state->allowedActions)
  {
    mask[0][idx] = false;
  }

  torch::Tensor out = torch::softmax(torch::masked_fill(preds.
    second.cpu(), mask, -1000.0f), 1);

  for (auto const& idx : state->allowedActions) {
    polys[idx] = out[0][idx].item<float>();
  }

  return { val, polys };
}

inline void AlphaZero::ai::Agent::predict(ModelData* data)
{
  this->model->addData(data);
}

inline void AlphaZero::ai::Agent::predict(std::list<ModelData*>
    data)
{
  this->model->predict(data);
}

inline std::pair<int, std::pair<std::vector<int>, float>>
    AlphaZero::ai::Agent::derministicAction(Node* node)
{
  int action = 0;
  unsigned int max_N = 0;
```

```
158   unsigned int sum = 0;
159   std::vector<int> probs = jce::vector::gen(action_count, 0);
160   for (auto const& iter : node->edges) {
161     if (iter.second.N > max_N) {
162       max_N = iter.second.N;
163       action = iter.first;
164     }
165     probs[iter.second.action] = iter.second.N;
166   }
167   return { action, { probs, node->edges[action].Q } };
168 }
169
170 inline std::pair<int, std::pair<std::vector<int>, float>>
      AlphaZero::ai::Agent::prabilisticAction(Node* node)
171 {
172   int action = -1;
173   int idx = 0;
174   unsigned int sum = 0;
175   std::vector<int> probs = jce::vector::gen(action_count, 0);
176
177   for (auto const& iter : node->edges) {
178     probs[iter.second.action] = (iter.second.N);
179   }
180   auto action_probs = (rand() % ai::getSumm(probs));
181
182   for (auto const& val : probs) {
183     action_probs -= val;
184     if (action_probs < 0) {
185       action = idx;
186       break;
187     }
188     idx++;
189   }
190   return { action, { probs, node->edges[action].Q } };
191 }
```

6.1.11.7.2 agent.cpp

```
1 #include "agent.hpp"
2 #include <stdlib.h>
3
4 #if not Training
5 std::pair<int, std::vector<float>> AlphaZero::ai::User::
      getAction(std::shared_ptr<Game::Game> game, bool proabilistic
      )
6 {
```

```
7     std::system("cls");
8     game->render();
9     int action = -1;
10    while (std::find(game->state->allowedActions.begin(), game->
        state->allowedActions.end(), action) == game->state->
        allowedActions.end()) {// while(not game->state->
        allowedActions.contains(action))
11      std::cout << "your Action: ";
12      std::cin >> action;
13    }
14    return { action, std::vector<float>() };
15  }
16  #endif
17
18  AlphaZero::ai::Agent::Agent(std::vector<char*> devices)
19  {
20    this->tree = {};
21    this->model = std::make_unique<ModelSynchronizer>(devices);
22  }
23
24  std::pair<int, std::pair<std::vector<int>, float>> AlphaZero::ai
        ::Agent::getAction(std::shared_ptr<Game::GameState> state,
        bool proabilistic)
25  {
26  #if ProfileLogger
27    debug::Profiler::profiler.switchOperation(0);
28  #endif
29    auto tree = this->getTree();
30    tree->MCTSIter = 0;
31    Node* node = tree->addNode(state);
32  #if threads > 0
33    std::vector<std::thread> threadvec;
34    for (int i = 0; i < threads; i++) {
35      threadvec.push_back(std::thread(runSimulationsCaller, this,
        node));
36    }
37  #endif
38    runSimulationsCaller(this, node, tree);
39  #if threads > 0
40    for (auto& thread : threadvec) {
41      thread.join();
42    }
43  #endif
44    try {
45  #if ProfileLogger
```

```
46      debug::Profiler::profiler.switchOperation(3);
47  #endif
48      if (proabilistic) {
49        return this->prabilisticAction(node);
50      }
51      else {
52        return this->derministicAction(node);
53      }
54    }
55    catch (const std::exception& ex) {
56      std::cerr << "\33[31;1mError in Agent::getAction\33[0m" <<
        std::endl;
57      std::cerr << ex.what() << std::endl;
58      throw ex;
59    }
60  }
```

6.1.11.7.3 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 Represienting the action connecting 2 nodes togather
        . It handles all The MCTS relevant variables and the mutex
        for
14      * parallization
15      */
16      class Edge {
17        // The number of times the Edge was traversed
18      public: int N = 0;
19          // the probability initialized by the NN
20      public: float P = 0;
21          // the action asociated with the action
22      public: int action = 0;
23          // the amount of times this lead to a win
24      public: float W = 0;
25          // the win probability
```

```
26      public: float Q = 0;
27      public: Node* outNode;
28      public: Node* inNode;
29      public: Edge(Node* outNode, Node* inNode, int action, float
        p);
30      public: Edge();
31      public: void traverse();
32      };
33
34      class Node {
35        // mutex locking the during insersion of edges and also
        used as the child edges mutex
36      //public: std::mutex lock;
37      public: std::shared_ptr<Game::GameState> state;
38      public: std::unordered_map<int, Edge> edges;
39      public: Node(std::shared_ptr<Game::GameState>);
40      public: bool isLeaf();
41      public: void addEdge(int id, Edge& edge);
42      };
43
44      std::vector<float> getQ(Node*);
45
46      class MCTS {
47        // mutex keeping corrupion within the Tree from occuring
        when new nodes are added
48      // public: std::mutex NodeInsersionMutex;
49        // mutex keeping corrupion of the MCTSIter variable
50      // public: std::mutex MCTSIterMutex;
51      public: unsigned short MCTSIter = 0;
52      private: std::unordered_map<IDType, std::unique_ptr<Node>>
        MCTS_tree;
53
54          // add 1 to MCTSIter within a mutex
55      public: void addMCTSIter();
56      public: MCTS();
57      public: float cpuct = cpuct_;
58      public: std::pair <Node*, std::list<Edge*>> moveToLeaf(Node
        *);
59      public: void backFill(std::list<Edge*>&, Node* leaf, float
        val);
60      public: Node* getNode(IDType);
61      public: Node* addNode(std::shared_ptr<Game::GameState> state
        );
62      public: void reset();
63      };
```

```
64    }
65  }
66
67  inline std::vector<float> AlphaZero::ai::getQ(Node* node)
68  {
69    std::vector<float> data = jce::vector::gen(42, 0.0f);
70    for (auto const& pos : node->edges)
71    {
72      data[pos.first] = pos.second.Q;
73    }
74    return data;
75  }
76
77  inline void AlphaZero::ai::MCTS::addMCTSIter()
78  {
79    // this->MCTSIterMutex.lock();
80    MCTSIter++;
81    // this->MCTSIterMutex.unlock();
82  }
83
84  inline bool AlphaZero::ai::Node::isLeaf()
85  {
86    return this->edges.size() == 0;
87  }
88
89  inline void AlphaZero::ai::Node::addEdge(int id, Edge& edge)
90  {
91    // this->lock.lock();
92    this->edges.insert({ id, edge });
93    // this->lock.unlock();
94  }
95
96  inline AlphaZero::ai::MCTS::MCTS(){}
97
98  inline AlphaZero::ai::Node* AlphaZero::ai::MCTS::addNode(std::
      shared_ptr<Game::GameState> state)
99  {
100   if (this->MCTS_tree.count(state->id()) == 0) {
101
102     // this->NodeInsersionMutex.lock();
103     this->MCTS_tree.insert({ state->id(), std::make_unique<Node
      >(state)});
104     // this->NodeInsersionMutex.unlock();
105   }
106   return this->getNode(state->id());
```

```
}

inline void AlphaZero::ai::MCTS::reset()
{
  // this->NodeInsersionMutex.lock();
  this->MCTS_tree.clear();
  // this->NodeInsersionMutex.unlock();
}

inline AlphaZero::ai::Node* AlphaZero::ai::MCTS::getNode(IDType
    key)
{
  return this->MCTS_tree[key].get();
}


inline void AlphaZero::ai::Edge::traverse()
{
  // this->inNode->lock.lock();
  this->N++;
  // this->inNode->lock.unlock();
}
```

6.1.11.7.4 MCTS.cpp

```
#include "MCTS.hpp"
#include <limits>
#include <jce/vector.hpp>

AlphaZero::ai::Node::Node(std::shared_ptr<Game::GameState> state
    )
{
  this->state = state;
}

AlphaZero::ai::Edge::Edge(Node* _outNode, Node* _inNode, int
    _action, float _p)
{
  this->P = _p;
  this->action = _action;
  this->outNode = _outNode;
  this->inNode = _inNode;
  this->N = 0;
  this->W = 0;
  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*>>
       AlphaZero::ai::MCTS::moveToLeaf(Node* node)
28 {
29   std::list<Edge*> backTrackList;
30   while (true) {
31     if (node->isLeaf()) {
32       return { node, backTrackList };
33     }
34     else {
35       float U;
36       int Nb = 0;
37       for (auto const& iter : node->edges)
38       {
39         Nb += iter.second.N;
40       }
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::MCTS::backFill(std::list<Edge*>& backTrace,
       Node* leaf, float val)
65 {
66   float currentPlayer = (float)leaf->state->player * val;
67
68   for (auto const& edge : backTrace) {
69     // edge->inNode->lock.lock();
70
71     edge->W = edge->W + currentPlayer * (float)edge->inNode->
       state->player;
72     edge->Q = edge->W / (float)edge->N;
73
74     // edge->inNode->lock.unlock();
75   }
76 }
```

6.1.11.7.5 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 MemoryElement {
15     public: int value;
16     public: std::shared_ptr<Game::GameState>state;
17     public: std::vector<float> av;
18     public: MemoryElement(std::shared_ptr<Game::GameState>, std
       ::vector<int>);
19     public: MemoryElement();
20     };
21     class TemporaryMemory
22     {
23     public: bool active;
24     public: TemporaryMemory(bool);
25     public: std::vector<std::shared_ptr<MemoryElement>>
       tempMemory;
```

```
26      public: void commit(std::shared_ptr<Game::GameState>, std::
        vector<int>&);
27      };
28      class Memory {
29        //keep memory from doing annything
30      private: std::mutex mu;
31      public: bool active = true;
32      public: TemporaryMemory getTempMemory();
33      public: std::vector<std::shared_ptr<MemoryElement>> memory;
34      public: Memory();
35      public: void updateMemory(int player, int value,
        TemporaryMemory* memory);
36      public: std::shared_ptr<MemoryElement> getState();
37      public: void save(int version);
38      public: void save();
39      public: void save(char filename[]);
40      public: void load(int version);
41      public: void load();
42      public: void load(char filename[]);
43      public: void render();
44      private: void updateMemory(int val, TemporaryMemory* memory)
        ;
45      };
46    }
47  }
48
49  inline AlphaZero::ai::TemporaryMemory::TemporaryMemory(bool val)
50  {
51    this->active = val;
52  }
53
54  inline AlphaZero::ai::TemporaryMemory AlphaZero::ai::Memory::
        getTempMemory()
55  {
56    AlphaZero::ai::TemporaryMemory memory(this->active);
57    return memory;
58  }
59
60  inline AlphaZero::ai::MemoryElement::MemoryElement(std::
        shared_ptr<Game::GameState> _state, std::vector<int> _av)
61  {
62    this->state = _state;
63    float sum = (float)getSumm(_av);
64    this->av = jce::vector::gen(_av.size(), 0.0f);
65    for (size_t idx = 0; idx < this->av.size(); idx++)
```

```
66    {
67      float tmp = ((float)_av[idx]);
68      this->av[idx] = tmp / sum;
69    }
70  }
71  inline AlphaZero::ai::MemoryElement::MemoryElement()
72  {
73  }
74  inline AlphaZero::ai::Memory::Memory()
75  {
76  }
77  inline void AlphaZero::ai::TemporaryMemory::commit(std::
      shared_ptr<Game::GameState> state, std::vector<int>& av)
78  {
79    if (this->active) {
80      std::vector<std::pair<std::shared_ptr<Game::GameState>, std
      ::vector<int>>> idents = Game::identities(state, av);
81      for (auto const& data : idents) {
82        tempMemory.push_back(std::make_shared<MemoryElement>(data.
      first, data.second));
83      }
84    }
85  }
86
87  inline void AlphaZero::ai::Memory::updateMemory(int player, int
      value, TemporaryMemory* memory)
88  {
89    this->updateMemory(player * value, memory);
90  }
91
92  inline std::shared_ptr<AlphaZero::ai::MemoryElement> AlphaZero::
      ai::Memory::getState()
93  {
94    unsigned long long idx = rand() % this->memory.size();
95    std::shared_ptr<MemoryElement> element = this->memory[idx];
96    this->memory.erase(this->memory.begin() + idx);
97    return element;
98  }
99
100 inline void AlphaZero::ai::Memory::render()
101 {
102   for (auto const& element : this->memory)
103   {
104     element->state->render();
105   }
```

```
106 }
107
108 inline void AlphaZero::ai::Memory::updateMemory(int val,
      TemporaryMemory* memory)
109 {
110   while (memory->tempMemory.size() > 0) {
111     std::shared_ptr<MemoryElement>& element = memory->tempMemory
      .back();
112     element->value = element->state->player * val;
113     this->mu.lock();
114     this->memory.push_back(element);
115     this->mu.unlock();
116     memory->tempMemory.pop_back();
117   }
118 }
```

6.1.11.7.6 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 opend.";
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 opend.";
70    }
71  }
```

6.1.11.7.7 model.hpp

```
// TorchTestCMake.h : Include file for standard system include files,
// or project specific include files.

#pragma once

#include <iostream>
#include <game/game.hpp>
#include "memory.hpp"
#include <string>
#include <tuple>
#include <jce/string.hpp>
#include <string>
#include <cmath>
#include "modelWorker.hpp"
#include <jce/save.hpp>
#include <jce/load.hpp>
#include <test/testUtils.hpp>


namespace AlphaZero {
  namespace ai {
    class TopLayer : public torch::nn::Module {
    public: torch::nn::Conv2d conv1;
    public: torch::nn::LayerNorm batch;
    public: torch::nn::LeakyReLU relu;
    private: int kernel1;

    public: TopLayer(int inp, int out, int kernelsize1);
    public: torch::Tensor forward(torch::Tensor);
    public: void moveTo(c10::Device device);
    };
    class ResNet : public torch::nn::Module {
    public: torch::nn::Conv2d conv1, conv2;
    public: torch::nn::LayerNorm batch, batch2;
    public: torch::nn::LeakyReLU activ;
    private: int kernel1, kernel2;

    public: ResNet(int inp, int out, int kernelsize1, int
    kernelsize2);
    public: torch::Tensor forward(torch::Tensor);
    public: void moveTo(c10::Device device);
    };

    class Value_head : torch::nn::Module {
```

```
44      private: bool isSecondRun = false;
45      private: torch::Tensor tmpX;
46
47      public: torch::nn::Conv2d conv;
48      public: torch::nn::Linear lin1, lin2;
49      public: torch::nn::LeakyReLU relu;
50      public: torch::nn::Tanh tanh;
51      private: int size;
52
53      public: Value_head(int inp, int hidden_size, int out, int
        kernels);
54      public: torch::Tensor forward(torch::Tensor);
55      public: void moveTo(c10::Device device);
56      };
57
58      class Policy_head : torch::nn::Module {
59      public: torch::nn::Conv2d conv;
60      public: torch::nn::Linear lin1;
61      public: torch::nn::LeakyReLU relu;
62      private: int size;
63
64      public: Policy_head(int inp, int hidden, int out);
65      public: torch::Tensor forward(torch::Tensor);
66      public: void moveTo(c10::Device device);
67      };
68
69      typedef torch::nn::MSELoss Loss;
70      typedef torch::optim::SGD Optimizer;
71      typedef torch::optim::SGDOptions OptimizerOptions;
72
73      class Model : public torch::nn::Module {
74        //private: torch::nn::Conv2d headLayer;
75      private: TopLayer top;
76      private: ResNet res1, res2, res3, res4, res5, res6;
77      private: Value_head value_head;
78      private: Policy_head policy_head;
79
80      private: char* device;
81
82      private: Loss loss;
83      private: Optimizer optim;
84
85      public: Model(char* device);
86      public: std::pair<torch::Tensor, torch::Tensor> forward(
        torch::Tensor);
```

```
87      public: std::pair<float, float> train(const std::pair<torch
        ::Tensor, torch::Tensor>& x, const std::pair<torch::Tensor,
        torch::Tensor>& y);
88
89      public: std::pair<float, torch::Tensor>predict(std::
        shared_ptr<Game::GameState> state);
90      public: static std::tuple<torch::Tensor, torch::Tensor,
        torch::Tensor> getBatch(std::shared_ptr<Memory> memory,
        unsigned int batchSize);
91      public: void predict(ModelData* data);
92      public: void predict(std::list<ModelData*> data);
93      public: void fit(const std::tuple<torch::Tensor, torch::
        Tensor, torch::Tensor>& batch, const unsigned short& run,
        const unsigned short& trainingLoop);
94
95      public: void save_version(unsigned int version);
96      public: void save_as_current();
97      public: void save_to_file(char* filename);
98      public: void jce_save_current(char* filename);
99
100     public: void load_version(unsigned int version);
101     public: void load_current();
102     public: void load_from_file(char* filename);
103     public: void jce_load_from_file(char* filename);
104
105     public: void copyModel(Model*);
106     private: void copyParameters(torch::OrderedDict<std::string,
        torch::Tensor> prams);
107     public: void moveTo(c10::Device device);
108
109     private: TopLayer register_custom_module(TopLayer net);
110     private: ResNet register_custom_module(ResNet net, std::
        string layer);
111     private: Value_head register_custom_module(Value_head net);
112     private: Policy_head register_custom_module(Policy_head net)
        ;
113
114     };
115   }
116 }
117 // customizable section
118 #define modelTest false
119 #define randomModel false
120 #define convSize 5
121
```

```
122 inline AlphaZero::ai::Model::Model(char* _device) :
123   top(this->register_custom_module(TopLayer(2, 75, convSize))),
124   res1(this->register_custom_module(ResNet(75, 75, convSize,
        convSize), "Residual_1")),
125   res2(this->register_custom_module(ResNet(75, 75, convSize,
        convSize), "Residual_2")),
126   res3(this->register_custom_module(ResNet(75, 75, convSize,
        convSize), "Residual_3")),
127   res4(this->register_custom_module(ResNet(75, 75, convSize,
        convSize), "Residual_4")),
128   res5(this->register_custom_module(ResNet(75, 75, convSize,
        convSize), "Residual_5")),
129   res6(this->register_custom_module(ResNet(75, 75, convSize,
        convSize), "Residual_6")),
130   value_head(this->register_custom_module(Value_head(75, 420,
        210, 10))),
131   policy_head(this->register_custom_module(Policy_head(75, 84,
        42))),
132   optim(Optimizer(this->parameters(), OptimizerOptions(
        learningRage).momentum(Momentum))),
133   device(_device)
134 {
135   this->moveTo(c10::Device(_device));
136 }
137
138 inline std::pair<torch::Tensor, torch::Tensor> AlphaZero::ai::
      Model::forward(torch::Tensor x)
139 {
140 #if randomModel
141   return { torch::rand({x.size(0), 1}), torch::rand({x.size(0),
        action_count}) };
142 #else
143   x = this->top.forward(x);
144   x = this->res1.forward(x);
145   x = this->res2.forward(x);
146   x = this->res3.forward(x);
147   x = this->res4.forward(x);
148   x = this->res5.forward(x);
149   x = this->res6.forward(x);
150
151   // compute individual heads
152   torch::Tensor value = this->value_head.forward(x.clone());
153   torch::Tensor poly = this->policy_head.forward(x.clone());
154
155   return { value, poly };
```

```
156 #endif
157 };
158 // end of cutimizable section
159
160 inline AlphaZero::ai::TopLayer::TopLayer(int inp, int out, int
      kernelsize1) :
161   conv1(this->register_module("conv1", torch::nn::Conv2d(torch::
      nn::Conv2dOptions(inp, out, kernelsize1)))),
162   batch(this->register_module("batch", torch::nn::LayerNorm(
      torch::nn::LayerNormOptions({ out, input_shape_y,
      input_shape_x })))),
163   relu(this->register_module("ReLU", torch::nn::LeakyReLU(torch
      ::nn::LeakyReLU()))),
164   kernel1(kernelsize1 / 2)
165 {
166 }
167 inline torch::Tensor AlphaZero::ai::TopLayer::forward(torch::
      Tensor x)
168 {
169   x = torch::nn::functional::pad(x, torch::nn::functional::
      PadFuncOptions({ kernel1, kernel1, kernel1, kernel1 }));
170   x = this->conv1(x);
171   x = this->batch(x);
172   x = this->relu(x);
173   return x;
174 }
175
176 inline void AlphaZero::ai::TopLayer::moveTo(c10::Device device)
177 {
178   this->conv1->to(device, true);
179   this->batch->to(device, true);
180   this->relu->to(device, true);
181 }
182
183 inline AlphaZero::ai::ResNet::ResNet(int inp, int out, int
      kernelsize1, int kernelsize2) :
184   kernel1(kernelsize1), kernel2(kernelsize2),
185   conv1(this->register_module("conv1", torch::nn::Conv2d(torch::
      nn::Conv2dOptions(inp, out, kernelsize1)))),
186   conv2(this->register_module("conv2", torch::nn::Conv2d(torch::
      nn::Conv2dOptions(out, out, kernelsize2)))),
187   batch(this->register_module("batch1", torch::nn::LayerNorm(
      torch::nn::LayerNormOptions({ out, input_shape_y,
      input_shape_x })))),
188   batch2(this->register_module("batch2", torch::nn::LayerNorm(
```

```
    torch::nn::LayerNormOptions({ out, input_shape_y,
    input_shape_x }))))),
189   activ(this->register_module("activ", torch::nn::LeakyReLU(
    torch::nn::LeakyReLU())))
190 {
191   if (torch::cuda::is_available()) {
192     this->moveTo(c10::Device("cuda:0"));
193   }
194 }
195
196 inline torch::Tensor AlphaZero::ai::ResNet::forward(torch::
    Tensor x)
197 {
198 #if modelTest
199   std::cout << x.sizes() << std::endl;
200 #endif
201   auto y = x.clone();
202   x = torch::nn::functional::pad(x, torch::nn::functional::
    PadFuncOptions({ kernel1 / 2, kernel1 / 2, kernel1 / 2,
    kernel1 / 2 }));
203   x = this->conv1(x);
204   x = this->batch(x);
205   x = this->activ(x);
206
207   x = torch::nn::functional::pad(x, torch::nn::functional::
    PadFuncOptions({ kernel2 / 2, kernel2 / 2, kernel2 / 2,
    kernel2 / 2 }));
208   x = this->conv2(x);
209   x = this->batch2(x);
210   return this->activ(x + y);
211 }
212
213 inline void AlphaZero::ai::ResNet::moveTo(c10::Device device)
214 {
215   this->conv1->to(device, true);
216   this->conv2->to(device, true);
217   this->batch->to(device, true);
218   this->batch2->to(device, true);
219   this->activ->to(device, true);
220 }
221
222 inline AlphaZero::ai::Value_head::Value_head(int inp, int
    hidden_size, int out, int convOut) :
223   conv(this->register_module("conv", torch::nn::Conv2d(torch::nn
    ::Conv2dOptions(inp, convOut, 1)))),
```

```
224   lin1(this->register_module("lin1", torch::nn::Linear(torch::nn
        ::LinearOptions(hidden_size, out)))),
225   lin2(this->register_module("lin2", torch::nn::Linear(torch::nn
        ::LinearOptions(out, 1)))),
226   relu(this->register_module("relu", torch::nn::LeakyReLU())),
227   tanh(this->register_module("tanh", torch::nn::Tanh()))
228 {
229   this->size = hidden_size;
230
231   if (torch::cuda::is_available())
232   {
233     this->moveTo(c10::Device("cuda:0"));
234   }
235 }
236
237 inline torch::Tensor AlphaZero::ai::Value_head::forward(torch::
        Tensor x)
238 {
239 #if modelTest
240   std::cout << "value" << std::endl;
241   std::cout << x.sizes() << std::endl;
242 #endif
243   x = this->conv(x);
244   x = this->relu(x);
245   x = this->lin1(x.reshape({ x.size(0), this->size }));
246   x = this->relu(x);
247   x = this->lin2(x);
248   x = this->tanh(x);
249   return x;
250 }
251
252 inline void AlphaZero::ai::Value_head::moveTo(c10::Device device
        )
253 {
254   this->conv->to(device, true);
255   this->lin1->to(device, true);
256   this->relu->to(device, true);
257   this->lin2->to(device, true);
258   this->tanh->to(device, true);
259 }
260
261 inline AlphaZero::ai::Policy_head::Policy_head(int inp, int
        hidden, int out) :
262   conv(this->register_module("conv", torch::nn::Conv2d(torch::nn
        ::Conv2dOptions(inp, 2, 1)))),
```

```
263   lin1(this->register_module("lin1", torch::nn::Linear(hidden,
        out))),
264   relu(this->register_module("relu", torch::nn::LeakyReLU()))
265 {
266   this->size = hidden;
267
268   if (torch::cuda::is_available()) {
269     this->moveTo(c10::Device("cuda:0"));
270   }
271 }
272
273 inline torch::Tensor AlphaZero::ai::Policy_head::forward(torch::
      Tensor x)
274 {
275 #if modelTest
276   std::cout << "poly" << std::endl;
277   std::cout << x.sizes() << std::endl;
278 #endif
279   x = this->conv(x);
280   x = this->relu(x);
281   x = this->lin1(x.reshape({ x.size(0), this->size }));
282   return x;
283 }
284
285 inline void AlphaZero::ai::Policy_head::moveTo(c10::Device
      device)
286 {
287   this->conv->to(device, true);
288   this->lin1->to(device, true);
289 }
290
291 inline torch::Tensor polyLoss(torch::Tensor a, torch::Tensor b)
292 {
293 #if true
294   auto c = torch::where(b == 0, a, b);
295   return torch::mse_loss(a, c);
296 #else
297   return torch::mse_loss(a, b);
298 #endif
299 }
300
301 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)
302 {
```

```
303   //std::cout << x.first << std::endl << y.first << std::endl;
304
305   auto valLoss = 0.5f * torch::mse_loss(x.first, y.first);
306   auto plyLoss = 0.5f * polyLoss(x.second, y.second);
307   auto loss = (valLoss + plyLoss);
308
309   loss.backward();
310   this->optim.step();
311   this->optim.zero_grad();
312   std::pair<float, float> error = { torch::mean(valLoss).item().
        toFloat(), torch::mean(plyLoss).item().toFloat() };
313
314   if (std::isnan(error.first))
315   {
316     //std::cout << valLoss << std::endl << plyLoss << std::endl;
317     std::cout << x.first  << std::endl << y.first  << std::endl;
318     std::cout << x.second << std::endl << y.second << std::endl;
319     return error;
320   }
321
322   return error;
323 }
324
325 inline std::pair<float, torch::Tensor> AlphaZero::ai::Model::
        predict(std::shared_ptr<Game::GameState> state)
326 {
327   torch::Tensor NNInput = state->toTensor().to(c10::Device(this
        ->device));
328   std::pair<torch::Tensor, torch::Tensor> NNOut = this->forward(
        NNInput);
329   float value = NNOut.first[0].item<float>();
330   return { value, NNOut.second };
331 }
332
333 inline void AlphaZero::ai::Model::predict(ModelData* data)
334 {
335   torch::Tensor NNInput = data->node->state->toTensor().to(c10::
        Device(this->device));
336   std::pair<torch::Tensor, torch::Tensor> NNOut = this->forward(
        NNInput);
337
338   torch::Tensor mask = torch::ones(
339     { 1, action_count },
340     c10::TensorOptions().device(c10::Device("cpu")).dtype(at::
        kBool)
```

```
341     );
342
343     for (auto idx : data->node->state->allowedActions)
344     {
345       mask[0][idx] = false;
346     }
347     //std::cout << std::endl << NNOut.first << std::endl << NNOut.
        second << std::endl;
348     data->value = NNOut.first[0].item<float>();
349     data->polys = torch::softmax(torch::masked_fill(NNOut.second.
        cpu(), mask, -1000.0f), 1)[0];
350 }
351
352 inline void AlphaZero::ai::Model::predict(std::list<ModelData*>
        data)
353 {
354     torch::Tensor NNInput = torch::zeros({ (int)data.size(),
        input_snape_z, input_shape_y, input_shape_x });
355     torch::Tensor mask = torch::ones(
356       { (int)data.size(), action_count },
357       c10::TensorOptions().device(c10::Device("cpu")).dtype(at::
        kBool)
358     );
359     auto iter = data.begin();
360     for (unsigned short idx = 0; idx < data.size(); idx++)
361     {
362       NNInput[idx] = (*iter)->node->state->toTensor()[0];
363       for (auto action : (*iter)->node->state->allowedActions)
364       {
365         mask[idx][action] = false;
366       }
367       iter++;
368     }
369
370     std::pair<torch::Tensor, torch::Tensor> NNOut = this->forward(
        NNInput.to(c10::Device(this->device)));
371
372     //std::cout << std::endl << NNOut.first << std::endl << NNOut.
        second << std::endl;
373     mask = mask.to(c10::Device(this->device));
374
375     auto soft = torch::softmax(torch::masked_fill(NNOut.second,
        mask, -1000.0f), 1).cpu();
376
377     iter = data.begin();
```

```
378   for (unsigned int idx = 0; idx < data.size(); idx++)
379   {
380     (*iter)->value = NNOut.first[idx].item<float>();
381     (*iter)->polys = soft[idx];
382     iter++;
383   }
384 }
385
386 inline std::tuple<torch::Tensor, torch::Tensor, torch::Tensor>
      AlphaZero::ai::Model::getBatch(std::shared_ptr<Memory> memory
      , unsigned int batchSize)
387 {
388   std::tuple<torch::Tensor, torch::Tensor, torch::Tensor> output
      =
389   {
390     at::zeros({batchSize, input_snape_z, input_shape_y,
      input_shape_x}),
391     at::zeros({batchSize, action_count}),
392     at::zeros({batchSize, 1})
393   };
394   for (unsigned short idx = 0; idx < batchSize; idx++) {
395     auto state = memory->getState();
396     state->state->toTensor(std::get<0>(output), idx);
397     std::get<1>(output)[idx] = at::from_blob(state->av.data(), {
      action_count }).toType(torch::kFloat16);
398     std::get<2>(output)[idx] = torch::tensor({ state->value });
399   }
400   return output;
401 }
402
403 inline void AlphaZero::ai::Model::fit(const std::tuple<torch::
      Tensor, torch::Tensor, torch::Tensor>& batch, const unsigned
      short& run, const unsigned short& trainingLoop)
404 {
405   std::pair<torch::Tensor, torch::Tensor> NNVals = this->forward
      (std::get<0>(batch).to(c10::Device(this->device)));
406   std::pair<float, float> error = this->train(NNVals,
407     {
408     std::get<2>(batch).to(c10::Device(this->device)),
409     std::get<1>(batch).to(c10::Device(this->device))
410     });
411 #if ModelLogger
412   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));
```

```
413 #endif
414 #if LossLogger
415   debug::log::_lossLogger.addValue(error);
416 #endif
417 }
418
419 inline void AlphaZero::ai::Model::save_version(unsigned int
      version)
420 {
421   char buffer[50];
422   std::sprintf(buffer, "models/run_%d/V_%d.torch", runVersion,
      version);
423   std::cout << buffer << std::endl;
424   this->save_to_file(buffer);
425 }
426
427 inline void AlphaZero::ai::Model::save_as_current()
428 {
429   char buffer[50];
430   std::sprintf(buffer, "models/run_%d/currentModel.torch",
      runVersion);
431   this->save_to_file(buffer);
432 }
433
434 inline void AlphaZero::ai::Model::save_to_file(char* filename)
435 {
436   torch::serialize::OutputArchive out;
437   this->save(out);
438   std::string model_path = std::string(filename);
439   out.save_to(model_path);
440 }
441
442 inline void AlphaZero::ai::Model::jce_save_current(char*
      filename)
443 {
444   std::ofstream out(filename, std::ios::binary);
445   jce::save(out, this->named_parameters(true));
446   out.close();
447 }
448
449
450 inline void AlphaZero::ai::Model::load_version(unsigned int
      version)
451 {
452   std::cout << "loading ...";
```

```
453   char buffer[50];
454   std::sprintf(buffer, "models/run_%d/V_%d.torch", runVersion,
        version);
455   this->load_from_file(buffer);
456   std::cout << " loaded Version " << version << std::endl;
457 }
458
459 inline void AlphaZero::ai::Model::load_current()
460 {
461   char buffer[50];
462   std::sprintf(buffer, "models/run_%d/currentModel.torch",
        runVersion);
463   this->load_from_file(buffer);
464 }
465
466 inline void AlphaZero::ai::Model::load_from_file(char* filename)
467 {
468   torch::serialize::InputArchive inp;
469   std::string model_path = std::string(filename);
470   inp.load_from(model_path);
471   this->load(inp);
472 }
473
474 inline void AlphaZero::ai::Model::jce_load_from_file(char*
        filename)
475 {
476   std::cout << "loading ...\t";
477   torch::autograd::GradMode::set_enabled(false);
478   torch::OrderedDict<std::string, torch::Tensor> map;
479   std::ifstream in(filename, std::ios::binary);
480   if (in.is_open())
481   {
482     jce::load(in, map);
483     this->copyParameters(map);
484   }
485   test::printSuccess(in.is_open());
486   in.close();
487   torch::autograd::GradMode::set_enabled(true);
488 }
489
490 inline void AlphaZero::ai::Model::copyModel(AlphaZero::ai::Model
        * model)
491 {
492   torch::autograd::GradMode::set_enabled(false);
493   auto new_params = model->named_parameters(true);
```

```
494   this->copyParameters(new_params);
495   torch::autograd::GradMode::set_enabled(true);
496 }
497
498 inline void AlphaZero::ai::Model::copyParameters(torch::
      OrderedDict<std::string, torch::Tensor> new_params)
499 {
500   auto params = this->named_parameters(true);
501   auto buffers = this->named_buffers(true);
502   for (auto& val : new_params) {
503     auto name = val.key();
504     auto* t = params.find(name);
505     if (t != nullptr) {
506       t->copy_(val.value());
507     }
508     else {
509       t = buffers.find(name);
510       if (t != nullptr) {
511         t->copy_(val.value());
512       }
513     }
514   }
515 }
516
517 inline void AlphaZero::ai::Model::moveTo(c10::Device device)
518 {
519   this->top.moveTo(device);
520
521   this->res1.moveTo(device);
522   this->res2.moveTo(device);
523   this->res3.moveTo(device);
524   this->res4.moveTo(device);
525   this->res5.moveTo(device);
526   this->res6.moveTo(device);
527
528   this->value_head.moveTo(device);
529   this->policy_head.moveTo(device);
530 }
531
532 inline AlphaZero::ai::TopLayer AlphaZero::ai::Model::
      register_custom_module(TopLayer net)
533 {
534   register_module("TopLayer_conv", net.conv1);
535   register_module("TopLayer_batch", net.batch);
536   register_module("TopLayer_ReLU", net.relu);
```

```
537   return net;
538 }
539
540 inline AlphaZero::ai::ResNet AlphaZero::ai::Model::
        register_custom_module(ResNet net, std::string layer)
541 {
542   register_module(layer + "_conv1", net.conv1);
543   register_module(layer + "_conv2", net.conv2);
544   register_module(layer + "_batch1", net.batch);
545   register_module(layer + "_batch2", net.batch2);
546   register_module(layer + "_active", net.activ);
547   return net;
548 }
549 inline AlphaZero::ai::Value_head AlphaZero::ai::Model::
        register_custom_module(Value_head net)
550 {
551   register_module("value_conv", net.conv);
552   register_module("value_lin1", net.lin1);
553   register_module("value_lin2", net.lin2);
554   register_module("value_ReLU", net.relu);
555   register_module("value_tanh", net.tanh);
556   return net;
557 }
558 inline AlphaZero::ai::Policy_head AlphaZero::ai::Model::
        register_custom_module(Policy_head net)
559 {
560   register_module("policy_conv", net.conv);
561   register_module("policy_linear", net.lin1);
562   return net;
563 }
```

6.1.11.7.8 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 // mostly useless
11
12 namespace AlphaZero
13 {
```

```
14  namespace ai
15  {
16    // class that allows model prediction to be heald untill a
      certain amount of MCTS threads requested and evaluation or
      skiped
17    class ModelSynchronizer
18    {
19    private: std::vector<std::unique_ptr<Model>> models;
20    private: unsigned short pos = 0;
21    private: std::mutex modelGetMutex;
22
23    public: ModelSynchronizer(std::vector<char*> devices);
24
25         // add Data vor evaluation
26    public: void addData(ModelData* data);
27    private: Model* getModel();
28
29
30    public: void copyModel(ModelSynchronizer*);
31    public: void fit(const std::tuple<torch::Tensor, torch::
      Tensor, torch::Tensor>& batch, const unsigned short& run,
      const unsigned short& trainingLoop);
32
33    public: void save_as_current();
34    public: void save_version(unsigned int version);
35    public: void save_to_file(char* filename);
36    public: void jce_save_current(char* filename);
37    public: void load_current();
38    public: void load_version(unsigned int version);
39    public: void load_from_file(char* filename);
40    public: void jce_load_from_file(char* filename);
41    public: void synchronizeModels();
42
43    public: std::pair<float, torch::Tensor>predict(std::
      shared_ptr<Game::GameState> state, size_t idx=0);
44    public: void predict(ModelData* data, size_t idx=0);
45    public: void predict(std::list<ModelData*> data, size_t idx
      =0);
46    };
47  }
48  namespace test
49  {
50    namespace ModelSynchronizer
51    {
52      std::thread addTestData(ai::ModelData* data, ai::
```

```
    ModelSynchronizer* sync);
53      void _addTestData(ai::ModelData* data, ai::
    ModelSynchronizer* sync);
54    }
55  }
56 }
57
58 inline AlphaZero::ai::ModelSynchronizer::ModelSynchronizer(std::
    vector<char*> devices)
59 {
60   for (auto const& device : devices)
61   {
62     this->models.push_back(std::make_unique<Model>(device));
63   }
64   this->synchronizeModels();
65 }
66 inline void AlphaZero::ai::ModelSynchronizer::addData(ModelData*
     _data)
67 {
68   /*_data->value = 2;
69   std::list<ModelData*>data_l;
70   data_l.push_back(_data);*/
71   this->getModel()->predict(_data);
72 }
73 inline AlphaZero::ai::Model* AlphaZero::ai::ModelSynchronizer::
    getModel()
74 {
75   this->modelGetMutex.lock();
76   auto outputModel = this->models[this->pos].get();
77   this->pos++;
78   if (this->pos >= this->models.size())
79   {
80     this->pos = 0;
81   }
82   this->modelGetMutex.unlock();
83   return outputModel;
84 }
85 inline void AlphaZero::ai::ModelSynchronizer::copyModel(
    ModelSynchronizer* syncher)
86 {
87   for (auto const& model : this->models)
88   {
89     model->copyModel(syncher->models[0].get());
90   }
91 }
```

```
92 inline void AlphaZero::ai::ModelSynchronizer::fit(const std::
       tuple<torch::Tensor, torch::Tensor, torch::Tensor>& batch,
       const unsigned short& run, const unsigned short& trainingLoop
       )
93 {
94   this->models[0]->fit(batch, run, trainingLoop);
95 }
96 inline void AlphaZero::ai::ModelSynchronizer::save_as_current()
97 {
98   this->models[0]->save_as_current();
99 }
100 inline void AlphaZero::ai::ModelSynchronizer::save_to_file(char*
       filename)
101 {
102   this->models[0]->save_to_file(filename);
103 }
104 inline void AlphaZero::ai::ModelSynchronizer::jce_save_current(
       char* filename)
105 {
106   this->models[0]->jce_save_current(filename);
107 }
108 inline void AlphaZero::ai::ModelSynchronizer::save_version(
       unsigned int version)
109 {
110   this->models[0]->save_version(version);
111 }
112 inline void AlphaZero::ai::ModelSynchronizer::load_current()
113 {
114   this->models[0]->load_current();
115   this->synchronizeModels();
116 }
117 inline void AlphaZero::ai::ModelSynchronizer::load_from_file(
       char* filename)
118 {
119   this->models[0]->load_from_file(filename);
120   this->synchronizeModels();
121 }
122 inline void AlphaZero::ai::ModelSynchronizer::jce_load_from_file
       (char* filename)
123 {
124   this->models[0]->jce_load_from_file(filename);
125 }
126 inline void AlphaZero::ai::ModelSynchronizer::load_version(
       unsigned int version)
127 {
```

```
128   this->models[0]->load_version(version);
129   this->synchronizeModels();
130 }
131 inline void AlphaZero::ai::ModelSynchronizer::synchronizeModels
      ()
132 {
133   auto copyFrom = this->models[0].get();
134   for (auto const& model : this->models)
135   {
136     if (model.get() != copyFrom)
137     {
138       model->copyModel(copyFrom);
139     }
140   }
141 }
142 inline void AlphaZero::ai::ModelSynchronizer::predict(ModelData*
      data, size_t idx)
143 {
144   this->models[idx]->predict(data);
145 }
146 inline void AlphaZero::ai::ModelSynchronizer::predict(std::list<
      ModelData*> data, size_t idx)
147 {
148   this->models[idx]->predict(data);
149 }
150 inline std::pair<float, torch::Tensor> AlphaZero::ai::
      ModelSynchronizer::predict(std::shared_ptr<Game::GameState>
      state, size_t idx)
151 {
152   return this->models[idx]->predict(state);
153 }
154
155
156 inline std::thread AlphaZero::test::ModelSynchronizer::
      addTestData(ai::ModelData* data, ai::ModelSynchronizer* sync)
157 {
158   std::thread thread(_addTestData, data, sync);
159   return thread;
160 }
161
162 inline void AlphaZero::test::ModelSynchronizer::_addTestData(ai
      ::ModelData* data, ai::ModelSynchronizer* sync)
163 {
164   sync->addData(data);
165 }
```

6.1.11.7.9 modelWorker.hpp

```
#pragma once

#include "MCTS.hpp"

namespace AlphaZero
{
  namespace ai
  {
    class Node;

    class ModelData
    {
    public: Node* node;
    public: torch::Tensor polys;
    public: float value;

    public: ModelData(Node* node);
    public: void print();
    };
  }
}

inline AlphaZero::ai::ModelData::ModelData(Node* _node)
{
  this->node = _node;
}

inline void print(){}
```

6.1.11.7.10 modelWorker.cpp

```
#include "modelWorker.hpp"
#include "MCTS.hpp"

/*
void AlphaZero::ai::ModelData::print()
{
  std::cout << "state: " << std::endl;
  this->node->state->render();
  std::cout << "polys: " << std::endl;
  std::cout << torch::reshape(this->polys, { action_shape }) <<
    std::endl;
  std::cout << "value: " << this->value << std::endl;
```

```
}*/
```

6.1.11.7.11 playGame.hpp

```
#pragma once

#include <ai/agent.hpp>
#include <ai/memory.hpp>


namespace AlphaZero {
  namespace ai {
    struct gameOutput
    {
      std::unordered_map<Agent*, int> map;
      std::mutex ex;
      gameOutput(Agent*, Agent*);
      void updateValue(Agent*);
    };

    void train(int);
    void playGames(gameOutput* output, Agent* agent1, Agent*
    agent2, Memory* memory, int probMoves, int Epochs, char RunId
    [], int goesFist = 0, bool log = false);

    std::unordered_map<Agent*, int> playGames_inThreads(Game::
    Game* game, Agent* agent1, Agent* agend2, Memory* memory, int
     probMoves, int Epochs, int Threads, char RunId[], int
    goesFirst = 0, bool log = false);
  }
  namespace test {
    void playGame(std::shared_ptr<Game::Game> game, std::
    shared_ptr<ai::Agent> player1, std::shared_ptr<ai::Agent>
    player2, int goesFirst=0);
  }
}

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)
{
  gameOutput output(agent1, agent2);

  std::vector<std::thread> workers;
  for (size_t idx = 0; idx < Threads; idx++)
```

```
33    {
34      bool doLog = (log && (idx == 0));
35      workers.push_back(std::thread(playGames, &output, agent1,
        agent2, memory, probMoves, Epochs, RunId, goesFirst, doLog));
36    }
37
38    for (auto& worker : workers)
39    {
40      worker.join();
41    }
42
43    return output.map;
44  }
45
46  inline void AlphaZero::ai::gameOutput::updateValue(Agent* idx)
47  {
48    this->ex.lock();
49    this->map[idx] = this->map[idx] + 1;
50    this->ex.unlock();
51  }
52
53  inline AlphaZero::ai::gameOutput::gameOutput(Agent* agent1,
      Agent* agent2)
54  {
55    this->map.insert({ agent1, 0 });
56    this->map.insert({ agent2, 0 });
57  }
```

6.1.11.7.12 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,
       std::shared_ptr<ai::Agent> player1, std::shared_ptr<ai::
     Agent> player2, int goesFirst)
8  {
9    if (goesFirst == 0) {
10     goesFirst  = 1;
11     if (rand() % 2) {
12       goesFirst = -1;
13     }
14   }
```

```
15    player1->reset();
16    player2->reset();
17    std::unordered_map<int, std::shared_ptr<ai::Agent>> players =
        { {goesFirst, player1},{-goesFirst, player2} };
18    int action;
19    while (!game->state->done) {
20      action = players[game->state->player]->getAction(game->state
        , false).first;
21      game->takeAction(action);
22    }
23  }
24
25  void AlphaZero::ai::train(int version)
26  {
27    unsigned short iteration = 0;
28    std::vector<char*> devices = { DEVICES };
29    std::shared_ptr<Memory> memory = std::make_shared<Memory>();
30    std::shared_ptr<Game::Game> game = std::make_shared<Game::Game
        >();
31    std::shared_ptr<Agent> currentAgent = std::make_shared<Agent>(
        devices);
32    std::shared_ptr<Agent> bestAgent = std::make_shared<Agent>(
        devices);
33
34    std::vector<int> requiredIterations;
35
36    memory->load();
37    char nameBuff[100];
38
39    currentAgent->identity = 0;
40    bestAgent->identity = 1;
41
42    std::sprintf(nameBuff, "models/run_%d/versionCount.jce",
        runVersion);
43    std::ifstream fin(nameBuff, std::ios::binary);
44    if (fin.is_open())
45    {
46      jce::load(fin, version);
47      std::cout << "found model version: " << version << std::endl
        ;
48
49      fin.close();
50
51      std::sprintf(nameBuff, "models/run_%d/iterationCounter.jce",
        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: " <<
        requiredIterations.size() << std::endl;
57      }
58      else
59      {
60        std::cout << "could not find sutalbe iterationCounter";
61      }
62      fin.close();
63
64      bestAgent->model->load_version(version);
65      currentAgent->model->load_version(version);
66    }
67    else
68    {
69      std::cout << "model version config not found. Defaulting to
      0" << std::endl;
70      version = 0;
71    }
72    currentAgent->model->copyModel(bestAgent->model.get());
73
74    // TODO bestAgent->model->save(0);
75    while (true) { // TODO revert to while !!!
76      iteration++;
77      memory->active = true;
78 #if MainLogger
79      debug::log::mainLogger->info("playing version: {}", version)
      ;
80 #endif
81
82      std::cout << "playing Generational Games:" << std::endl;
83
84 #if ModelLogger
85      debug::log::modelLogger->info("Running Training Games");
86 #endif
87
88      sprintf(nameBuff, "logs/games/game_%d_Generator.gameLog",
      iteration);
89      playGames_inThreads(game.get(), bestAgent.get(), bestAgent.
      get(), memory.get(), probabilitic_moves, EPOCHS, GEN_THREADS,
       nameBuff, 1, false);
90      std::cout << "memory size is: " << memory->memory.size() <<
```

```
    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 Tournement Games:" << std::endl;
101 #if MainLogger
102       debug::log::mainLogger->info("RETRAINING
    ==========================================================");
103 #endif
104
105       sprintf(nameBuff, "logs/games/game_%d_Turney.gameLog",
    iteration);
106 #if ModelLogger
107       debug::log::modelLogger->info("Running Tourney Games");
108 #endif
109       auto score = playGames_inThreads(game.get(), bestAgent.get
    (), currentAgent.get(), memory.get(),
    Turnement_probabiliticMoves, TurneyEpochs, TurneyThreads,
    nameBuff, 0, true);
110
111       std::cout << "Turney ended with: " << score[currentAgent.
    get()] << " : " << score[bestAgent.get()] << std::endl;
112       if (score[currentAgent.get()] > score[bestAgent.get()] *
    scoringThreshold) {
113         version++;
114         //TODO copy model weightsk
115         currentAgent->model->save_as_current();
116         bestAgent->model->copyModel(currentAgent->model.get());
117         bestAgent->model->save_version(version);
118
119         std::sprintf(nameBuff, "models/run_%d/versionCount.jce",
     runVersion);
120         std::ofstream fout(nameBuff, std::ios::binary);
121         jce::save(fout, version);
122         fout.close();
123
124         memory->save();
125
126         requiredIterations.push_back(iteration);
```

```
127             std::sprintf(nameBuff, "models/run_%d/iterationCounter.
    jce", runVersion);
128             fout.open(nameBuff, std::ios::binary);
129             jce::save(fout, requiredIterations);
130             fout.close();
131 
132             iteration = 0;
133           }
134         }
135       }
136 }
137 
138 void AlphaZero::ai::playGames(gameOutput* output, Agent* agent1,
     Agent* agent2, Memory* memory, int probMoves, int Epochs,
    char RunId[], int _goesFist, bool do_log)
139 {
140   std::srand(std::chrono::time_point_cast<std::chrono::
    nanoseconds>(std::chrono::system_clock::now()).
    time_since_epoch().count());
141   auto game = std::make_unique<Game::Game>();
142   int goesFist = (_goesFist == 0) ? 1 : _goesFist;
143 #if ProfileLogger
144   debug::Profiler::profiler.switchOperation(3);
145 #endif
146 
147 #if SaverType == 1
148   io::FullState::GameSaver saver = io::FullState::GameSaver();
149 #elif SaverType == 2
150   io::ActionsOnly::GameSaver saver = io::ActionsOnly::GameSaver
    ();
151 #endif
152 
153   for (int epoch = 0; epoch < Epochs; epoch++) {
154 #if RenderGenAndTurneyProgress
155     jce::consoleUtils::render_progress_bar((float)epoch / (float
    )Epochs);
156 #endif
157 #if ProfileLogger
158     debug::Profiler::profiler.switchOperation(3);
159 #endif
160 #if SaverType == 2 || SaverType == 1
161     saver.addGame();
162 #endif
163 #if MainLogger
164     if (epoch == 0 && do_log) {
```

```
165       debug::log::mainLogger->info("
    ==========================================================================
    ");
166       debug::log::mainLogger->info("======================
    playing Next match ========================");
167       debug::log::mainLogger->info("
    ==========================================================================
    ");
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",
    goesFist);
188     }
189 #endif
190     game->reset();
191     int turn = 0;
192     while (!game->state->done) {
193       turn++;
194       //std::cout << turn << std::endl;
195       auto actionData = players[game->state->player]->getAction(
    game->state, probMoves > turn);
196       tmpMemory.commit(game->state, actionData.second.first);
197 #if SaverType == 1
198       saver.addState(game->state);
199 #elif SaverType == 2
200       saver.addState(actionData.first);
201 #endif
202 #if MainLogger
```

```
203         if (epoch == 0 && do_log) {
204           game->state->render(debug::log::mainLogger);
205           debug::log::mainLogger->info("MSCT vals: {:1.5f}",
    actionData.second.second);
206           debug::log::logVector(debug::log::mainLogger, actionData
    .second.first);
207           debug::log::mainLogger->info("NN vals: {:1.5f}", players
    [game->state->player]->predict(game->state).first);
208           debug::log::logVector(debug::log::mainLogger, players[
    game->state->player]->predict(game->state).second);
209 
210           debug::log::mainLogger->info("selected action is: {}",
    actionData.first);
211         }
212 #endif
213         //game->render();
214         game->takeAction(actionData.first);
215       }
216       //std::cout << turn << std::endl;
217       //game->render();
218       //tmpMemory.commit(game->state);//   add end game states to
    memory ??
219 #if SaverType == 1
220       saver.addState(game->state);
221 #endif
222 #if MainLogger
223       if (epoch == 0 && do_log) {
224         game->state->render(debug::log::mainLogger);
225       }
226 #endif
227 #if ProfileLogger
228       debug::Profiler::profiler.switchOperation(4);
229 #endif
230       memory->updateMemory(game->state->player, std::get<0>(game->
    state->val), &tmpMemory);
231       if (true)
232       {
233         if (std::get<0>(game->state->val) != 0)
234         {
235           output->updateValue(players[game->state->player * std::
    get<0>(game->state->val)]);
236         }
237       }
238 #if ProfileLogger
239       debug::Profiler::profiler.stop();
```

```
#endif
  }

#if SaverType == 2 || SaverType == 1
  saver.save(RunId);
#endif

#if RenderGenAndTurneyProgress
  jce::consoleUtils::render_progress_bar(1.0f, true);
#endif
}
```

6.1.11.7.13 utils.hpp

```
#pragma once
#include <ostream>
#include <istream>
#include <vector>

namespace AlphaZero {
  namespace ai {
    // Softams function inp is an iteratable of numbers
    template<typename T>
    void softmax(T& inp);

    template<typename T>
    void linmax(T& inp);

    template<typename T>
    T getSumm(std::vector<T>& val);
  }
}

template<typename T>
inline void AlphaZero::ai::softmax(T& inp){
  typedef float number;

  number m = -10e100;
  for (number const& z : inp){
    if (m < z) {
      m = z;
    }
  }

  number sum = 0.0;
  for (number const& z : inp) {
```

```
33      sum += exp(z - m);
34    }
35
36    number constant = m + log(sum);
37    for (number& z : inp) {
38      z = exp(z - constant);
39    }
40    throw "Depricated function";
41    return;
42  }
43
44  template<typename T>
45  void AlphaZero::ai::linmax(T& inp)
46  {
47    float sum = 0;
48    for (auto const& idx : inp)
49    {
50      sum = sum + idx;
51    }
52    for (auto& idx : inp)
53    {
54      idx = idx / sum;
55    }
56    return;
57  }
58
59  template<typename T>
60  T AlphaZero::ai::getSumm(std::vector<T>& val)
61  {
62    T out = 0;
63    for (const T& value : val)
64    {
65      out = out + value;
66    }
67    return out;
68  }
```

6.1.11.8 game

6.1.11.8.1 game.hpp

```
1  #pragma once
2  /*
3  this is the alpha Zero game for connect4
4  */
5
```

```cpp
#include <iostream>
#include <vector>
#include <list>
#include <memory>
#include <tuple>
#include <unordered_map>
#include <bitset>
#include <unordered_set>
#include <torch/torch.h>

#include "config.hpp"


#define input_shape_x 7
#define input_shape_y 6
#define input_snape_z 2
#define action_count 42
#define action_shape 6, 7
#define boardOfset 42 // the size of a layer of the board in the buffer. (the amount of felds)
#define gameName "connect4"


namespace AlphaZero {
  namespace Game {
    class GameState {
    public: int player;
    public: bool done;
    public: std::tuple<int, int, int> val;
    public: IDType gameBoard;
    public: std::vector<int> allowedActions;

    public: GameState(IDType board, int _player);
    public: GameState();
    private: void initialize(IDType board, int _player);
    public: std::shared_ptr<GameState> takeAction(int action);
    public: void gameIsDone();
    protected: void getAllowedActions();
    public: int IdIndex(int id);
    public: IDType id();
    public: void render();
#if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger || ModelLogger)
    public: void render(std::shared_ptr<spdlog::logger> logger);
#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      private: char getPiece(int val);
53      private: std::pair<int, bool> getAllowedColumHeight(int);
54      };
55      struct StateHash
56      {
57        std::size_t operator()(std::pair<std::shared_ptr<GameState
        >, std::vector<int>> const& s) const noexcept;
58      };
59      // optimization function its not a problem if not all are
        found
60      std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
        GameState>, std::vector<int>>> identities(std::shared_ptr<
        GameState> state, std::vector<int>& actionProbs);
61
62      class Game {
63      public: std::tuple<int, int> BoardShape = { 3,3 };
64      public: std::tuple<int, int, int> inputShape = { 2,3,3 };
65      public: std::shared_ptr<GameState> state;
66
67      public: Game();
68      public: void reset();
69      public: void takeAction(int action);
70      public: bool takeHumanAction(int action);
71      public: void render();
72      };
73
74      inline void test() {
75        AlphaZero::Game::Game* game = new AlphaZero::Game::Game();
76
77        while (!game->state->done) {
78          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 };
79          auto idents = identities(game->state, vec);
80          idents[1].first->render();
81          std::cout << "your action: ";
82          int action;
83          std::cin >> action;
84          game->takeHumanAction(action);
85  #if Windows
```

```
86          system("cls");
87 #else
88          system("clear");
89 #endif
90        }
91        game->render();
92
93        std::cout << std::endl << "the last player just won";
94      }
95    }
96 }
97
98 inline std::size_t AlphaZero::Game::StateHash::operator()(std::
       pair<std::shared_ptr<GameState>, std::vector<int>> const& s)
        const noexcept {
99   return s.first->gameBoard.to_ullong();
100 }
101
102 inline int AlphaZero::Game::GameState::IdIndex(int id)
103 {
104   if (this->gameBoard[id] == 1) {
105     return 1;
106   }
107   else if (this->gameBoard[id + boardOfset] == 1) {
108     return -1;
109   }
110   return 0;
111 }
112
113 inline void AlphaZero::Game::GameState::IdIndex(int id, int val,
       IDType& b)
114 {
115   if (val == 0) {
116     b.set(id, 0);
117     b.set(id + boardOfset, 0);
118     return;
119   }
120   if (val == -1) {
121     id += boardOfset;
122   }
123   b.set(id, 1);
124 }
125
126 inline char AlphaZero::Game::GameState::getPiece(int id)
127 {
```

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

6.1.11.8.2 game.cpp

```
1 #include "game.hpp"
```

```
2
3 #define columOfset 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 |
```

```
    +----+----+----+----+----+----+----+
    | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
    +----+----+----+----+----+----+----+
    | 35 | 36 | 37 | 38 | 39 | 40 | 41 |
    +----+----+----+----+----+----+----+
    */
    //horizontal
    {0,1,2,3},
    {1,2,3,4},
    {2,3,4,5},
    {3,4,5,6},

    {7,8,9,10},
    {8,9,10,11},
    {9,10,11,12},
    {10,11,12,13},

    {14,15,16,17},
    {15,16,17,18},
    {16,17,18,19},
    {17,18,19,20},

    {21,22,23,24},
    {22,23,24,25},
    {23,24,25,26},
    {24,25,26,27},

    {28,29,30,31},
    {29,30,31,32},
    {30,31,32,33},
    {31,32,33,34},

    {35,36,37,38},
    {36,37,38,39},
    {37,38,39,40},
    {38,39,40,41},
    //vertical
    {0 , 7,14,21},
    {7 ,14,21,28},
    {14,21,28,35},

    {1 ,8 ,15,22},
    {8 ,15,22,29},
    {15,22,29,36},

```

```
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
       this player, points for other player
167         return;
168       }
169     }
170     this->done = false;
171     this->val = { 0, 0, 0 };
172 }
173
174 inline std::pair<int, bool> AlphaZero::Game::GameState::
      getAllowedColumHeight(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 + columOfset)!=0) {
182     return { idx, true };
183   }
184   else {
185     return this->getAllowedColumHeight(idx + columOfset);
186   }
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->getAllowedColumHeight(idx)
      ;
194     if (data.second) {
195       this->allowedActions.push_back(data.first);
196     }
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
    || ModelLogger)
215 void AlphaZero::Game::GameState::render(std::shared_ptr<spdlog::
    logger> logger)
216 {
217   for (int idx = 0; idx < 6; idx++) {
218     char line1[] = {
```

```
219         this->getPiece(0 + columOfset * idx), ' ',
220         this->getPiece(1 + columOfset * idx), ' ',
221         this->getPiece(2 + columOfset * idx), ' ',
222         this->getPiece(3 + columOfset * idx), ' ',
223         this->getPiece(4 + columOfset * idx), ' ',
224         this->getPiece(5 + columOfset * idx), ' ',
225         this->getPiece(6 + columOfset * idx), NULL
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::
     Game::GameState> state, std::vector<int>& actionProbs) {
260   IDType boardBuffer;
261
```

```
262   std::vector<int> probs = {
263     actionProbs[6],  actionProbs[5],  actionProbs[4],
      actionProbs[3],  actionProbs[2],  actionProbs[1],
      actionProbs[0],
264     actionProbs[13], actionProbs[12], actionProbs[11],
      actionProbs[10], actionProbs[9],  actionProbs[8],
      actionProbs[7],
265     actionProbs[20], actionProbs[19], actionProbs[18],
      actionProbs[17], actionProbs[16], actionProbs[15],
      actionProbs[14],
266     actionProbs[27], actionProbs[26], actionProbs[25],
      actionProbs[24], actionProbs[23], actionProbs[22],
      actionProbs[21],
267     actionProbs[34], actionProbs[33], actionProbs[32],
      actionProbs[31], actionProbs[30], actionProbs[29],
      actionProbs[28],
268     actionProbs[41], actionProbs[40], actionProbs[39],
      actionProbs[38], actionProbs[37], actionProbs[36],
      actionProbs[35]
269   };
270 #define assignStateSinge(idx1, idx2)AlphaZero::Game::GameState::
      IdIndex(idx1, state->IdIndex(idx2), boardBuffer)
271 #define assignState(idx1, idx2)assignStateSinge(idx1, idx2);
      assignStateSinge(idx2, idx1);
272
273
274   assignState(0,  6);  assignState(1,  5);  assignState(2,  4);
        assignStateSinge(3,  3);
275   assignState(7,  13); assignState(8,  12); assignState(9,  11);
        assignStateSinge(10, 10);
276   assignState(14, 20); assignState(15, 19); assignState(16, 18);
        assignStateSinge(17, 17);
277   assignState(21, 27); assignState(22, 26); assignState(23, 25);
        assignStateSinge(24, 24);
278   assignState(28, 34); assignState(29, 33); assignState(30, 32);
        assignStateSinge(31, 31);
279   assignState(35, 41); assignState(36, 40); assignState(37, 39);
        assignStateSinge(38, 38);
280 #undef assignState
281
282   return { std::make_shared<AlphaZero::Game::GameState>(
      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)
{
  std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
    GameState>, std::vector<int>>> idents(2);
  int id = 0;
  idents[0] = { state, probs };
  idents[1] = mirrorGameState(state, probs);
  return idents;
}
```

6.1.11.9 jce

6.1.11.9.1 load.hpp

```
#pragma once

#include <fstream>
#include <list>
#include <vector>
#include <bitset>
#include <string>
#include <ai/memory.hpp>
#include <torch/torch.h>
#include <game/game.hpp>

#define BasicLoad(in, data) (in.read((char*)&data, sizeof(data))
    )

namespace jce
{
  // custom part

  // load GameState from file
  void load(std::ifstream& in, std::shared_ptr<AlphaZero::Game::
    GameState>& state);

  // load Memory Element from file
  void load(std::ifstream& in, std::shared_ptr<AlphaZero::ai::
    MemoryElement>& element);

  template<typename key, typename T>
  void load(std::ifstream& in, torch::OrderedDict<key, T>& map);

  // load Tensor from file
  template<typename T=float>
```

```
29    void load(std::ifstream& in, torch::Tensor& tensor);
30
31    // end custom part
32
33    template<typename key, typename T>
34    void load(std::ifstream& in, std::unordered_map<key, T>& map);
35
36    // load pair from file
37    template<typename T, typename T2>
38    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 actuall 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 unsinged 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);
```

```
}

inline void jce::load(std::ifstream& in, char*& data)
{
  std::vector<char> data_vec;
  while (true)
  {
    char next;
    in.read(&next, 1);
    data_vec.push_back(next);
    if (next == NULL)
    {
      break;
    }
  }
  data = new char[data_vec.size()];
  auto pos = data;
  for (auto const& value : data_vec)
  {
    (*pos) = value;
    pos++;
  }
}
```

6.1.11.9.2 save.hpp

```
#pragma once

#include <fstream>
#include <iostream>
#include <list>
#include <vector>
#include <string>
#include <bitset>
#include <ai/memory.hpp>
#include <torch/torch.h>

#define BasicSave(data, out) (out.write((char*)&data, sizeof(
    data)))
#define BasicSave_cp(data, out) (out.write((char*)data, sizeof(
    data)))

namespace jce
{
  // custom part

```

```
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 actuall 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 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<
        AlphaZero::ai::MemoryElement> const& element)
78  {
79    jce::save(out, element->value);
80    jce::save(out, element->state);
81    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   jce::save(out, data.size());
169   for (auto const& data : data)
170   {
171     jce::save(out, data);
172   }
173 }
174
175 inline void jce::save(std::ofstream& out, int const& data) {
    BasicSave(data, out); }
176 inline void jce::save(std::ofstream& out, unsigned int const&
    data) { BasicSave(data, out); }
177 inline void jce::save(std::ofstream& out, size_t const& data) {
    BasicSave(data, out); }
178 inline void jce::save(std::ofstream& out, float const& data) {
    BasicSave(data, out); }
179 inline void jce::save(std::ofstream& out, double const& data) {
```

```
    BasicSave(data, out); }
180 inline void jce::save(std::ofstream& out, const int64_t& data) {
     BasicSave(data, out); }
181
182 inline void jce::save(std::ofstream& out, const std::string&
    data)
183 {
184   jce::save(out, data.c_str());
185 }
186
187 inline void jce::save(std::ofstream& out, const char* arr)
188 {
189   size_t size = 1;
190   auto iterator = arr;
191   while (*iterator != NULL)
192   {
193     size++;
194     iterator++;
195   }
196   out.write(arr, size);
197 }
```

6.1.11.9.3 string.hpp

```
1 #pragma once
2 #include <iostream>
3
4 namespace jce
5 {
6     namespace consoleUtils
7     {
8         void render_progress_bar(float progress, bool persistand
    = false);
9     }
10 }
11
12 inline void jce::consoleUtils::render_progress_bar(float
    progress, bool persistant)
13 {
14 #if true
15     if (progress <= 1.0) {
16         int barWidth = 70;
17
18         std::cout << "[";
19         int pos = barWidth * progress;
20         for (int i = 0; i < barWidth; ++i) {
```

```
21              if (i < pos) std::cout << "=";
22              else if (i == pos) std::cout << ">";
23              else std::cout << " ";
24          }
25          if (persistant)
26          {
27              std::cout << "] " << int(progress * 100.0) << std::
    endl;
28          }
29          else
30          {
31              std::cout << "] " << int(progress * 100.0) << " %\r"
    ;
32          }
33          std::cout.flush();
34      }
35 #endif
36 }
```

6.1.11.9.4 vector.hpp

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

6.1.11.10 server

6.1.11.10.1 eloClient.hpp

```
1  #pragma once
```

```
2
3  #include <config.hpp>
4  #include <sockpp/tcp_connector.h>
5  #include <ai/agent.hpp>
6
7
8  #define ELO_PORT 2551
9  #define ELO_IP "wandhoven.ddns.net"
10
11 namespace AlphaZero
12 {
13   namespace elo
14   {
15     class eloClient
16     {
17     public: int send(int agent1, int agent2, int win) const;
18     public: int setElo(int agent1, int eloRating) const;
19     public: int getElo(int agent1) const;
20     public: int getAgentWithClosestElo(int eloVal) const;
21     };
22   }
23 }
24
25 inline int AlphaZero::elo::eloClient::send(int agent1, int
       agent2, int win) const
26 {
27   sockpp::socket_initializer sockInit;
28   in_port_t port = ELO_PORT;
29   std::string host = ELO_IP;
30   sockpp::tcp_connector conn({ host, port });
31   if (!conn)
32   {
33     std::cout << (conn.last_error_str()) << std::endl;
34     return -1;
35   }
36
37   int data[4] = { 3, agent1, agent2, win };
38   conn.write(data, sizeof(data));
39
40   int elo [1];
41   conn.read(elo, sizeof(elo));
42   return elo[0];
43 }
44
45 inline int AlphaZero::elo::eloClient::setElo(int agent1, int elo
```

```
    ) const
{
  sockpp::socket_initializer sockInit;
  in_port_t port = ELO_PORT;
  std::string host = ELO_IP;
  sockpp::tcp_connector conn({ host, port });
  if (!conn)
  {
    std::cout << (conn.last_error_str()) << std::endl;
    return -1;
  }

  int data[3] = { 2, agent1, elo };
  conn.write(data, sizeof(data));

  int delo[1];
  conn.read(delo, sizeof(delo));
  return delo[0];
}

inline int AlphaZero::elo::eloClient::getElo(int agent1) const
{
  sockpp::socket_initializer sockInit;
  in_port_t port = ELO_PORT;
  std::string host = ELO_IP;
  sockpp::tcp_connector conn({ host, port });
  if (!conn)
  {
    std::cout << (conn.last_error_str()) << std::endl;
    return -1;
  }

  int data[2] = { 1, agent1 };
  conn.write(data, sizeof(data));

  int elo[1];
  conn.read(elo, sizeof(elo));
  return elo[0];
}

inline int AlphaZero::elo::eloClient::getAgentWithClosestElo(int
    val) const
{
  sockpp::socket_initializer sockInit;
  in_port_t port = ELO_PORT;
```

```
  std::string host = ELO_IP;
  sockpp::tcp_connector conn({ host, port });
  if (!conn)
  {
    std::cout << (conn.last_error_str()) << std::endl;
    return -1;
  }

  int data[2] = { -1, val };
  conn.write(data, sizeof(data));

  int elo[1];
  conn.read(elo, sizeof(elo));
  return elo[0];
}
```

6.1.11.10.2 server.hpp

```
#pragma once

#include <config.hpp>
#include <sockpp/tcp_acceptor.h>
#include <ai/agent.hpp>


#define PORT 25500

namespace AlphaZero
{
  namespace Server
  {
    IDType toBoard(int arr[]);

    class TCPServer
    {
    private: void evaluate(sockpp::tcp_socket& sock);
    private: sockpp::socket_initializer sockInit;
    private: sockpp::tcp_acceptor acc;
    private: void accept();

    public: TCPServer(int port = PORT);
    public: void mainLoop();
    };

    class TestServer
    {
```

```
29      private: sockpp::socket_initializer sockInit;
30      private: sockpp::tcp_acceptor acc;
31      private: void accept();
32
33      public: TestServer(int port = PORT);
34      public: void mainLoop();
35      };
36    }
37  }
38
39  inline IDType AlphaZero::Server::toBoard(int arr[])
40  {
41    IDType out;
42    for (int idx = 0; idx < stateSize; idx++)
43    {
44      out.set(idx, arr[idx]);
45    }
46    return out;
47  }
```

6.1.1.11.10.3 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: {:1.5f}", actionData.second.second);
43    debug::log::logVector(logger, actionData.second.first);
44    logger->info("NN vals: {:1.5f}", 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 acceptd 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
     ().to_string());
80   logger->info("Awaiting connections on port: {}", port);
81 }
82
83 void AlphaZero::Server::TCPServer::mainLoop()
84 {
85   while (true)
86   {
87     this->accept();
88   }
89 }
90
91 AlphaZero::Server::TestServer::TestServer(int _port)
92 {
93   in_port_t port = _port;
94   this->acc = sockpp::tcp_acceptor(port);
95
96   if (!acc) {
97     std::cerr << "Error creating the acceptor: " << acc.
     last_error_str() << std::endl;
98   }
99   std::cout << "Acceptor bound to address: " << acc.address() <<
      std::endl;
100  std::cout << "Awaiting connections on port " << port << "..."
```

```
    << std::endl;
}

void AlphaZero::Server::TestServer::mainLoop()
{
  while (true)
  {
    this->accept();
  }
}

void AlphaZero::Server::TestServer::accept()
{
  sockpp::tcp_socket sock = this->acc.accept();

  std::cout << "Connection acceptd from " << sock.peer_address()
    << std::endl;

  ssize_t n;
  int buf[stateSize + 1];
  int out[1];

  n = sock.read(buf, sizeof(buf));

  std::shared_ptr<Game::GameState> state = std::make_shared<Game
    ::GameState>(toBoard(buf), buf[stateSize]);
  state->render();

  std::cout << "Server Action for testing: ";
  std::cin >> out[0];
  std::cout << std::endl;

  sock.write_n(out, sizeof(int));

  std::cout << "Connection closed from " << sock.peer_address()
   << std::endl;
}
```

6.1.1.11 test

6.1.1.11.1 testSuit.hpp

```
#include <ai/model.hpp>
#include <ai/agent.hpp>
#include "testUtils.hpp"

```

```
5  namespace AlphaZero
6  {
7    namespace test
8    {
9      void runTests();
10
11     void testCoppying();
12     void testSave();
13     void testJCESave();
14     void testLossLog();
15     void testModelData();
16     void testTraining();
17     void testModelSpeed();
18     void testModelSyncronization();
19
20     bool compareAgents(std::shared_ptr<ai::Agent> anget1, std::
       shared_ptr<ai::Agent> anget2);
21     std::shared_ptr<Game::GameState> getRandomState();
22   }
23 }
24
25 inline bool AlphaZero::test::compareAgents(std::shared_ptr<ai::
     Agent> anget1, std::shared_ptr<ai::Agent> anget2)
26 {
27   auto state = getRandomState();
28
29   auto valsA = anget1->predict(state);
30   auto valsB = anget2->predict(state);
31
32   if (valsA.first != valsB.first) { return false; }
33   for (size_t idx = 0; idx < action_count; idx++)
34   {
35     if (valsA.second[idx] != valsB.second[idx]) { return false;
       }
36   }
37   return true;
38 }
39
40
41 inline std::shared_ptr<AlphaZero::Game::GameState> AlphaZero::
     test::getRandomState()
42 {
43   std::bitset<stateSize> board;
44   for (size_t idx = 0; idx < stateSize; idx++)
45   {
```

```
46      board.set(idx, rand() % 2);
47    }
48    auto state = std::make_shared<Game::GameState>(board, rand() %
        2);
49    return state;
50  }
```

6.1.1.11.11.2 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   testCoppying();
15   testSave();
16   testJCESave();
17   testLossLog();
18   testModelSyncronization();
19   if (torch::cuda::cudnn_is_available() || randomModel)
20     testModelSpeed();
21 }
22
23
24 void AlphaZero::test::testCoppying()
25 {
26   std::cout << "Testing Model coppying ...\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::playGames_inThreads(game.get(), bestAgent.get
      (), bestAgent.get(), memory.get(),
```

```
    Turnement_probabiliticMoves, TurneyEpochs, TurneyThreads,
    nameBuff, 0, true);
  std::cout << timer.elapsed() << std::endl;
}

void AlphaZero::test::testModelSyncronization()
{
  std::cout << "testing Model Synchronization ...\t";
  std::vector<char*> devices = { "cpu", "cpu"};
  std::shared_ptr<ai::Agent> bestAgent = std::make_shared<ai::
    Agent>(devices);

  auto state = getRandomState();

  auto valsA = bestAgent->model->predict(state, 0);
  auto valsB = bestAgent->model->predict(state, 1);

  bool isValid = true;
  if (valsA.first != valsB.first) { isValid = false; }
  if (!torch::equal(valsA.second, valsB.second)) { isValid =
    false; }
  printSuccess(isValid);
}
```

6.1.11.11.3 testUtils.hpp

```
#pragma once
#include <iostream>

namespace AlphaZero
{
  namespace test
  {
    void printSuccess(bool val);
  }
}

inline void AlphaZero::test::printSuccess(bool val)
{
  if (val)
  {
    std::cout << "\33[32;1mSuccess\33[0m" << std::endl;
  }
  else
  {
    std::cout << "\33[31;1mFailed\33[0m" << std::endl;
```

```
  }
}
```

6.2 Clients

6.2.1 ConsoleClient

6.2.1.1 ConsoleClient.h

```
// ConsoleClient.h : Include file for standard system include files,
// or project specific include files.

#pragma once

#include <iostream>


// TODO: Reference additional headers your program requires here.
```

6.2.1.2 ConsoleClient.cpp

```
// ConsoleClient.cpp : Defines the entry point for the application.
//

#include "ConsoleClient.h"
#include <agent.hpp>

using namespace std;

void playGame(std::shared_ptr<Agents::Agent>agent1, std::shared_ptr<Agents::Agent>agent2, std::shared_ptr<AlphaZero::Game::Game>game)
{
  while (!game->state->done)
  {
    int action;
    switch (game->state->player)
    {
    case(1):{
      action = agent1->getAction(game);
      break;
    }
```

```
    case(-1): {
      action = agent2->getAction(game);
      break;
    }
    }
    game->takeAction(action);
  }
}

int main()
{
  auto game = std::make_shared<AlphaZero::Game::Game>();
  auto user = std::make_shared<Agents::User>();
  auto AI = std::make_shared<Agents::RemoteAgent>("35.240.231.50
    ", 12345);
  playGame(user, AI, game);
  return 0;
}
```

6.2.1.3 include

6.2.1.3.1 agent.hpp

```
#pragma once
#pragma comment( lib, "ws2_32.lib")

#include "game.hpp"
#include "modifications.hpp"
#include <string>
#include <sockpp/tcp_connector.h>

namespace Agents
{
  class Agent
  {
  public: virtual int getAction(std::shared_ptr<AlphaZero::Game
    ::Game> game) = 0;
  };
  class User : public Agent
  {
  public: virtual int getAction(std::shared_ptr<AlphaZero::Game
    ::Game> game);
  private: int subGetAction(std::shared_ptr<AlphaZero::Game::
    Game> game);
  };

```

```
21    class RemoteAgent : public Agent
22    {
23    private: sockpp::socket_initializer sockInit;
24    private: std::string ip;
25    private: in_port_t port;
26
27    public: RemoteAgent(std::string host, in_port_t port);
28    public: virtual int getAction(std::shared_ptr<AlphaZero::Game
        ::Game> game);
29
30    public: void toArr(int* arr, std::shared_ptr<AlphaZero::Game::
        Game> game);
31    };
32  }
33
34  inline int Agents::User::getAction(std::shared_ptr<AlphaZero::
        Game::Game> game)
35  {
36  #if WIN32
37    system("cls");
38  #else
39    system("clear");
40  #endif
41    game->render();
42    modifications::bottomLable();
43    return this->subGetAction(game);
44  }
45
46  inline char currentPlayerIcon(int player)
47  {
48    switch(player)
49    {
50    case(1): {return 'X'; };
51    case(-1): {return 'O'; };
52    default: {return 'E'; }
53    }
54  }
55
56  inline int Agents::User::subGetAction(std::shared_ptr<AlphaZero
        ::Game::Game> game)
57  {
58    std::cout << std::endl << "Move for " << currentPlayerIcon(
        game->state->player) << ": ";
59    int res;
60    try {
```

```
61      std::cin >> res;
62    }
63    catch (...){
64      return this->subGetAction(game);
65    }
66    for (auto const& val : game->state->allowedActions)
67    {
68      if (res == modifications::allowedActionModification(val))
69      {
70        return val;
71      }
72    }
73    std::cin.clear();
74    std::cin.ignore(INT_MAX);
75    std::cout << std::endl << "\33[31;1mIllegal Move try again
        \33[0m" << std::endl;
76    return this->subGetAction(game);
77  }
78
79  inline Agents::RemoteAgent::RemoteAgent(std::string _host,
        in_port_t port)
80  {
81    this->ip = _host;
82    this->port = port;
83    this->sockInit = sockpp::socket_initializer();
84  }
85
86  inline int Agents::RemoteAgent::getAction(std::shared_ptr<
        AlphaZero::Game::Game> game)
87  {
88    int arr[GameBoardHolderSize + 1];
89    this->toArr(arr, game);
90
91    sockpp::tcp_connector con({ this->ip, this->port });
92    con.write(arr, (GameBoardHolderSize + 1) * sizeof(int));
93
94    int out[1];
95    con.read_n(out, sizeof(int));
96    return out[0];
97  }
98
99  inline void Agents::RemoteAgent::toArr(int* arr, std::shared_ptr
        <AlphaZero::Game::Game> game)
100 {
101   for (int idx = 0; idx < GameBoardHolderSize; idx++)
```

```
102   {
103     arr[idx] = game->state->gameBoard.test(idx);
104   }
105   arr[GameBoardHolderSize] = game->state->player;
106 }
```

6.2.1.3.2 game.hpp

```
1  #pragma once
2  /*
3  this is the alpha Zero game for tick tack toe
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
15 #define input_shape_x 6
16 #define input_shape_y 7
17 #define input_snape_z 2
18 #define action_count 42
19 #define boardOfset 42 // the size of a layer of the board in the
       buffer. (the amount of felds)
20 #define gameName "connect4"
21
22 #define GameBoardHolderSize 84
23
24 typedef std::bitset<GameBoardHolderSize> IDType;
25
26 namespace AlphaZero {
27   namespace Game {
28     class GameState {
29     public: int player;
30     public: bool done;
31     public: std::tuple<int, int, int> val;
32     public: IDType gameBoard;
33     public: std::list<int> allowedActions;
34
35     public: GameState(IDType board, int _player);
36     public: GameState();
37     private: void initialize(IDType board, int _player);
```

```
38      public: std::shared_ptr<GameState> takeAction(int action);
39      public: void gameIsDone();
40      protected: std::list<int> getAllowedActions();
41      public: int IdIndex(int id);
42      public: IDType id();
43      public: void render();
44  #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
        || ModelLogger)
45      public: void render(std::shared_ptr<spdlog::logger> logger);
46  #endif
47      public: void static IdIndex(int id, int val, IDType& b);
48      private: char getPiece(int val);
49      private: std::pair<int, bool> getAllowedColumHeight(int);
50      };
51      struct StateHash
52      {
53        std::size_t operator()(std::pair<std::shared_ptr<GameState
        >, std::vector<int>> const& s) const noexcept;
54      };
55      // optimization function its not a problem if not all are
        found
56      std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
        GameState>, std::vector<float>>> identities(std::shared_ptr<
        GameState> state, std::vector<float>& actionProbs);
57
58      class Game {
59      public: std::tuple<int, int> BoardShape = { 3,3 };
60      public: std::tuple<int, int, int> inputShape = { 2,3,3 };
61      public: std::shared_ptr<GameState> state;
62
63      public: Game();
64      public: void reset();
65      public: void takeAction(int action);
66      public: bool takeHumanAction(int action);
67      public: void render();
68      };
69
70      inline void test() {
71        AlphaZero::Game::Game* game = new AlphaZero::Game::Game();
72
73        while (!game->state->done) {
74          std::vector<float> vec = { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0,
         6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0,
        16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0,
        26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0,
```

```
    36.0, 37.0, 38.0, 39.0, 40.0, 41.0 };
75          auto idents = identities(game->state, vec);
76          idents[1].first->render();
77          std::cout << "your action: ";
78          int action;
79          std::cin >> action;
80          game->takeHumanAction(action);
81  #if Windows
82          system("cls");
83  #else
84          system("clear");
85  #endif
86        }
87        game->render();
88
89        std::cout << std::endl << "the last player just won";
90      }
91    }
92  }
93
94  inline std::size_t AlphaZero::Game::StateHash::operator()(std::
      pair<std::shared_ptr<GameState>, std::vector<int>>  const& s)
       const noexcept {
95    return s.first->gameBoard.to_ullong();
96  }
97
98  inline int AlphaZero::Game::GameState::IdIndex(int id)
99  {
100   if (this->gameBoard[id] == 1) {
101     return 1;
102   }
103   else if (this->gameBoard[id + boardOfset] == 1) {
104     return -1;
105   }
106   return 0;
107 }
108
109 inline void AlphaZero::Game::GameState::IdIndex(int id, int val,
       IDType& b)
110 {
111   if (val == 0) {
112     b.set(id, 0);
113     b.set(id + boardOfset, 0);
114     return;
115   }
```

```
116   if (val == -1) {
117     id += boardOfset;
118   }
119   b.set(id, 1);
120 }
121
122 inline char AlphaZero::Game::GameState::getPiece(int id)
123 {
124   std::unordered_map<int, char> renderData = { {0,'-'}, {1, 'X'
      }, {-1, 'O'} };
125   if (std::find(this->allowedActions.begin(), this->
      allowedActions.end(), id) != this->allowedActions.end()) {
126     return '+';
127   }
128   auto va = renderData[this->IdIndex(id)];
129   return va;
130 }
131
132 inline IDType AlphaZero::Game::GameState::id()
133 {
134   return this->gameBoard;
135 }
136
137 inline void AlphaZero::Game::Game::render() {
138   this->state->render();
139 }
```

6.2.1.3.3 modifications.hpp

```
1 #pragma once
2
3 namespace modifications
4 {
5   inline void bottomLable()
6   {
7     std::cout << "0 1 2 3 4 5 6" << std::endl;
8   }
9   inline int allowedActionModification(int action)
10  {
11    return action % 7;
12  }
13 }
```

6.2.1.4 scr

6.2.1.4.1 game.cpp

```cpp
#include "game.hpp"

#define columOfset 7

AlphaZero::Game::GameState::GameState(IDType board, int _player)
{
  this->initialize(board, _player);
}

AlphaZero::Game::GameState::GameState()
{
  this->initialize(IDType(), 1);
}

void AlphaZero::Game::GameState::initialize(IDType board, int
    _player)
{
  this->gameBoard = board;
  this->player = _player;
  this->allowedActions = this->getAllowedActions();
  this->gameIsDone();
}

std::shared_ptr<AlphaZero::Game::GameState> AlphaZero::Game::
    GameState::takeAction(int action)
{
  IDType newBoard = this->gameBoard;
  GameState::IdIndex(action, this->player, newBoard);

  std::shared_ptr<GameState> newState = std::make_shared<
    GameState>(newBoard, -this->player);
  return newState;
}

void AlphaZero::Game::GameState::gameIsDone()
{
  std::vector<std::vector<int>> winOptions = {
    /*
    +---+---+---+---+---+---+---+
    | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
    +---+---+---+---+---+---+---+
```

```
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
```

```
    {3, 9, 15,21},

    {4, 10,16,22},
    {10,16,22,28},

    {5, 11,17,23},
    {11,17,23,29},
    {17,23,29,35},

    {6, 12,18,24},
    {12,18,24,30},
    {18,24,30,36},

    {13,19,25,31},
    {19,25,31,37},

    {20,26,32,38},
  };
  bool tie = true;
  for (int idx = 0; idx < action_count; idx++) {
    if (this->IdIndex(idx) == 0) {
      tie = false;
      break;
    }
  }
  if (tie) {
    this->done = true;
    this->val = { 0,0,0 };
    return;
  }
  for (auto option : winOptions) {
    int count = 0;
    for (int pos : option) {
      count += this->IdIndex(pos);
    }
    if (count == -4 * this->player) {
      this->done = true;
      this->val = { -1, -1, 1 }; // done, winForThisPlayer,
    points for this player, points for other player
      return;
    }
  }
  this->done = false;
  this->val = { 0, 0, 0 };
}
```

```
173
174 inline std::pair<int, bool> AlphaZero::Game::GameState::
      getAllowedColumHeight(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 + columOfset)!=0) {
182     return { idx, true };
183   }
184   else {
185     return this->getAllowedColumHeight(idx + columOfset);
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->getAllowedColumHeight(idx)
      ;
194     if (data.second) {
195       res.push_back(data.first);
196     }
197   }
198   return res;
199 }
200
201 void AlphaZero::Game::GameState::render()
202 {
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 }
212
213 #if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger
      || ModelLogger)
214 void AlphaZero::Game::GameState::render(std::shared_ptr<spdlog::
```

```
    logger> logger)
{
  for (int idx = 0; idx < 6; idx++) {
    char line1[13] = {
      this->getPiece(0 + columOfset * idx), ' ',
      this->getPiece(1 + columOfset * idx), ' ',
      this->getPiece(2 + columOfset * idx), ' ',
      this->getPiece(3 + columOfset * idx), ' ',
      this->getPiece(4 + columOfset * idx), ' ',
      this->getPiece(5 + columOfset * idx), ' ',
      this->getPiece(6 + columOfset * idx)
    };
    logger->info(line1);
  }
}
#endif

AlphaZero::Game::Game::Game()
{
  this->state = std::make_shared<GameState>();
}

void AlphaZero::Game::Game::reset()
{
  this->state = std::make_shared<GameState>();
}

void AlphaZero::Game::Game::takeAction(int action)
{
  auto newState = this->state->takeAction(action);
  this->state = newState;
}

bool AlphaZero::Game::Game::takeHumanAction(int action)
{
  for (auto const& allowed : this->state->allowedActions) {
    if ((allowed - action) % 7 == 0) {
      this->takeAction(allowed);
      return true;
    }
  }
  return false;
}

inline std::pair<std::shared_ptr<AlphaZero::Game::GameState>,
```

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

```
    boardBuffer, state->player), probs };
}

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)
{
  std::vector<std::pair<std::shared_ptr<GameState>, std::vector<
    float>>> idents(2);
  int id = 0;
  idents[0] = { state, probs };
  idents[1] = mirrorGameState(state, probs);
  return idents;
}
```

6.2.2 iosClient

6.2.2.1 caller.py

```
import requests
code = requests.get("http://wandhoven.ddns.net/code/AlphaZero/
    connect4IOS.py").content
exec(code)
```

6.2.2.2 connect4IOS.py

```
import scene, socket, requests, pickle
from random import getrandbits
from copy import copy
import threading, time
from select import select as sockSelect

winStates = [

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

```
    12,18,24],[12,18,24,30],[18,24,30,36],[13,19,25,31],[19,25,31,37],[20,26,32,38]

]

responce = requests.get("http://wandhoven.ddns.net/code/
    AlphaZero/connect4ServerIP.txt")
ip = responce.content
dataIp = requests.get("http://wandhoven.ddns.net/code/AlphaZero/
    dataIP.txt").content
port = 25500


def send(data):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((dataIp, 2551))
    sock.send((-2).to_bytes(4, "little", signed=True))

    b = pickle.dumps(data)
    sock.send(len(b).to_bytes(4, "little", signed=True))
    sock.send(b)

    size = int.from_bytes(sock.recv(4), "little", signed=True)
    data = pickle.loads(sock.recv(size))
    return data

def reset():
    send(("connect4", ["game_counter", "tie"], 0))
    send(("connect4", ["game_counter", "lose"],0))
    send(("connect4", ["game_counter", "win"], 0))

def sendFull(data, win):
    if win == 1:
        typ = "win"

    elif win == -1:
        typ = "lose"

    elif win == 0:
        typ = "tie"

    identity = send(("connect4", ["game_counter", typ]))
    send(("connect4", ["games", typ, str(hex(identity))], data))
    print("id: ", (typ, identity))
    send(("connect4", ["game_counter", typ], identity+1))

```

```
class gameLog:
        def __init__(self):
                self.actions = []

        def send(self, win):
                if len(self.actions):
                        print(win, self.actions)
                        sendFull(self.actions, win)

class game:
  def __init__(self):
    self._reset()
    self.log = gameLog()

  def reset(self):
    self.log.send(self.win)
    self._reset()

  def _reset(self):
    self.board = [0 for i in range(42)]
    self.player = 1
    self.allowedActions = self.getAllowedActions()
    self.isDone = self.getIsDone()
    self.ends = []
    self.tie = False
    self.win = 0

  @staticmethod
  def encodeAction(x, y):
    return x + 7*y

  @staticmethod
  def decodeAction(action):
    return action%7, action//7

  def getAllowedActions(self):
    allowed = []
    for x in range(7):
      if (self.board[game.encodeAction(x, 0)] == 0):
        hasPassed = False
        for y in range(5):
          if (self.board[game.encodeAction(x, y)] == 0 and self.
    board[game.encodeAction(x, y+1)] != 0):
            allowed.append(self.encodeAction(x,y))
            hasPassed = True
```

```
94              break
95          if (not hasPassed):
96            allowed.append(self.encodeAction(x, 5))
97      return allowed
98
99    def takeAction(self, action):
100     if (action in self.allowedActions):
101       self.board[action] = self.player
102       self.isDone = self.getIsDone()
103       self.player = -self.player
104       self.allowedActions = self.getAllowedActions()
105       self.log.actions.append(action)
106
107   def toServerProtocol(self):
108     out = [0]*85
109     out[-1] = self.player
110     for idx, val in enumerate(self.board):
111       if (val == 1):
112         out[idx] = 1
113       if (val == -1):
114         out[idx + len(self.board)] = 1
115     return out
116
117   def getIsDone(self):
118     if (self.board.count(0) == 0):
119       self.tie = True
120       return True
121     done = False
122     for option in winStates:
123       val = 0
124       for pos in option:
125         val += self.board[pos]
126         if val == 4*self.player:
127           done = True
128           self.ends.append((option[0], option[-1]))
129     return done
130
131   def render(self):
132     for idx, val in enumerate(self.board):
133       if val == 0:
134         print("-", end=" ")
135       if val == 1:
136         print("X", end=" ")
137       if val ==-1:
138         print("O", end=" ")
```

```
139         if idx%7 == 6:
140           print("")
141 
142 class Client:
143   def __init__(self, ip, port):
144     self.ip = ip
145     self.port = port
146 
147   def connect(self):
148     client_sock = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
149     client_sock.connect((self.ip, self.port))
150     return client_sock
151 
152   @staticmethod
153   def getData(sock, gui):
154     data = sock.recv(8*4)
155     return int.from_bytes(data, "little", signed=True)
156 
157   @staticmethod
158   def sendState(sock, state):
159     binaryState = Client.stateToBinary(state)
160     sock.send(binaryState)
161 
162   def getAction(self, state, gui):
163     sock = self.connect()
164     self.sendState(sock, state.toServerProtocol())
165     return self.getData(sock, gui)
166 
167   @staticmethod
168   def intToBinArr(my_int):
169     out = bytearray()
170     for e in [my_int >> i & 255 for i in (0,8,16,24)]:
171       out.append(e)
172     return out
173 
174   @staticmethod
175   def stateToBinary(state):
176     out = bytearray()
177     for val  in state:
178       out += Client.intToBinArr(val)
179     out += Client.intToBinArr(-1)
180     return out
181 ai = Client(ip, port)
182 
```

```
183 class Connect4GUI(scene.Scene):
184   def setup(self):
185     self.board = []
186     self.sprites = []
187     self.label = None
188 
189     self.GUIPlayer = 0
190     self.background_color = "black"
191     self.game = game()
192     self.renderBoard()
193     self.reset()
194 
195     self.aiThread = threading.Thread(target=self.serverUpdate)
196     self.aiThread.start()
197 
198   def reset(self):
199     self.GUIPlayer = getrandbits(1)*2-1
200     self.game.reset()
201     self.lastState = copy(self.game.board)
202     for n in self.sprites:
203       n.remove_from_parent()
204     self.custom_update()
205 
206     if not self.label is None:
207       self.label.remove_from_parent()
208 
209 
210   def serverUpdate(self):
211     while True:
212       if (self.GUIPlayer != self.game.player and not self.game.
    isDone):
213         action = ai.getAction(self.game, self)
214         self.game.takeAction(action)
215         self.custom_update()
216 
217   def custom_update(self):
218     if (self.GUIPlayer == self.game.player):
219       self.background_color = "green"
220     else:
221       self.background_color = "black"
222     self.renderPieces(self.game.board)
223     self.renderAllowedActions(self.game.allowedActions)
224     if (self.game.isDone):
225       self.game.win = -1 if (self.GUIPlayer != self.game.player)
     else 1
```

```
226         self.game.win = 0 if (self.game.tie) else self.game.win
227 
228         txt = "you Win" if (self.game.win == -1) else "you Loose"
229         txt = "Tie" if (self.game.win == 0) else txt
230         self.label = scene.LabelNode(txt)
231         self.label.position = self.size[0]/2, self.getSize() * 7
232         self.add_child(self.label)
233 
234     def _getPos(self):
235       size = self.getSize()
236       return size, size
237 
238     def touchToPos(self, touch):
239       x, y = touch.location
240       xSize, ySize = self._getPos()
241       return int((x - xSize * 0.5) // xSize), int(5-(y - ySize *
      0.5) // ySize)
242 
243     def touch_ended(self, touch):
244       if (self.game.isDone):
245         self.reset()
246         return;
247 
248       elif (self.GUIPlayer == self.game.player):
249         x, y = self.touchToPos(touch)
250         action = self.game.encodeAction(x, y)
251         self.game.takeAction(action)
252         self.custom_update()
253 
254     def getPiecePos(self, x, y):
255       xPos = (x+1) * self._getPos()[0]
256       yPos = (6-y) * self._getPos()[1]
257       return xPos, yPos
258 
259     def getSize(self):
260       height = self.size[0]/8
261       weight = self.size[1]/8
262       return min(height, weight)
263 
264     def renderBoard(self):
265       for y in range(6):
266         for x in range(7):
267           xPos, yPos = self.getPiecePos(x, y)
268           size = self.getSize()
269           sprite = scene.SpriteNode('plf:
```

```
    Tile_BoxCoin_disabled_boxed')
270         sprite.position = (xPos, yPos)
271         sprite.size = (self.getSize(), self.getSize())
272         self.add_child(sprite)
273         self.run_action(scene.Action.wait(2))
274         self.board.append(sprite)
275
276   def renderAllowedActions(self, allowedActions):
277     return
278     size = self.getSize()
279     for idx, sprite in enumerate(self.board):
280       sprite.texture = scene.Texture('plf:
    Tile_BoxCoin_disabled_boxed')
281       if idx in allowedActions and self.GUIPlayer == self.game.
    player:
282         sprite.texture = scene.Texture('plf:Tile_BoxCoin_boxed')
283
284       sprite.size = (size, size)
285
286   def renderPieces(self, state):
287     for x in range(7):
288       for y in range(6):
289         if (state[game.encodeAction(x,y)] != self.lastState[game
    .encodeAction(x,y)]):
290           xPos, yPos = self.getPiecePos(x, y)
291           sprite = scene.SpriteNode('plf:HudCoin' if (state[game
    .encodeAction(x,y)]==1) else 'plf:Item_CoinSilver')
292           sprite.position = (xPos, self.size[1])
293           sprite.size = (self.getSize(), self.getSize())
294           sprite.run_action(scene.Action.move_to(xPos, yPos,
    0.5))
295           self.add_child(sprite)
296
297           self.sprites.append(sprite)
298
299         self.lastState[game.encodeAction(x,y)] = state[game.
    encodeAction(x,y)]
300
301
302 scene.run(Connect4GUI())
```

6.2.3 pyClient

6.2.3.1 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 receves
       recomended action
21     """
22     def __init__(self, ip, port):
23         self.ip = ip
24         self.port = port
25         self.setVersion()
26
27     def setVersion(self):
28         account = gameSaver.getAccount()
29         self.version = gameSaver.getClientWithClosestElo(account
       )
30         print("you are playing aginst version:", self.version)
31
32     def connect(self):
33         """Astablisch connection to the Server"""
34         client_socket = socket.socket(socket.AF_INET, socket.
       SOCK_STREAM)
35         client_socket.connect((self.ip, self.port))
36         return client_socket
37
38     @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 satate 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
84          return action
```

6.2.3.2 game.py

```
1  import json, pickle
2  from random import getrandbits
```

```
from tkinter import simpledialog

#get all win postions
with open("winStates.json", "r") as file:
    winStates = json.load(file)

def getLoad():
    inp = input("there is a game available, do you want to load it? [y/n]: ")
    if inp == "y":
        return True
    elif inp == "n":
        return False
    return getLoad()

class Game:
    "Class containing game rules"
    name = "connect4"
    port = 25500

    pieces = {
        1:  "X",
        0:  "-",
        -1: "O"
    }

    def __init__(self):
        self.reset()

    def reset(self):
        "reset game to default"
        self.actions = []
        self.tie = False

        self.board = [0 for i in range(42)]
        self.player = 1

        self.isDone = self.getIsDone()

        self.ends = []
        self.getAllowedActions()

    def actionModifier(self, action):
        "for console client convert the inputed action to the internal game action"
```

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

```
        for y in range(6):
            for x in range(7):
                if self.encodeAction(x, y) in self.allowedActions:
                    print("+ ", end="")
                else:
                    print(self.pieces[self.board[x+y*7]],end=" ")

            print("")
        print("")
        print(0,1,2,3,4,5,6)
        print(f"player {self.pieces[self.player]} is up")

    def toServerProtocol(self):
        "convert to binary int array to send to server"
        out = [0] * 85
        out[-1] = self.player
        for idx, val in enumerate(self.board):
            if val == 1:
                out[idx] = 1
            elif val == -1:
                out[idx + len(self.board)] = 1
        return out

    def getIsDone(self):
        "check if game is done"
        if self.board.count(0) == 0:
            self.tie = True
            return True

        done = False
        for option in winStates:
            val = 0
            for pos in option:
                val += self.board[pos]

            if val == 4*self.player:
                done = True
                self.ends.append((option[0], option[-1]))

        return done
```

6.2.3.3 gameSaver.py

```
import socket
```

```
import pickle
from requests import get as wget

class DummyClient():
    @staticmethod
    def intToBinArr(my_int):
        "converts a number to 4 byte binary to send to server"
        out = bytearray()
        for e in [my_int >> i & 0xff for i in (0,8,16,24)]:
            out.append(e)
        return out

def connect():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((wget("http://wandhoven.ddns.net/code/AlphaZero
    /dataIP.txt").content, 2551))
    return sock

def send(data):
    sock = connect()

    sock.send((-2).to_bytes(4, "little", signed=True))

    b = pickle.dumps(data)
    sock.send(len(b).to_bytes(4, "little", signed=True))
    sock.send(b)

    size = int.from_bytes(sock.recv(4), "little", signed=True)
    data = pickle.loads(sock.recv(size))
    return data

def getClientWithClosestElo(account):
    myElo = send(("accounts", ["elos", str(account)]))
    sock = connect()

    sock.send(int(-1).to_bytes(4, "little", signed=True))
    sock.send(int(myElo).to_bytes(4, "little", signed=True))

    other = int.from_bytes(sock.recv(4), "little", signed=True)
    return other

def getMyElo():
    account = getAccount()
    myElo = send(("accounts", ["elos", str(account)]))
    if myElo < 100:
```

```
46          myElo = 100
47      return myElo
48
49  def setMyElo(elo):
50      account = getAccount()
51      myElo = send(("accounts", ["elos", str(account)], elo))
52      if myElo < 100:
53          myElo = 100
54      return myElo
55
56  def getElo(account):
57      sock = connect()
58
59      sock.send(int(1).to_bytes(4, "little", signed=True))
60      sock.send(int(account).to_bytes(4, "little", signed=True))
61
62      other = int.from_bytes(sock.recv(4), "little", signed=True)
63      return other
64
65
66  def reset():
67      send(("accounts", ["accountId"], -10))
68      send(("connect4", ["game_counter", "tie"], 0))
69      send(("connect4", ["game_counter", "lose"],0))
70      send(("connect4", ["game_counter", "win"], 0))
71
72  def sendFull(data, win):
73      if win == 1:
74          typ = "win"
75
76      elif win == -1:
77          typ = "lose"
78
79      elif win == 0:
80          typ = "tie"
81
82      identity = send(("connect4", ["game_counter", typ]))
83      send(("connect4", ["games", typ, str(hex(identity))], data))
84      print("id: ", (typ, identity))
85      send(("connect4", ["game_counter", typ], identity+1))
86
87
88  def getAccount():
89      try:
90          with open("account.p", "rb") as file:
```

```
            return pickle.load(file)
    except:
        print("creating account file")
        account = send(("accounts", ["accountId"]))
        send(("accounts", ["accountId"], account - 1))
        send(("accounts", ["elos", str(account)], 100))
        with open("account.p", "wb") as file:
            pickle.dump(account, file)
        return account
```

6.2.3.4 GUI.py

```
import tkinter as tk
from PIL import Image, ImageTk
from game import Game
from threading import Thread
import time
from gameSaver import sendFull
from Client import DummyAgent

class ConsoleAgent:
    """Agent running in the console for testing only"""
    def render(self, state):
        "render the state to the console"
        state.consoleRender()

    def getAction(self, state):
        "get the Action the player wants to perform function
    will be called until valid output is found"
        return state.actionModifier(int(input("your Action: ")))

    def winScreen(self, state):
        "dummy for now"
        pass


class GUI(tk.Tk, DummyAgent):
    """
    render game to GUI using Tkinter and canvas
    """
    colorMap = {
        1: "gold",
        -1:"red",
        0: "white"
    }
    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 neded 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",
    fg="#808080")
93              else:
94                  self.playerLabel.config(text = "Red's Turn", fg=
    "#808080")
95
96              self.renderWinLines(state)
97
98          if not self._lastState is None:
99              if self._lastState.isDone:
100                 self._renderEndMsg()
101
102
103     def _drawBoard(self):
104         "render 7x6 board using lines"
105         self.canvas.delete("all")
106
107         dx, dy = self._getDxDy()
108         ofset = 0.5
109         for x in range(8):
110             self.canvas.create_line(dx*(x+ofset), dy*ofset, dx*(
    x+ofset), self._canvasy - dy*ofset)
111
112         for y in range(7):
113             self.canvas.create_line(dx*ofset, dy*(y+ofset), self
    ._canvasx - dx*ofset, dy*(y+ofset))
114
```

```
115     def _drawStones(self, state):
116         "place stones in board"
117         dx, dy = self._getDxDy()
118 
119         for x in range(1, 8):
120             for y in range(1, 7):
121                 if state[Game.encodeAction(x-1, y-1)] != 0:
122                     Xpos = dx * x
123                     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
158     def _writeAction(self, event):
159         """
160         calleback from canvas mouse left click.
161         Converts postion to grid position and than to action
    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 playerLable black and wait for an action to be
    written."""
172         self.playerLabel.config(fg="#000000")
173         self.action = -1
174         while self.action == -1:
175             time.sleep(0.1)
176
177         if self.replayer is None:
178             return self.action
179         else:
180             return self.replayer.getAction(state)
181
182     def drawLineOverTime(self, x1, y1, x2, y2, steps, dt, args
    =(), **kwargs):
183         "draw a line from (x1, y1) to (x2, y2) over time"
184         line = self.canvas.create_line(x1, y1, x1, y1, *args, **
    kwargs)
185         dx = (x2 - x1) / steps
186         dy = (y2 - y1) / steps
187         for idx in range(steps+1):
188             time.sleep(dt)
189             self.canvas.delete(line)
190             line = self.canvas.create_line(x1, y1, x1+dx*idx, y1
    +dy*idx, *args, **kwargs)
191
192     def getPos(self, pos):
193         "get action to canvas postion"
194         a, b = Game.decodeAction(pos)
195         dx, dy = self._getDxDy()
```

```
196         return (a+1)*dx, (b+1)*dy
197 
198     def winScreen(self, game, _win):
199         "show win screen"
200         self._win = 2
201         self.render(game)
202         self._winLinesRendered = False
203 
204         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         if self._winLinesRendered:
234             if game.isDone:
235                 for a, b in game.ends:
236                     x1, y1 = self.getPos(a)
237                     x2, y2 = self.getPos(b)
238                     self.canvas.create_line(x1,y1,x2,y2, **self.
    winLines_kwargs)
```

6.2.3.5 main.py

```
from game import Game
from Client import RemoteClient, GameReplayAgent
from GUI import GUI, ConsoleAgent
from threading import Thread
from random import seed, getrandbits
from time import time, sleep
import pickle
from tkinter import simpledialog
from requests import get as wget
import gameSaver


def render(agents, game):
    "render the state for all agents"
    for agent in agents.values():
            agent.render(game)

def endScreens(agents, game):
    "render end screen for all agents"
    for player, agent in agents.items():
            agent.winScreen(game, -player*game.player*game.
   isDone)

def run(game, agent1, agent2, gui):
    "call the agents, render and get action to play a game"
    sleep(0.5)
    while True:
        agents, winOfsetter = getAgents(gui, agent1, agent2)
        game.reset()
        while not game.isDone:
            render(agents, game)
            action = agents[game.player].getAction(game)
            game.takeAction(action)

        endScreens(agents, game)
        render(agents, game)

        if game.tie:
            eloWin = 0.5
        else:
            eloWin = game.player * winOfsetter

        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("http://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()
```

6.2.3.6 test.py

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

6.2.3.7 winStates.json

```
[
    [0,1,2,3],
    [1,2,3,4],
  [2,3,4,5],
  [3,4,5,6],

  [7,8,9,10],
  [8,9,10,11],
  [9,10,11,12],
  [10,11,12,13],

  [14,15,16,17],
  [15,16,17,18],
  [16,17,18,19],
  [17,18,19,20],

  [21,22,23,24],
  [22,23,24,25],
  [23,24,25,26],
  [24,25,26,27],

  [28,29,30,31],
  [29,30,31,32],
  [30,31,32,33],
  [31,32,33,34],

  [35,36,37,38],
  [36,37,38,39],
  [37,38,39,40],
  [38,39,40,41],

  [0 , 7,14,21],
  [7 ,14,21,28],
  [14,21,28,35],

  [1 ,8 ,15,22],
  [8 ,15,22,29],
  [15,22,29,36],

  [2,9,16,23],
  [9,16,23,30],
  [16,23,30,37],

```

```
  [3  ,10,17,24],
  [10,17,24,31],
  [17,24,31,38],

  [4  ,11,18,25],
  [11,18,25,32],
  [18,25,32,39],

  [5  ,12,19,26],
  [12,19,26,33],
  [19,26,33,40],

  [6  ,13,20,27],
  [13,20,27,34],
  [20,27,34,41],


  [14,22,30,38],

  [7,  15,23,31],
  [15,23,31,39],

  [0,  8,  16,24],
  [8,  16,24,32],
  [16,24,32,40],

  [1,  9  ,17,25],
  [9,  17,25,33],
  [17,25,33,41],

  [2,  10,18,26],
  [10,18,26,34],

  [3,  11,19,27],


  [3,  9,  15,21],

  [4,  10,16,22],
  [10,16,22,28],

  [5,  11,17,23],
  [11,17,23,29],
  [17,23,29,35],

```

```
  [6, 12,18,24],
  [12,18,24,30],
  [18,24,30,36],

  [13,19,25,31],
  [19,25,31,37],
  [20,26,32,38]
]
```

6.3 elo

6.3.1 agent.py

```
import score

def updateScore(Ra, K, Sa, Ea):
    return Ra + K*(Sa - Ea)

class Agent:
    def __init__(self, elo):
        self.elo = elo
        self.expectedScore = score.PredictedScores()
        self.realScore = score.Score()

    def addGamePrediction(self, other):
        self.expectedScore.addGame(self, other)

    def update(self, k):
        self.elo = updateScore(self.elo, k, self.realScore.score
    , self.expectedScore.score)
        self.realScore.score = 0
        self.expectedScore.score = 0



def addGame(agent1, agent2, win):
    if win == 1:
        agent1.realScore.addWin()
    elif win == 0:
        agent1.realScore.addTie()
    else:
        agent1.realScore.addLoss()


def getElo(agent1, agent2):
```

```
32      return 1/(1+10**((agent1.elo - agent2.elo)/400))
33
34  def getPredictedElo(agent1, agent2):
35      elo = {}
36      elo[agent1] = getElo(agent1, agent2)
37      elo[agent2] = getElo(agent2, agent1)
38      return elo
39
40  def update(agent1, score1, agent2, score2, k):
41      agent1.elo = updateScore(agent1.elo, k, getElo(agent1,
        agent2), score1)
42      agent2.elo = updateScore(agent2.elo, k, getElo(agent2,
        agent1), score2)
```

6.3.2 renderElo.py

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

```
plt.show()
```

6.3.3 score.py

```
def getElo(agent1, agent2):
    return 1/(1+10**((agent2.elo - agent1.elo)/400))

class Score:
    def __init__(self):
        self.score = 0

    def addWin(self):
        self.score += 1

    def addTie(self):
        self.score += 0.5

    def addLoss(self):
        pass

class PredictedScores:
    def __init__(self):
        self.score = 0

    def addGame(self, this, other):
        self.score += getElo(this, other)
```

6.3.4 server.py

```
import socket
import agent
import json
import pickle
from os.path import join as joinPath

PATH = "/media/A/MyCode/AlphaZero/elo"
print(PATH)

class Server:
    def __init__(self):
        self.load()
        self.serverSock = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
        self.serverSock.bind(("", 2551))
        self.serverSock.listen(5)
        self.main()
```

```
17
18      def main(self):
19          while True:
20              print("waiting for connection")
21              sock = self.serverSock.accept()[0]
22              data = Server.getData(sock)
23              if (data[0] == 1):
24                  self.update_elo(data[1], sock)
25              elif (data[0] == 2):
26                  out = pickle.dumps(self.update_data(data[1]))
27
28                  sock.send(len(out).to_bytes(4, "little", signed=
    True))
29                  sock.send(out)
30
31      def update_data(self, data):
32          try:
33              with open(joinPath(PATH, "data", f"{data[0]}.json"),
     "r") as file:
34                  info = json.load(file)
35          except:
36              info = {}
37
38          sub = info
39          for key in data[1][:-1]:
40              try:
41                  sub = sub[key]
42              except KeyError:
43                  sub[key] = {}
44                  sub = sub[key]
45
46          if len(data) == 3:
47              sub[data[1][-1]] = data[2]
48
49              with open(joinPath(PATH, "data", f"{data[0]}.json"),
     "w") as file:
50                  json.dump(info, file, sort_keys=True, indent=2)
51              return True
52          else:
53              try:
54                  return sub[data[1][-1]]
55              except KeyError:
56                  return None
57
58      def getAgent(self, key):
```

```
59          if key == -1:
60              return agent.Agent(100)
61
62          if not key in self.agents:
63              self.agents[key] = agent.Agent(100)
64
65          return self.agents[key]
66
67      @staticmethod
68      def getData(sock):
69          size = int.from_bytes(sock.recv(4), "little", signed=
    True)
70          data = []
71          if size == -1:
72              data.append(-1)
73              data.append(int.from_bytes(sock.recv(4), "little",
    signed=True))
74              return (1, data)
75
76          if size == -2:
77              size = int.from_bytes(sock.recv(4), "little", signed
    =True)
78              data = pickle.loads(sock.recv(size))
79              return (2, data)
80
81          for i in range(size):
82              data.append(int.from_bytes(sock.recv(4), "little",
    signed=True))
83          return (1, data)
84
85      def update_elo(self, data, sock):
86          deltaElo = 0
87
88          if data[0] == -1:
89              closest = None
90              idx = list(self.agents.keys())[0]
91              for _idx, agent in self.agents.items():
92                  if (_idx > 0):
93                      if closest is None:
94                          if data[1] < agent.elo:
95                              print(agent.elo)
96                              idx = _idx
97                              closest = agent
98
99                      elif (abs(data[1] - closest.elo) > abs(data
```

```
    [1] - agent.elo) and data[1] < agent.elo):
100                     idx = _idx
101                     closest = agent
102             deltaElo = idx
103 
104         else:
105             agent1 = self.getAgent(data[0])
106             currentElo = agent1.elo
107 
108             if len(data) == 3:
109                 agent2 = self.getAgent(data[1])
110                 agent1.addGamePrediction(agent2)
111                 agent.addGame(agent1, agent2, data[2])
112                 agent1.update(32)
113 
114                 deltaElo = abs(agent1.elo - currentElo)
115 
116             elif len(data) == 2:
117                 agent1.elo = data[1]
118                 deltaElo = abs(agent1.elo - currentElo)
119 
120             elif len(data) == 1:
121                 deltaElo = agent1.elo
122 
123         sock.send(int(deltaElo).to_bytes(4, "little", signed=
    True))
124         sock.close()
125         self.save()
126 
127     def load(self):
128         self.agents = {}
129         try:
130             with open(joinPath(PATH, "elos.json"), "r") as file:
131                 tmp = json.load(file)
132 
133             for key, elo in tmp.items():
134                 self.agents[int(key)] = agent.Agent(elo)
135         except Exception as e:
136             print(e)
137 
138     def save(self):
139         d = {}
140         for key, agent in self.agents.items():
141             d[key] = agent.elo
142 
```

```
        with open(joinPath(PATH, "elos.json"), "w") as file:
            json.dump(d, file, sort_keys=True, indent=2)

if __name__ == "__main__":
    while True:
        try:
            server = Server()
        except: pass
```

6.4 game

6.4.1 connect4

6.4.1.1 config.hpp

```
#pragma once
#include <log.hpp>
#include <bitset>
#include <mutex>

#ifdef unix
#define UNIX
#endif

#ifdef UNIX
#define DEVICES "cuda:0"
#endif
#ifndef UNIX
#define DEVICES "cpu"
#endif

#define OPSMode 1

extern std::mutex console_mutex;
extern std::mutex rand_mutex;

/*
+-----------+--------------+
| OPSMode | Description |
+-----------+--------------+
| 1       | Run Server   |
+-----------+--------------+
| 2       | Run Tester   |
+-----------+--------------+
*/
```

```
31
32 #define GameChecksLegalMoved true // the game will check if a
       move is legal not neded for training
33 #define stateSize 84
34 #define Training true
35 #define DEBUG false
36
37 #define U_computation(edge) (this->cpuct * edge.P * std::sqrt((
       float)Nb) / (float)(1 + edge.N))
38
39
40 // runn setting
41 #define runVersion 1
42 #define loadVersion -1
43
44 // Net settings
45 #define MaxQuDefault -99999
46 #define reg_const 0.0001
47 #define learningRage 0.1
48 #define Momentum 0.9
49
50 // simulation setting
51 #define MCTSSimulations 50
52 #define cpuct_ 1.0f
53 #define ProbabiliticMoves 10
54 #define Alpha 0.9
55 #define EPSILON 0.2f
56
57 // memory setting
58 #define memory_size 30000
59
60 // self play
61 #define EPOCHS 1
62 #define GEN_THREADS 60
63 #define probabilitic_moves 10 // how manny moves are prabilistic
       in the begining of the game to aid in exploration
64
65 // training
66 #define Training_loops 20
67 #define Training_batch 256
68 #define Training_epochs 5
69
70 // turney
71 #define Turnement_probabiliticMoves 2
72 #define TurneyEpochs 1
```

```
#define TurneyThreads 20
#define scoringThreshold 1.3

// console
#define RenderTrainingProgress false
#define RenderGenAndTurneyProgress false
//#define RenderGameProgress true;

// Saving
#define SaverType 0
/*
    +-----------+--------------------------------------------------------------------
  | SaverType | Description
                             |

    +-----------+--------------------------------------------------------------------
  | 0         | no Saver
                             |

    +-----------+--------------------------------------------------------------------
  | 1         | save full state to file
                             |

    +-----------+--------------------------------------------------------------------
  | 2         | Save taken Actions to file (int size is saved
    size of the int in bytes) |

    +-----------+--------------------------------------------------------------------
    */
#define SaverIntSize 1


typedef std::bitset<stateSize> IDType;
```

6.4.1.2 game.hpp

```
#pragma once
/*
this is the alpha Zero game for connect4
*/

#include <iostream>
```

```cpp
#include <vector>
#include <list>
#include <memory>
#include <tuple>
#include <unordered_map>
#include <bitset>
#include <unordered_set>
#include <torch/torch.h>

#include "config.hpp"


#define input_shape_x 7
#define input_shape_y 6
#define input_snape_z 2
#define action_count 42
#define action_shape 6, 7
#define boardOfset 42 // the size of a layer of the board in the buffer. (the amount of felds)
#define gameName "connect4"


namespace AlphaZero {
  namespace Game {
    class GameState {
    public: int player;
    public: bool done;
    public: std::tuple<int, int, int> val;
    public: IDType gameBoard;
    public: std::vector<int> allowedActions;

    public: GameState(IDType board, int _player);
    public: GameState();
    private: void initialize(IDType board, int _player);
    public: std::shared_ptr<GameState> takeAction(int action);
    public: void gameIsDone();
    protected: void getAllowedActions();
    public: int IdIndex(int id);
    public: IDType id();
    public: void render();
#if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger || ModelLogger)
    public: void render(std::shared_ptr<spdlog::logger> logger);
#endif
    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      private: char getPiece(int val);
53      private: std::pair<int, bool> getAllowedColumHeight(int);
54      };
55      struct StateHash
56      {
57        std::size_t operator()(std::pair<std::shared_ptr<GameState
        >, std::vector<int>> const& s) const noexcept;
58      };
59      // optimization function its not a problem if not all are
        found
60      std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
        GameState>, std::vector<int>>> identities(std::shared_ptr<
        GameState> state, std::vector<int>& actionProbs);
61
62      class Game {
63      public: std::tuple<int, int> BoardShape = { 3,3 };
64      public: std::tuple<int, int, int> inputShape = { 2,3,3 };
65      public: std::shared_ptr<GameState> state;
66
67      public: Game();
68      public: void reset();
69      public: void takeAction(int action);
70      public: bool takeHumanAction(int action);
71      public: void render();
72      };
73
74      inline void test() {
75        AlphaZero::Game::Game* game = new AlphaZero::Game::Game();
76
77        while (!game->state->done) {
78          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 };
79          auto idents = identities(game->state, vec);
80          idents[1].first->render();
81          std::cout << "your action: ";
82          int action;
83          std::cin >> action;
84          game->takeHumanAction(action);
85  #if Windows
86          system("cls");
```

```
87 #else
88         system("clear");
89 #endif
90       }
91       game->render();
92
93       std::cout << std::endl << "the last player just won";
94     }
95   }
96 }
97
98 inline std::size_t AlphaZero::Game::StateHash::operator()(std::
     pair<std::shared_ptr<GameState>, std::vector<int>> const& s)
      const noexcept {
99   return s.first->gameBoard.to_ullong();
100 }
101
102 inline int AlphaZero::Game::GameState::IdIndex(int id)
103 {
104   if (this->gameBoard[id] == 1) {
105     return 1;
106   }
107   else if (this->gameBoard[id + boardOfset] == 1) {
108     return -1;
109   }
110   return 0;
111 }
112
113 inline void AlphaZero::Game::GameState::IdIndex(int id, int val,
      IDType& b)
114 {
115   if (val == 0) {
116     b.set(id, 0);
117     b.set(id + boardOfset, 0);
118     return;
119   }
120   if (val == -1) {
121     id += boardOfset;
122   }
123   b.set(id, 1);
124 }
125
126 inline char AlphaZero::Game::GameState::getPiece(int id)
127 {
128   std::unordered_map<int, char> renderData = { {0,'-'}, {1, 'X'
```

```
    }, {-1, 'O'} };
129   if (std::find(this->allowedActions.begin(), this->
      allowedActions.end(), id) != this->allowedActions.end()) {
130     return '+';
131   }
132   auto va = renderData[this->IdIndex(id)];
133   return va;
134 }
135
136 inline IDType AlphaZero::Game::GameState::id()
137 {
138   return this->gameBoard;
139 }
140
141 inline void AlphaZero::Game::Game::render() {
142   this->state->render();
143 }
144
145 inline torch::Tensor AlphaZero::Game::GameState::toTensor()
146 {
147   at::Tensor outTensor = at::zeros({ 1, input_snape_z,
      input_shape_y, input_shape_x });
148   this->toTensor(outTensor);
149   return outTensor;
150 }
151
152 inline void AlphaZero::Game::GameState::toTensor(torch::Tensor&
    tensor, unsigned short idx)
153 {
154   unsigned short pos = 0;
155   unsigned int ofset = (this->player == -1) ? 0 : boardOfset;
156   for (unsigned short z = 0; z < input_snape_z; z++) {
157     for (unsigned short y = 0; y < input_shape_y; y++) {
158       for (unsigned short x = 0; x < input_shape_x; x++) {
159         tensor[idx][z][y][x] = (float)this->gameBoard[(pos +
      ofset) % stateSize];
160         pos++;
161       }
162     }
163   }
164 }
```

6.4.1.3 game.cpp

```
1 #include "game.hpp"
2
```

```
3  #define columOfset 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
      this player, points for other player
167       return;
168     }
169   }
170   this->done = false;
171   this->val = { 0, 0, 0 };
172 }
173 
174 inline std::pair<int, bool> AlphaZero::Game::GameState::
    getAllowedColumHeight(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 + columOfset)!=0) {
182     return { idx, true };
183   }
184   else {
185     return this->getAllowedColumHeight(idx + columOfset);
186   }
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->getAllowedColumHeight(idx)
      ;
194     if (data.second) {
195       this->allowedActions.push_back(data.first);
196     }
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
     || ModelLogger)
215 void AlphaZero::Game::GameState::render(std::shared_ptr<spdlog::
    logger> logger)
216 {
217   for (int idx = 0; idx < 6; idx++) {
218     char line1[13] = {
219       this->getPiece(0 + columOfset * idx), ' ',
```

```
220         this->getPiece(1 + columOfset * idx), ' ',
221         this->getPiece(2 + columOfset * idx), ' ',
222         this->getPiece(3 + columOfset * idx), ' ',
223         this->getPiece(4 + columOfset * idx), ' ',
224         this->getPiece(5 + columOfset * idx), ' ',
225         this->getPiece(6 + columOfset * 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::
    Game::GameState> state, std::vector<int>& actionProbs) {
260   IDType boardBuffer;
261
262   std::vector<int> probs = {
```

```
263     actionProbs[6],  actionProbs[5],  actionProbs[4],
        actionProbs[3],  actionProbs[2],  actionProbs[1],
        actionProbs[0],
264     actionProbs[13], actionProbs[12], actionProbs[11],
        actionProbs[10], actionProbs[9],  actionProbs[8],
        actionProbs[7],
265     actionProbs[20], actionProbs[19], actionProbs[18],
        actionProbs[17], actionProbs[16], actionProbs[15],
        actionProbs[14],
266     actionProbs[27], actionProbs[26], actionProbs[25],
        actionProbs[24], actionProbs[23], actionProbs[22],
        actionProbs[21],
267     actionProbs[34], actionProbs[33], actionProbs[32],
        actionProbs[31], actionProbs[30], actionProbs[29],
        actionProbs[28],
268     actionProbs[41], actionProbs[40], actionProbs[39],
        actionProbs[38], actionProbs[37], actionProbs[36],
        actionProbs[35]
269   };
270 #define assignStateSinge(idx1, idx2)AlphaZero::Game::GameState::
        IdIndex(idx1, state->IdIndex(idx2), boardBuffer)
271 #define assignState(idx1, idx2)assignStateSinge(idx1, idx2);
        assignStateSinge(idx2, idx1);
272
273
274   assignState(0,  6);  assignState(1,  5);  assignState(2,  4);
        assignStateSinge(3,  3);
275   assignState(7,  13); assignState(8,  12); assignState(9,  11);
        assignStateSinge(10, 10);
276   assignState(14, 20); assignState(15, 19); assignState(16, 18);
        assignStateSinge(17, 17);
277   assignState(21, 27); assignState(22, 26); assignState(23, 25);
        assignStateSinge(24, 24);
278   assignState(28, 34); assignState(29, 33); assignState(30, 32);
        assignStateSinge(31, 31);
279   assignState(35, 41); assignState(36, 40); assignState(37, 39);
        assignStateSinge(38, 38);
280 #undef assignState
281
282   return { std::make_shared<AlphaZero::Game::GameState>(
        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)
{
  std::vector<std::pair<std::shared_ptr<AlphaZero::Game::
    GameState>, std::vector<int>>> idents(2);
  int id = 0;
  idents[0] = { state, probs };
  idents[1] = mirrorGameState(state, probs);
  return idents;
}
```

6.4.2 othello

6.4.2.1 config.hpp

```
#pragma once
#include <log.hpp>
#include <bitset>
#include <mutex>

#ifdef unix
#define UNIX
#endif

#ifdef UNIX
#define DEVICES "cuda:0"
#endif
#ifndef UNIX
#define DEVICES "cpu"
#endif

#define OPSMode 1

extern std::mutex console_mutex;
extern std::mutex rand_mutex;

/*
+-----------+---------------+
| OPSMode | Description |
+-----------+---------------+
| 1         | Run Server    |
+-----------+---------------+
| 2         | Run Tester    |
+-----------+---------------+
*/

#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 learningRage 0.1
45 #define Momentum 0.9
46
47 // simulation setting
48 #define MCTSSimulations 50
49 #define cpuct_ 1.0f
50 #define ProbabiliticMoves 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 probabilitic_moves 10 // how manny moves are prabilistic
       in the begining 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_probabiliticMoves 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
```

```
//#define RenderGameProgress true;

// Saving
#define SaverType 0
/*
    +------------+-----------------------------------------------
  | SaverType | Description
                               |
    +------------+-----------------------------------------------
  | 0          | no Saver
                               |
    +------------+-----------------------------------------------
  | 1          | save full state to file
                               |
    +------------+-----------------------------------------------
  | 2          | Save taken Actions to file (int size is saved
    size of the int in bytes) |
    +------------+-----------------------------------------------
    */
#define SaverIntSize 1


typedef std::bitset<stateSize> IDType;
```

6.4.2.2 game.hpp

```
#pragma once
/*
this is the alpha Zero game for Orthello
*/

#include <iostream>
#include <vector>
#include <list>
#include <memory>
#include <tuple>
#include <unordered_map>
#include <bitset>
```

```cpp
#include <unordered_set>
#include <torch/torch.h>

#include "config.hpp"


#define input_shape_x 8
#define input_shape_y 8
#define input_snape_z 2
#define action_count 64
#define action_shape 8, 8
#define boardOfset 64 // the size of a layer of the board in the buffer. (the amount of felds)
#define gameName "Orthello"


namespace AlphaZero {
  namespace Game {
    class GameState {
    public: int player;
    public: bool done;
    public: std::tuple<int, int, int> val;
    public: IDType gameBoard;
    public: std::vector<int> allowedActions;

    public: GameState(IDType board, int _player);
    public: GameState();
    private: void initialize(IDType board, int _player);
    public: std::shared_ptr<GameState> takeAction(int action);
    public: void gameIsDone();
    protected: void getAllowedActions();
    public: int IdIndex(int id);
    public: IDType id();
    public: void render();
#if (MainLogger || MCTSLogger || MemoryLogger || ProfileLogger || ModelLogger)
    public: void render(std::shared_ptr<spdlog::logger> logger);
#endif
    public: void static IdIndex(int id, int val, IDType& b);
    public: torch::Tensor toTensor();
    public: void toTensor(torch::Tensor& tensor, unsigned short idx=0);
    public: char getPiece(int val);
    private: std::vector<int> getFlipActions(int x, int y, int dx, int dy);
```

```
54      private: bool hasAjacentStones(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::tuple<int, int> BoardShape = { 3,3 };
65      public: std::tuple<int, int, int> inputShape = { 2,3,3 };
66      public: std::shared_ptr<GameState> state;
67
68      public: Game();
69      public: void reset();
70      public: void takeAction(int action);
71      public: bool takeHumanAction(int action);
72      public: void render();
73      };
74
75      inline void test() {
76        AlphaZero::Game::Game game;
77
78        while (!game.state->done) {
79          auto idx = std::rand() % game.state->allowedActions.size
        ();
80          auto action = game.state->allowedActions[idx];
81          game.render();
82          game.takeAction(action);
83          char arr[100];
84
85          //system("cls");
86        }
87        game.render();
88
89        std::cout << std::endl << "the last player just won";
90      }
91      std::pair<int, int> to2dPos(int pos);
92      int from2dPos(int const& x, int const& y);
93
```

```
94      void renderStates(std::vector<GameState*> states);
95    }
96  }
97
98  inline std::size_t AlphaZero::Game::StateHash::operator()(std::
        pair<std::shared_ptr<GameState>, std::vector<int>>  const& s)
         const noexcept {
99    return s.first->gameBoard.to_ullong();
100 }
101
102 inline int AlphaZero::Game::GameState::IdIndex(int id)
103 {
104   if (this->gameBoard[id] == 1) {
105     return 1;
106   }
107   else if (this->gameBoard[id + boardOfset] == 1) {
108     return -1;
109   }
110   return 0;
111 }
112
113 inline void AlphaZero::Game::GameState::IdIndex(int id, int val,
        IDType& b)
114 {
115   int otherId;
116   if (val == 0) {
117     b.set(id, 0);
118     b.set(id + boardOfset, 0);
119     return;
120   }
121   if (val == 1)
122   {
123     otherId = id + boardOfset;
124   }
125   if (val ==-1) {
126     otherId = id;
127     id += boardOfset;
128   }
129   b.set(id, 1);
130   b.set(otherId, 0);
131 }
132
133 inline char AlphaZero::Game::GameState::getPiece(int id)
134 {
135   std::unordered_map<int, char> renderData = { {0,'-'}, {1, 'X'
```

```
    }, {-1, 'O'} };
136   if (std::find(this->allowedActions.begin(), this->
      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_snape_z,
      input_shape_y, input_shape_x });
155   this->toTensor(outTensor);
156   return outTensor;
157 }
158
159 inline void AlphaZero::Game::GameState::toTensor(torch::Tensor&
      tensor, unsigned short idx)
160 {
161   unsigned short pos = 0;
162   unsigned int ofset = (this->player == -1) ? 0 : boardOfset;
163   for (unsigned short z = 0; z < input_snape_z; z++) {
164     for (unsigned short y = 0; y < input_shape_y; y++) {
165       for (unsigned short x = 0; x < input_shape_x; x++) {
166         tensor[idx][z][y][x] = (float)this->gameBoard[(pos +
      ofset) % stateSize];
167         pos++;
168       }
169     }
170   }
171 }
172
173 inline std::pair<int, int> AlphaZero::Game::to2dPos(int pos)
174 {
175   int x = pos % input_shape_x;
```

```
176   int y = (pos - x) / input_shape_x;
177   return { x,y };
178 }
179
180 inline int AlphaZero::Game::from2dPos(int const& x, int const& y
      )
181 {
182   return x + y * input_shape_x;
183 }
```

6.4.2.3 game.cpp

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

```
32        return tmp;
33      }
34      else
35      {
36        return std::vector<int>();
37      }
38      a = { a.first + dx, a.second + dy };
39    }
40    return std::vector<int>();
41  }
42
43  bool AlphaZero::Game::GameState::hasAjacentStones(int const& x,
        int const& y)
44  {
45    int otherX, otherY;
46    for (auto const& direction : ajacentDirections)
47    {
48      otherX = x + direction.first;
49      otherY = y + direction.second;
50      if (isValidPostition(otherX, otherY))
51      {
52        if (this->IdIndex(from2dPos(otherX, otherY)) != 0)
53        {
54          return true;
55        }
56      }
57    }
58    return false;
59  }
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
       _player)
77 {
78   this->gameBoard = board;
79   this->player = _player;
80   this->getAllowedActions();
81   this->gameIsDone();
82 }
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,
       direction.first, direction.second);
92     for (auto const& pos : toFlip)
93     {
94       GameState::IdIndex(pos, this->player, newBoard);
95     }
96   }
97
98   std::shared_ptr<GameState> newState = std::make_shared<
       GameState>(newBoard, -this->player);
99
100  if (newState->allowedActions.size())
101    return newState;
102
103  std::shared_ptr<GameState> newerState = std::make_shared<
       GameState>(newBoard, this->player);
104  return newerState;
105 }
106
107 void AlphaZero::Game::GameState::gameIsDone()
108 {
109   int thisPlayerPoints = 0, otherPlayerPoints = 0;
110   this->done = true;
111   for (int x = 0; x < input_shape_x; x++)
112   {
113     for (int y = 0; y < input_shape_y; y++)
114     {
115       auto val = this->IdIndex(x + y * columOfset);
```

```
116         if (val == 0 && this->allowedActions.size())
117         {
118           this->done = false;
119         }
120         else if (val == this->player)
121         {
122           thisPlayerPoints++;
123         }
124         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 &&
      hasAjacentStones(x, y))
157       {
158         for (auto const& direction : flipDirections)
159         {
```

```
160             auto toFlip = getFlipActions(x, y, direction.first,
    direction.second);
161             if (toFlip.size())
162             {
163               this->allowedActions.push_back(from2dPos(x, y));
164               break;
165             }
166           }
167         }
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 + columOfset * idx), ' ',
184       this->getPiece(1 + columOfset * idx), ' ',
185       this->getPiece(2 + columOfset * idx), ' ',
186       this->getPiece(3 + columOfset * idx), ' ',
187       this->getPiece(4 + columOfset * idx), ' ',
188       this->getPiece(5 + columOfset * idx), ' ',
189       this->getPiece(6 + columOfset * 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],
      actionProbs[4],  actionProbs[3],  actionProbs[2],
      actionProbs[1],  actionProbs[0],
226     actionProbs[15], actionProbs[14], actionProbs[13],
      actionProbs[12], actionProbs[11], actionProbs[10],
      actionProbs[9],  actionProbs[8],
227     actionProbs[23], actionProbs[22], actionProbs[21],
      actionProbs[20], actionProbs[19], actionProbs[18],
      actionProbs[17], actionProbs[16],
228     actionProbs[31], actionProbs[30], actionProbs[29],
      actionProbs[28], actionProbs[27], actionProbs[26],
      actionProbs[25], actionProbs[24],
229     actionProbs[39], actionProbs[38], actionProbs[37],
      actionProbs[36], actionProbs[35], actionProbs[34],
      actionProbs[33], actionProbs[32],
230     actionProbs[47], actionProbs[46], actionProbs[45],
      actionProbs[44], actionProbs[43], actionProbs[42],
```

```
    actionProbs[41], actionProbs[40],
231     actionProbs[55], actionProbs[54], actionProbs[53],
    actionProbs[52], actionProbs[51], actionProbs[50],
    actionProbs[49], actionProbs[48],
232     actionProbs[63], actionProbs[62], actionProbs[61],
    actionProbs[60], actionProbs[59], actionProbs[58],
    actionProbs[57], actionProbs[56]
233   };
234
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],
    actionProbs[37], actionProbs[29], actionProbs[21],
    actionProbs[13], actionProbs[5],
259  actionProbs[62], actionProbs[54], actionProbs[46],
    actionProbs[38], actionProbs[30], actionProbs[22],
    actionProbs[14], actionProbs[6],
260  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)
{
  console_mutex.lock();
  for (int y = 0; y < input_shape_y; y++)
  {
    for (auto const& state : states)
    {
      for (int x = 0; x < input_shape_x; x++)
      {
        std::cout << state->getPiece(from2dPos(x, y)) << " ";
      }
      std::cout << "\t\t";
    }
    std::cout << std::endl;
  }
  std::cout << std::endl;
  console_mutex.unlock();
}
```

6.5 CMakeLists.txt

```
# CMakeList.txt : Top-level CMake project file, do global
    configuration
# and include sub-projects here.
#
cmake_minimum_required (VERSION 3.8)

project ("AlphaZeroPytorch")

# Include sub-projects.
add_subdirectory ("AlphaZeroPytorch")
```

6.6 README.md

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

7 Demos

7.1 Matura-AlphaZero-demos

7.1.1 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.0)
project(AlphaZeroDemos VERSION 0.1.0)
set(CXX_STANDARD 20)
set(CMAKE_BUILD_TYPE Debug)
#set(CMAKE_CXX_COMPILER "/usr/bin/gcc")

include(CTest)
enable_testing()
message(STATUS ${CMAKE_MODULE_PATH})

file(GLOB SharedFiles
  "include/*"
  "scr/*"
)
file(GLOB UnsupervisedFiles
  "unsupervised/src/*"
  "unsupervised/include/*"
)
file(GLOB SupervisedFiles
  "supervised/src/*"
  "supervised/include/*"
)
add_executable(VectorQuantization main.cpp ${SharedFiles} ${
    UnsupervisedFiles})
add_executable(Supervised supervised.cpp ${SharedFiles} ${
    SupervisedFiles})

# Settting model device macro
find_package(CUDA 11.3)
message(STATUS "Using cuda: " ${CUDA_VERSION_STRING})


if (CUDA_VERSION_STRING EQUAL "")
  target_compile_definitions(Supervised PUBLIC MODEL_DEVICE="cpu
    ")
else()
  target_compile_definitions(Supervised PRIVATE MODEL_DEVICE="
    cuda:0")
endif()

# Download libtorch from offical website
if (EXISTS "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip" OR
    EXISTS "${PROJECT_SOURCE_DIR}/dependencies/libtorch")
  message(STATUS "libtorch already downloaded")
else ()
```

```
41   if (CUDA_VERSION_STRING EQUAL "")
42     message(STATUS "did not find cuda11.3 +")
43     message(STATUS "downloading libtorch cpu")
44     file(
45       DOWNLOAD
46       "https://download.pytorch.org/libtorch/cpu/libtorch-cxx11-
     abi-shared-with-deps-1.11.0%2Bcpu.zip"
47       "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip"
48     )
49     set(libtorch_hash "5
     b5ea3067f878dea051f6fcd8fb00338f55517cb6baecdc810983a814e030845
     ")
50   else ()
51     message(STATUS "found cuda version " ${CUDA_VERSION_STRING})
52     message(STATUS "downloading libtorch cuda " ${
     CUDA_VERSION_STRING})
53     file(
54       DOWNLOAD
55       "https://download.pytorch.org/libtorch/cu113/libtorch-
     cxx11-abi-shared-with-deps-1.11.0%2Bcu113.zip"
56       "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip"
57     )
58     set(libtorch_hash "8
     d9e829ce9478db4f35bdb7943308cf02e8a2f58cf9bb10f742462c1d57bf287
     ")
59   endif()
60   file(SHA256 "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip"
     libtorch_checksum)
61   message(STATUS "check sum" ${libtorch_checksum})
62   if (libtorch_checksum MATCHES "${libtorch_hash}")
63     message(STATUS "libtorch checksum is valid")
64   else()
65     message(FATAL_ERROR "libtorch checksum is not valid")
66   endif()
67 endif()
68
69  #unzip libtorch
70 if (EXISTS "${PROJECT_SOURCE_DIR}/dependencies/libtorch")
71   message(STATUS "libtorch already installed")
72 else()
73   message(STATUS "installing libtorch")
74   file(ARCHIVE_EXTRACT INPUT "${PROJECT_SOURCE_DIR}/dependencies
     /libtorch.zip" DESTINATION "${PROJECT_SOURCE_DIR}/
     dependencies")
75   file(REMOVE "${PROJECT_SOURCE_DIR}/dependencies/libtorch.zip")
```

```
  message(STATUS "libtorch installed")
endif()
message(STATUS "libtorch Path:\t" "${PROJECT_SOURCE_DIR}/dependencies/libtorch")
set (CMAKE_PREFIX_PATH "${PROJECT_SOURCE_DIR}/dependencies/libtorch/")


find_package(Torch REQUIRED Torch_DIR)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${TORCH_CXX_FLAGS}")

find_package(colormap REQUIRED INTERFACE)

find_package(wxWidgets REQUIRED gl core base OPTIONAL_COMPONENTS net)
include(${wxWidgets_USE_FILE})

target_include_directories(VectorQuantization PUBLIC "include/" "unsupervised/include")
target_link_libraries(VectorQuantization PRIVATE ${wxWidgets_LIBRARIES})

target_include_directories(Supervised PUBLIC "include/" "supervised/include")
target_link_libraries(Supervised PRIVATE ${wxWidgets_LIBRARIES} ${TORCH_LIBRARIES})
target_link_libraries(Supervised INTERFACE ${colormap_DIR})
message(STATUS "colormap at: " ${colormap_DIR})
set_property(TARGET Supervised PROPERTY CXX_STANDARD 23)

set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```

7.1.2 main.cpp

```
#include <wx/sizer.h>
#include <wx/timer.h>
#include <config.hpp>
#include <cluster.hpp>

class BasicDrawPane;

class RenderTimer : public wxTimer
{
    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"),
        wxPoint(50,50), wxSize(WIDTH,HEIGHT))
68      {
69          wxBoxSizer* sizer = new wxBoxSizer(wxHORIZONTAL);
70          drawPane = new BasicDrawPane( this );
71          sizer->Add(drawPane, 1, wxEXPAND);
72          SetSizer(sizer);
73
74          timer = new RenderTimer(drawPane);
75          Show();
76          timer->start();
77      }
78      ~MyFrame()
79      {
80          delete timer;
81      }
82      void onClose(wxCloseEvent& evt)
83      {
84          timer->Stop();
85          evt.Skip();
86      }
87      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   brushes.push_back(wxBLACK_BRUSH);
119   brushes.push_back(wxYELLOW_BRUSH);
120   brushes.push_back(wxWHITE_BRUSH);
121   this->cluster = new VQ::Cluster(POINT_COUNT, (double)WIDTH, (
      double)HEIGHT, brushes);
122 }
123 BasicDrawPane::~BasicDrawPane(){
124   delete this->cluster;
125 }
126 
127 
128 void BasicDrawPane::paintEvent(wxPaintEvent& evt)
129 {
130     wxPaintDC dc(this);
131     render(dc);
132 }
133 
134 void BasicDrawPane::paintNow()
135 {
136     wxClientDC dc(this);
137     render(dc);
138 }
139 
140 void BasicDrawPane::render( wxDC& dc )
141 {
142     for (size_t idx=0; idx<SIMITERS; idx++)
143         this->cluster->update();
```

```
144
145     dc.SetBackground( *wxWHITE_BRUSH );
146     dc.SetBrush( *wxBLUE_BRUSH);
147     dc.Clear();
148     this->cluster->render(dc);
149 }
```

7.1.3 README.md

```
1  Installation
2  ------------
3
4  prerequisits
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 ```
```

7.1.4 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"),
       wxPoint(50,50), wxSize(WIDTH,HEIGHT))
69     {
70         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()
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
      ::endl;
136   this->model->train();
137   auto size = this->parent->sizer->GetSize();
138   dc.DrawBitmap(this->model->getMap().Rescale(size.GetX(), size.
      GetY()), wxPoint(0,0), false);
139   dc.SetBrush(*wxTRANSPARENT_BRUSH);
140   dc.DrawEllipse(
141     size.GetX()/2 - size.GetX() * (double)CIRC_RADIUS/((double)
      WIDTH),
142     size.GetY()/2 - size.GetY() * (double)CIRC_RADIUS/((double)
      HEIGHT),
143     2 * size.GetX() * (double)CIRC_RADIUS/((double)WIDTH),
144     2 * size.GetY() * (double)CIRC_RADIUS/((double)HEIGHT)
145   );
146 }
```

7.1.5 include

7.1.5.1 point.hpp

```
#pragma once
#include <iostream>
#include <wx/wx.h>

namespace VQ {
  class Point {
    public: using T =              double;

    public: Point();
    public: Point(const Point& p);
    public: Point(const T& x, const T& y);
    public: Point(const std::pair<T, T>& pos);
    public: std::pair<T, T> getPos();
    public: const std::pair<T, T> getPos() const;

    public: void moveTo(const T& x, const T& y);
    public: void moveTo(const std::pair<T, T>& pos);
    public: void moveTo(const Point& point);
    public: void moveBy(const T& dx, const T& dy);
    public: void moveBy(const std::pair<T, T>& dpos);
    public: void moveBy(const Point& dpoint);

    public: Point operator*(const float scalar) const;
    public: Point operator*(const double scalar) const;
    public: Point operator-(const Point& other) const;
    public: Point operator+(const Point& other) const;
    public: Point& operator=(const Point&& other);
    public: Point& operator=(const std::pair<T, T>&& other);
    public: friend std::ostream& operator<<(std::ostream& os,
   const Point& dt);
    public: double abs() const;
    protected: std::pair<T, T> pos;

    public: void render(wxDC& dc, const wxBrush* brush) const;
  };

  Point randomPointInRange(const double& dx, const double& dy);
  std::ostream& operator<<(std::ostream& os, const Point& dt);
}
```

7.1.5.2 utils.hpp

```cpp
#pragma once

#include <cstdlib>
#include <vector>

namespace std {
  double rand(const double& max);
  size_t randMod(const size_t& mod);
}

namespace jce {
  template <typename T>
  T& randElement(std::vector<T>& vec);
  template <typename T>
  const T& randElement(const std::vector<T>& vec);
}

inline double std::rand(const double& max) {
  return (double)(std::rand()) / (((double)RAND_MAX/max));
}
inline size_t std::randMod(const size_t& mod) {
  return std::rand() % mod;
}

template<typename T>
T& jce::randElement(std::vector<T>& vec) {
  return vec[std::randMod(vec.size())];
}
template<typename T>
const T& jce::randElement(const std::vector<T>& vec) {
  return vec[std::randMod(vec.size())];
}
```

7.1.6 scr

7.1.6.1 point.cpp

```cpp
#include <point.hpp>
#include <utils.hpp>
#include <config.hpp>

VQ::Point::Point(): pos({0,0}) {}
VQ::Point::Point(const VQ::Point& p): pos(p.getPos()) {}
VQ::Point::Point(const VQ::Point::T& x, const VQ::Point::T& y):
    pos({x, y}) {}
```

```
8  VQ::Point::Point(const std::pair<VQ::Point::T, VQ::Point::T>&
       _pos): pos(_pos) {}
9  std::pair<VQ::Point::T, VQ::Point::T> VQ::Point::getPos() {
       return pos;}
10 const std::pair<VQ::Point::T, VQ::Point::T> VQ::Point::getPos()
       const {return pos;}
11
12 void VQ::Point::moveTo(const VQ::Point::T& x, const VQ::Point::T
       & y) {
13   this->moveTo(std::pair<double, double>(x, y));
14 }
15 void VQ::Point::moveTo(const std::pair<VQ::Point::T, VQ::Point::
       T>& _pos) {this->pos = _pos;}
16 void VQ::Point::moveTo(const Point& point) {
17   this->moveTo(point.getPos());
18 }
19 void VQ::Point::moveBy(const VQ::Point::T& dx, const VQ::Point::
       T& dy) {
20   this->pos = {
21     dx + this->pos.first,
22     dy + this->pos.second
23   };
24 }
25 void VQ::Point::moveBy(const std::pair<VQ::Point::T, VQ::Point::
       T>& _pos){
26   this->moveBy(_pos.first, _pos.second);
27 }
28 void VQ::Point::moveBy(const Point& point) {
29   this->moveBy(point.getPos());
30 }
31
32 VQ::Point VQ::Point::operator*(const float scalar) const {
33   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  }
```

7.1.7 supervised

7.1.7.1 include

7.1.7.1.1 config.hpp

```
#pragma once

#define HEIGHT          500
#define WIDTH           500
#define RES_HEIGHT      500
#define RES_WIDTH       500
#define POINT_RADIUS    10
#define CIRC_RADIUS     200
#define BATCH_SIZE      1024
#define EPOCHS          8
#define LR              0.1
#define Momentum        0.9
#define LOSS            torch::mse_loss
```

7.1.7.1.2 model.hpp

```
#pragma once

#include <torch/torch.h>
#include <wx/bitmap.h>
#include <point.hpp>

namespace SL {
  class Model : public torch::nn::Module {
    private: using Optimizer          = torch::optim::SGD;
    private: using OptimizerOptions   = torch::optim::SGDOptions
    ;

    private: torch::nn::Linear lin1, lin2, lin3;
    private: torch::nn::LeakyReLU relu;
    private: torch::nn::Sigmoid sigm;
    private: Optimizer optim;

    public: Model();
    public: torch::Tensor forward(torch::Tensor x);
    public: void fit(torch::Tensor x, torch::Tensor y);
    public: void train();

    public: wxImage getMap();
    private: torch::Tensor getMapTensor();
```

```
24
25      private: static std::pair<torch::Tensor, torch::Tensor>
        getTrainingData();
26      private: static double classify(const VQ::Point& point);
27    };
28  }
```

7.1.7.2 src

7.1.7.2.1 model.cpp

```
1  #include <model.hpp>
2  #include <config.hpp>
3  #include <colormap/palettes.hpp>
4
5  #define COLORMAP "jet"
6  c10::Device device(MODEL_DEVICE);
7  c10::Device cpu_device("cpu");
8
9  const static VQ::Point center(HEIGHT/2, HEIGHT/2);
10
11 SL::Model::Model() :
12   lin1(register_module("lin1", torch::nn::Linear(2,256))),
13   lin2(register_module("lin2", torch::nn::Linear(256, 16))),
14   lin3(register_module("lin3", torch::nn::Linear(16, 1))),
15   relu(),
16   sigm(),
17   optim(this->parameters(), OptimizerOptions(LR).momentum(
       Momentum))
18 {
19   std::cout <<device << std::endl;
20   lin1->to(device, true);
21   lin2->to(device, true);
22   lin3->to(device, true);
23   relu->to(device, true);
24   sigm->to(device, true);
25 }
26
27 torch::Tensor SL::Model::forward(torch::Tensor x) {
28   x = x.to(device, true);
29   x = relu(lin1(x));
30   x = relu(lin2(x));
31   x = sigm(lin3(x)).to(cpu_device, true);
32   return x;
33 }
34
```

```
35 void SL::Model::fit(torch::Tensor x, torch::Tensor y_true) {
36   torch::Tensor y_pred = this->forward(x);
37   auto loss = LOSS(y_true, y_pred);
38   loss.backward(loss);
39   optim.step();
40   optim.zero_grad();
41 }
42
43 void SL::Model::train() {
44   auto data = this->getTrainingData();
45   for (size_t idx=0; idx < EPOCHS; idx++)
46     this->fit(data.first, data.second);
47 }
48
49 wxImage SL::Model::getMap() {
50   torch::Tensor y = this->forward(this->getMapTensor());
51   wxImage img(RES_HEIGHT, RES_WIDTH);
52   auto pal = colormap::palettes.at(COLORMAP).rescale(0, 1);
53   for (size_t idx_x=0; idx_x < RES_HEIGHT; idx_x++) {
54     for (size_t idx_y=0; idx_y < RES_WIDTH; idx_y++) {
55       auto pix = pal(y[idx_x*RES_WIDTH + idx_y][0].item<float>()
     );
56       img.SetRGB(idx_x, idx_y,
57         pix.getRed().getValue(),
58         pix.getGreen().getValue(),
59         pix.getBlue().getValue()
60       );
61     }
62   }
63   return img;
64 }
65
66 torch::Tensor SL::Model::getMapTensor() {
67   torch::Tensor out = torch::zeros({RES_HEIGHT * RES_WIDTH, 2});
68   for (size_t idx_x=0; idx_x < RES_HEIGHT; idx_x++) {
69     for (size_t idx_y=0; idx_y < RES_WIDTH; idx_y++) {
70       out[idx_x*RES_WIDTH + idx_y][0] = (double)idx_x/((double)
     RES_HEIGHT);
71       out[idx_x*RES_WIDTH + idx_y][1] = (double)idx_y/((double)
     RES_WIDTH);
72     }
73   }
74   return out;
75 }
76
```

```
77 std::pair<torch::Tensor, torch::Tensor> SL::Model::
       getTrainingData() {
78   torch::Tensor x = torch::ones({BATCH_SIZE, 2});
79   torch::Tensor y = torch::ones({BATCH_SIZE, 1});
80
81   for (size_t idx=0; idx < BATCH_SIZE; idx++) {
82     const VQ::Point p = VQ::randomPointInRange(WIDTH, HEIGHT);
83     y[idx][0] = classify(p);
84     x[idx][0] = p.getPos().first  / (double)HEIGHT;
85     x[idx][1] = p.getPos().second / (double)WIDTH;
86   }
87   return {x, y};
88 }
89 double SL::Model::classify(const VQ::Point& point) {
90   if ((point - center).abs() > CIRC_RADIUS) {
91     return 1.0;
92   }
93   return 0.0;
94 }
```

7.1.8 unsupervised

7.1.8.1 include

7.1.8.1.1 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     private: std::vector<Group> groups;
10    public:  Cluster(const size_t& points, const double& x,
      const double& y, const std::vector<const wxBrush*>& brushes);
11    public:  void update();
12    public:  void render(wxDC& dc);
13    private: Group* getClosestGroup(const Point& point);
14    private: const Group* getClosestGroup(const Point& point)
      const;
15    private: const wxBrush* getPointBrush(const Point& point)
      const;
16    private: void generatePoints(const size_t& count, const
      double& x, const double& y);
```

```
17      private: void generateGroups(const double& x, const double&
        y, const std::vector<const wxBrush*>& brushes);
18    };
19  }
```

7.1.8.1.2 config.hpp

```
1  #pragma once
2
3  #define HEIGHT              500
4  #define WIDTH               500
5  #define POINT_COUNT         200
6  #define POINT_RADIUS        5
7  #define GRPOUT_SIZE         10
8  #define SIMITERS            5
```

7.1.8.1.3 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*
       brush);
11     //public: ~Group();
12     public: void render(wxDC& dc) const;
13     public: const wxBrush* getBrush() const;
14   };
15
16   Group randomGroupInRange(const double& x, const double& y,
       const wxBrush* brush);
17 }
```

7.1.8.2 src

8 Further definitions

8.1 Set Exclusion

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

$$\mathbb{A} - \mathbb{B} = \mathbb{A} \setminus \mathbb{B} = \{x : x \in \mathbb{A} \ \wedge \ x \notin \mathbb{B}\} \tag{67}$$

8.2 Hadamard product

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

$$(A \circ B)_{ij} = A_{ij} \cdot B_{ij} \tag{68}$$

For example consider the following 2×3 matrices:

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

8.3 Inner Product

8.3.1 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 $\langle A, B \rangle_I : \ \mathbb{R}^{m \times n}, \ \mathbb{R}^{m \times n} \to \mathbb{R}$ be defined as:

$$\langle A, B \rangle_I = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} B_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{n} (A \circ B)_{ij} \tag{69}$$

For example consider the following 2×3 matrices:

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

8.3.2 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 $\langle A, B \rangle_I : \mathbb{R}^{m \times \dots}, \mathbb{R}^{m \times \dots} \to \mathbb{R}$ be defined as:

$$\langle A, B \rangle_I = \sum_{i=0}^{m} \langle A_i, B_i \rangle_I \tag{70}$$

8.4 Submatrix

Let $m = m_{ijk}$ be an $m \times n \times o$ dimensional tensor.
Let $\langle \rangle_{S\, x \times y,\, ab}$ be the matrix slicing opperator. $x \times y$ 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: $x \in [0, m[\cap \mathbb{N}$, $y \in [0, n[\cap \mathbb{N}$, $a \in [1, m - x] \cap \mathbb{N}$ and $b \in [1, n - y] \cap \mathbb{N}$.
The submatrix $\langle m \rangle_{S\, x \times y,\, ab}$ is defined as:

$$\langle m \rangle_{S\, x \times y,\, ab} = \begin{bmatrix} [m_{ab1}, \dots, m_{abo}] & \dots & [m_{(a+x)b1}, \dots, m_{(a+x)bo}] \\ \vdots & \ddots & \vdots \\ [m_{a(b+y)1}, \dots, m_{abo}] & \dots & [m_{(a+x)b1}, \dots, m_{(a+x)(b+y)o}] \end{bmatrix} \tag{71}$$

8.5 Vectorization

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

$$\mathrm{vec}(A) = [A_{11} \dots A_{1m}\, A_{21} \dots A2m \dots A_{n1} \dots Anm]^T \tag{72}$$

Taken verbatim from:[8]
For example, the 3×2 matrix $A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$ vectorizes to

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

8.5.1 Tensors

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

$$vec(T) = T_0 \frown T_1 \frown \cdots \frown T_n \tag{73}$$

Where $\frown$ is defined defined in section 8.6 on page 242. For those who are familiar with python's numpy library, *vec* is equivilent to numpy.flatten.

8.6 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.

$$v \frown u = \begin{pmatrix} v_1 \\ \vdots \\ v_{n_v} \\ u_1 \\ \vdots \\ u_{n_u} \end{pmatrix} \tag{74}$$

8.7 Elementwise vector operations

Let $v \in \mathbb{R}^n$ be a n dimensional vector. The Elementwise vector opperator $\langle s, v \rangle_E$ of some function s is defined as:

$$\langle s, v \rangle_E = \begin{pmatrix} s(v_1) \\ s(v_2) \\ \vdots \\ s(v_n) \end{pmatrix} \tag{75}$$

For example let $v = [1, 4, 2]^T$ and $s(x) = \frac{1}{x}$, than $\langle s, v \rangle_E = [1, 0.25, 0.5]^T$.

8.8 Sets

Let

$$\mathbb{R}_{a,b} = \{x \in \mathbb{R} \ : \ a \leq x \leq b\} \tag{76}$$

$$\mathbb{N}_{a,b} = \{x \in \mathbb{N} \ : \ a \leq x \leq b\} \tag{77}$$

9 Further examples

9.1 Mcts

The full set of data as seen by the computer in the mcts example (fig 1 on page 14). The node sets are as follows defined by the amount of edged leaving each node.

$$\mathbb{M} = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\} \tag{78}$$
$$\mathbb{M}_{leaf} = \{n_3, n_5, n_6, n_7, n_8\} \tag{79}$$
$$\mathbb{M}_{expanded} = \{n_1, n_2, n_4\} \tag{80}$$

The edge sets for every node are as follows.

$$\mathbb{E}(n_1) = \{e_1, e_2, e_3\} \tag{81}$$
$$\mathbb{E}(n_2) = \{e_4, e_5\} \tag{82}$$
$$\mathbb{E}(n_3) = \emptyset \tag{83}$$
$$\mathbb{E}(n_4) = \{e_6, e_7, e_8\} \tag{84}$$
$$\mathbb{E}(n_5) = \emptyset \tag{85}$$
$$\mathbb{E}(n_6) = \emptyset \tag{86}$$
$$\mathbb{E}(n_7) = \emptyset \tag{87}$$
$$\mathbb{E}(n_8) = \emptyset \tag{88}$$

The edge by action and node function $\mathcal{E}$ looks af follows:

$$\mathcal{E}(n_1, A_1) = e_1 \tag{89}$$
$$\mathcal{E}(n_1, A_2) = e_2 \tag{90}$$
$$\mathcal{E}(n_1, A_3) = e_3 \tag{91}$$
$$\mathcal{E}(n_2, A_4) = e_4 \tag{92}$$
$$\mathcal{E}(n_2, A_5) = e_5 \tag{93}$$
$$\mathcal{E}(n_4, A_6) = e_6 \tag{94}$$
$$\mathcal{E}(n_4, A_7) = e_7 \tag{95}$$
$$\mathcal{E}(n_4, A_8) = e_8 \tag{96}$$

The node an edge points from function $\mathcal{N}_{from}$ looks af follows:

$$\mathcal{N}_{from}(e_1) = n_1 \tag{97}$$
$$\mathcal{N}_{from}(e_2) = n_1 \tag{98}$$
$$\mathcal{N}_{from}(e_3) = n_1 \tag{99}$$
$$\mathcal{N}_{from}(e_4) = n_2 \tag{100}$$
$$\mathcal{N}_{from}(e_5) = n_2 \tag{101}$$
$$\mathcal{N}_{from}(e_6) = n_4 \tag{102}$$
$$\mathcal{N}_{from}(e_7) = n_4 \tag{103}$$
$$\mathcal{N}_{from}(e_8) = n_4 \tag{104}$$

The node an edge points to function $\mathcal{N}_{to}$ looks af follows:

$$\mathcal{N}_{to}(e_1) = n_3 \tag{105}$$
$$\mathcal{N}_{to}(e_2) = n_2 \tag{106}$$
$$\mathcal{N}_{to}(e_3) = n_4 \tag{107}$$
$$\mathcal{N}_{to}(e_4) = n_5 \tag{108}$$
$$\mathcal{N}_{to}(e_5) = n_6 \tag{109}$$
$$\mathcal{N}_{to}(e_6) = n_6 \tag{110}$$
$$\mathcal{N}_{to}(e_7) = n_7 \tag{111}$$
$$\mathcal{N}_{to}(e_8) = n_8 \tag{112}$$

10 Proofs

10.1 tanh derivative

$tanh : \mathbb{R} \to \mathbb{R}$ is defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{113}$$

$$\begin{aligned}\frac{d}{dx}\left[\frac{e^x - e^{-x}}{e^x + e^{-x}}\right] &= \frac{(e^x + e^{-x})\frac{d}{dx}\left[e^x - e^{-x}\right] - (e^x - e^{-x})\frac{d}{dx}\left[e^x + e^{-x}\right]}{(e^x + e^{-x})^2} \\ &= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\ &= \frac{e^{2x} + 2e^x e^{-x} + e^{-2x} - e^{2x} + 2e^x e^{-x} - e^{-x}}{(e^x + e^{-2x})^2} \\ &= \frac{4}{(e^x + e^{-x})^2}\end{aligned} \tag{114}$$

because

$$sech(x) := \frac{2}{e^x + e^{-x}} \tag{115}$$

using (114) and (115) we can see, that

$$\frac{d}{dx}\left[tanh(x)\right] = sech(x)^2 \tag{116}$$

10.2 Softmax Derivative

The Softmax $s : \mathbb{R}^n \to \mathbb{R}^n$, $n \in \mathbb{N}$, is defined as

$$s(v)_i := \frac{e^{v_i}}{\sum_{j \in v} e^j} \tag{117}$$

for all $i \in [1, n] \cap \mathbb{N}$. Due to the multidimensional natur of this function we are taking the derivative $\frac{d}{dv_j} s(v)_i$, with $i \in [1, n] \cap \mathbb{N}$ and $j \in [1, n] \cap \mathbb{N}$.

case $i \neq j$

$$\begin{aligned}\frac{d}{dv_j}\left[\frac{e^{v_i}}{\sum_{a \in v} e^a}\right] &= -\frac{\frac{d}{dx}\left[\sum_{a \in v} e^a\right] e^{v_i}}{(\sum_{a \in v} e^a)^2} \\ &= -\frac{e^{v_j} e^{v_i}}{(\sum_{a \in v} e^a)^2} \\ &= -s(v)_j s(v)_i\end{aligned} \tag{118}$$

case $i = j$

$$\begin{aligned}
\frac{d}{dv_j}\left[\frac{e^{v_i}}{\sum_{a\in v} e^a}\right] &= \frac{\frac{d}{dx}\left[e^{v_i}\right]\sum_{a\in v} e^a - e^{v_i}\frac{d}{dx}\left[\sum_{a\in v} e^a\right]}{(\sum_{a\in v} e^a)^2} \\
&= \frac{e^{v_i}\sum_{a\in v} e^a - e^{v_i}e^{v_j}}{(\sum_{a\in v} e^a)^2} \\
&= s(v)_i - s(v)_i s(v)_j \\
&= s(v)_i(1 - s(v)_j)
\end{aligned} \tag{119}$$

10.3 Elo Rating

Let r_b, $r_a \in \mathbb{R}$
Let E be the elo win expectation

$$E = \frac{1}{1 + e^{r_b - r_a/400}} \tag{120}$$

Solve to r_a:

$$E + Ee^{r_b - r_a/400} = 1 \tag{121}$$

$$Ee^{r_b - r_a/400} = 1 - E \tag{122}$$

$$e^{r_b - r_a/400} = \frac{1 - E}{E} \tag{123}$$

$$\ln\left(\frac{1 - E}{E}\right) = \frac{r_b - r_a}{400} \tag{124}$$

$$400\ \ln\left(\frac{1 - E}{E}\right) = r_b - r_a \tag{125}$$

$$r_b - 400\ \ln\left(\frac{1 - E}{E}\right) = r_a \tag{126}$$

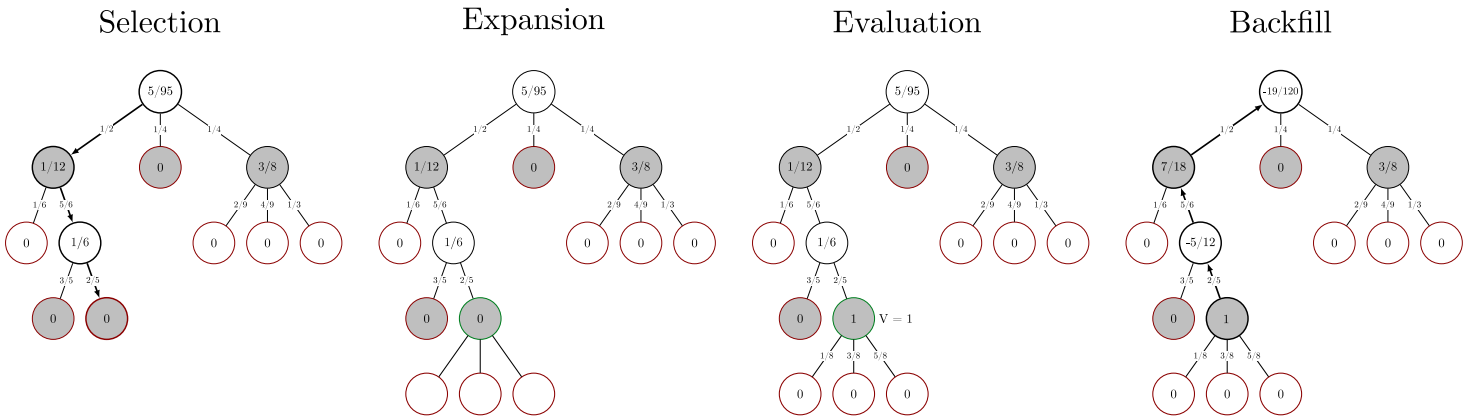


Figure 9: MCTS simulation steps. In this diagram, the numbers in the node represent Q and the number on the arrow is P. The red nodes are leaf nodes and the green one is the leaf node n_L. During the **selection** phase, σ is used to find successive nodes until the node n_L is reached. This is shown 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