Contents

1 Introduction 1

2 Contents 3
   2.1 Game Environments .................................................. 3
   2.2 Evaluators ............................................................... 7
   2.3 Game Play ............................................................... 8
   2.4 Neural Networks ....................................................... 8
   2.5 Players ................................................................. 9
   2.6 Data Processing ....................................................... 10
   2.7 Search Algorithm ..................................................... 13
   2.8 Reinforcement Learning ........................................... 15

Python Module Index 19

Index 21
CHAPTER 1

Introduction

AlphaZero is a replication of Mastering the game of Go without human knowledge and Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.
2.1 Game Environments

```python
class AlphaZero.env.go.GameState(size=19, komi=7.5, enforce_superko=False, history_length=8)
```

State of a game of Go and some basic functions to interact with it

```python
get_group(position)
```
Get the group of connected same-color stones to the given position.

Parameters

- **position** – a tuple of (x, y), x being the column index of the starting position of the search,

- **being the row index of the starting position of the search** (y) –

Returns a set of tuples consist of (x, y)s which are the same-color cluster, which contains the input single position. len(group) is size of the cluster, can be large.

Return type set

```python
get_groups_around(position)
```
returns a list of the unique groups adjacent to position ‘unique’ means that, for example in this position:

```
. . . .
. B W .
. W W .
. . . .
```

only the one white group would be returned on get_groups_around((1,1))

Parameters **position** – a tuple of (x, y)

Returns a list of the unique groups adjacent to position.
Return type list

copy() Gets a copy of this Game state
    Returns a copy of this Game state
    Return type AlphaZero.env.go.GameState

is_suicide(action)
    Parameters action – a tuple of (x, y)
    Returns return true if having current_player play at <action> would be suicide
    Return type bool

is_positional_superko(action)
    Find all actions that the current_player has done in the past, taking into account the fact that history starts
    with BLACK when there are no handicaps or with WHITE when there are. :param action: a tuple of (x, y)
    Returns if the move is positional superko.
    Return type bool

is_legal(action)
    Determines if the given action (x,y) is a legal move :param action: a tuple of (x, y)
    Returns if the move is legal.
    Return type bool

is_eyeish(position, owner)
    Parameters
        • position – a tuple of (x, y)
        • owner – the color
    Returns whether the position is empty and is surrounded by all stones of ‘owner’
    Return type bool

is_eye(position, owner, stack=[])
    returns whether the position is a true eye of ‘owner’ Requires a recursive call; empty spaces diagonal to
    ‘position’ are fine as long as they themselves are eyes

get_legal_moves(include_eyes=True)
    Parameters include_eyes – whether to include eyes in legal moves
    Returns a list of tuples.
    Return type list

get_winner()
    Calculate score of board state and return player ID (1, -1, or 0 for tie) corresponding to winner. Uses ‘Area
    scoring’.
    Returns the color of the winner.
    Return type int

place_handicaps(actions)
    Place handicap stones of black. :param actions: a list of tuples of (x, y)
    Returns None
place_handicap_stone\(\text{\textit{action, color=1}}\)
Place a handicap stone of the specified color.
: param action: a tuple of (x, y) : param color: the color of the move

Returns None

get_current_player()

Returns the color of the player who will make the next move.
Return type int

do_move\(\text{\textit{action, color=None}}\)
Play stone at action=(x,y). If color is not specified, current_player is used. If it is a legal move, current_player switches to the opposite color. If not, an IllegalMove exception is raised.

Parameters
  - \textit{action} – a tuple of (x, y)
  - \textit{color} – the color of the move

Returns if it is the end of game.
Return type bool

transform\(\text{\textit{transform_id}}\)

Transform the current board and the history boards according to D(4). Caution: self.history (action history) is not modified, thus this function should ONLY be used for state evaluation.

Parameters \textit{transform_id} – integer in range [0, 7]
Returns None

exception AlphaZero.env.go.IllegalMove

class AlphaZero.env.mnk.GameState\(\text{\textit{history_length=8}}\)
Game state of mnk Game.

copy()

Gets a copy of this Game state

Returns a copy of this Game state
Return type AlphaZero.env.mnk.GameState

is_legal\(\text{\textit{action}}\)
Determines if the given action (x,y) is a legal move.
: param action: a tuple of (x, y)

Returns if the move is legal.
Return type bool

get_legal_moves()

Returns a list of legal moves.
Return type list

get_winner()

Returns: The winner, None if the game is not ended yet

do_move\(\text{\textit{action, color=None}}\)
Play stone at action=(x,y). If color is not specified, current_player is used. If it is a legal move, current_player switches to the opposite color. If not, an IllegalMove exception is raised.
Parameters

• action – a tuple of (x, y)
• color – the color of the move

Returns if it is the end of game.

Return type bool

transform (transform_id)

Transform the current board and the history boards according to D(4). Caution: self.history (action history) is not modified, thus this function should ONLY be used for state evaluation.

Parameters transform_id – integer in range [0, 7]

Returns None

exception AlphaZero.env.mnk.IllegalMove

class AlphaZero.env.reversi.GameState (size=8, history_length=8)

Game state of Reversi Game.

copy ()

Gets a copy of this Game state

Returns a copy of this Game state

Return type AlphaZero.env.reversi.GameState

is_legal (action)

Determines if the given action (x,y) is a legal move

:param action: a tuple of (x, y)

Returns if the move is legal.

Return type bool

get_legal_moves ()

This function is infrequently used, therefore not optimized. Checks all non-pass moves

Returns a list of legal moves

Return type list

get_winner ()

Counts the stones on the board, assumes the game is ended

Returns The winner, None if the game is not ended yet

Return type int

do_move (action, color=None)

Play stone at action=(x,y). If color is not specified, current_player is used If it is a legal move, current_player switches to the opposite color If not, an IllegalMove exception is raised

Parameters

• action – a tuple of (x, y)
• color – the color of the move

Returns if it is the end of game.

Return type bool
transform\((\text{transform\_id})\)

Transform the current board and the history boards according to $D(4)$. Caution: self.history (action history) is not modified, thus this function should ONLY be used for state evaluation.

Parameters \text{transform\_id} – integer in range $[0, 7]$

Returns None

exception AlphaZero.env.reversi.IllegalMove

2.2 Evaluators

class AlphaZero.evaluator.nn_eval_parallel.NNEvaluator\((\text{cluster, game\_config, ext\_config})\)

Provide neural network evaluation services for model evaluator and data generator. Instances should be created by the main evaluator/generator thread. Context manager (with statement) is preferred because of the automatic start and termination of the listening thread.

Example

with NNEvaluator(...) as eval: pass

Parameters

• \text{cluster} – Tensorflow cluster spec

• \text{game\_config} – A dictionary of game environment configuration

• \text{ext\_config} – A dictionary of system configuration

\text{eval}(\text{state})

This function is called by mcts threads.

Parameters \text{state} – GameState

Returns (policy, value) pair

Return type Tuple

\text{sl\_listen}()

The listener for saving and loading the network parameters. This is run in new thread instead of process.

\text{load}(\text{filename})

Send the load request.

Parameters \text{filename} – the filename of the checkpoint

\text{save}(\text{filename})

Send the save request.

Parameters \text{filename} – the filename of the checkpoint

\text{listen}()

The listener for collecting the computation requests and performing neural network evaluation.
2.3 Game Play

```python
class AlphaZero.game.gameplay.Game(nn_eval_1, nn_eval_2, game_config, ext_config)
A single game of two players.
```

**Parameters**

- **nn_eval_1** – NNEvaluator instance. This class doesn’t create evaluator.
- **nn_eval_2** – NNEvaluator instance.

**start()**

Make the instance callable. Start playing.

**Returns** Game winner. Definition is in go.py.

**get_history()**

Convert the format of game history for training.

**Returns** game states, probability maps and game results

**Return type** tuple of numpy arrays

2.4 Neural Networks

```python
class AlphaZero.network.main.Network(game_config, num_gpu=1, train_config="/home/docs/checkouts/readthedocs.org/user_builds/alphazero/checkouts/latest/AlphaZero/network/..../config/reinforce.yaml", load_pretrained=False, data_format='NHWC', cluster=<MagicMock name='mock.ClusterSpec()' id='140228231301888'>, job='main')
```

This module defines the network structure and its operations.

**Parameters**

- **game_config** – the rules and size of the game
- **train_config** – defines the size of the network and configurations in model training.
- **num_gpu** – the number of GPUs used for computation.
- **load_pretrained** – whether to load the pre-trained model
- **data_format** – input format, either “NCHW” or “NHWC”. “NCHW” achieves higher performance on GPU, but it’s not compatible with CPU.
- **job (cluster,)** – for distributed training.

**update(data)**

Update the model parameters.

**Parameters** **data** – tuple (state, action, result, ). *state* is a numpy array of shape [None, filters, board_height, board_width]. *action* is a numpy array of shape [None, flat_move_output]. *result* is a numpy array of shape [None].

**Returns** Average loss of the minibatch.

**response(data)**

Predict the action and result given current state.

**Parameters** **data** – (state, ). *state* is a numpy array of shape [None, filters, board_height, board_width].
Returns A tuple \((R_p, R_v)\). \(R_p\) is the probability distribution of action, a numpy array of shape \([None, 362]\). \(R_v\) is the expected value of current state, a numpy array of shape \([None]\).

evaluate(data)
Calculate loss and result based on supervised data.

Parameters data – tuple \((state, action, result, )\). \(state\) is a numpy array of shape \([None, filters, board_height, board_width]\). \(action\) is a numpy array of shape \([None, flat_move_output]\). \(result\) is a numpy array of shape \([None]\).

Returns A tuple \((loss, acc, mse)\). \(loss\) is the average loss of the minibatch. \(acc\) is the position prediction accuracy. \(mse\) is the mean squared error of game outcome.

global_step()
Get global step.

save(filename)
Save the model.

Parameters filename – prefix to the saved file. The final name is filename + global_step

load(filename)
Load the model.

Parameters filename – the name of saved file.

class AlphaZero.network.model.Model(game_config, train_config, data_format='NHWC')
Neural network for AlphaGoZero. As described in “Mastering the game of Go without human knowledge”.

Parameters

• game_config – the rules and size of the game
• train_config – defines the size of the network and configurations in model training.
• data_format – input format, either “NCHW” or “NHWC”.

2.5 Players

class AlphaZero.player.cmd_player.Player
Represents a player controlled by a human in the command line playing interface.

think(state)
Asks the user for input and returns if it’s legal.

Parameters state – the current game state.

Returns a tuple of the input move and None.

Return type tuple

ack(move)
Does nothing.

Parameters move – the move played.

Returns None

class AlphaZero.player.mcts_player.Player(eval_fun, game_config, ext_config)
Represents a player playing according to Monto Carlo Tree Search.

think(state, dirichlet=False)
Generate a move according to a game state.
Parameters

- **state** – a game state
- **dirichlet** – whether to apply dirichlet noise to the result prob distribution

Returns
The generated move and probabilities of moves

Return type
tuple

\[ \text{ack}(\text{move}) \]\n
Update the MCT.

Parameters **move** – A new move

**class** AlphaZero.player.nn_player.Player\( (\text{nn_eval}, \text{game_config}) \)

Represents a player playing according to an evaluation function.

\[ \text{think}(\text{state}) \]\n
Chooses the move with the highest probability by evaluating the current state with the evaluation function.

:param state: the current game state.

Returns
a tuple of the calculated move and None.

Return type
tuple

\[ \text{ack}(\text{move}) \]\n
Does nothing.

Parameters **move** – the current move.

Returns
None

### 2.6 Data Processing

**exception** AlphaZero.processing.go.game_converter.SizeMismatchError

**exception** AlphaZero.processing.go.game_converter.NoResultError

**exception** AlphaZero.processing.go.game_converter.SearchProbsMismatchError

**class** AlphaZero.processing.go.game_converter.GameConverter\( (\text{features}) \)

Convert SGF files to network input feature files.

\[ \text{convert_game}(\text{file_name}, \text{bd_size}) \]\n
Read the given SGF file into an iterable of (input,output) pairs for neural network training

Each input is a GameState converted into one-hot neural net features Each output is an action as an \((x,y)\) pair (passes are skipped)

If this game’s size does not match bd_size, a SizeMismatchError is raised

Parameters

- **file_name** – file name
- **bd_size** – board size

Returns
neural network input, move and result

Return type
tuple
\texttt{sgfs\_to\_hdf5} \((\text{sgf\_files}, \text{hdf5\_file}, \text{bd\_size}=19, \text{ignore\_errors}=True, \text{verbose}=False)\)

Convert all files in the iterable \texttt{sgf\_files} into an hdf5 group to be stored in \texttt{hdf5\_file}.

The resulting file has the following properties:

- \texttt{states} : dataset with shape \((n\_data, n\_features, \text{board width}, \text{board height})\)
- \texttt{actions} : dataset with shape \((n\_data, 2)\) (actions are stored as \(x,y\) tuples of where the move was played)
- \texttt{results} : dataset with shape \((n\_data, 1)\), +1 if current player wins, -1 otherwise
- \texttt{file\_offsets} : group mapping from filenames to tuples of \((\text{index}, \text{length})\)

For example, to find what positions in the dataset come from ‘test.sgf’:

\begin{verbatim}
index, length = file_offsets['test.sgf']
test_states = states[index:index+length]
test_actions = actions[index:index+length]
\end{verbatim}

\textbf{Parameters}

- \texttt{sgf\_files} – an iterable of relative or absolute paths to SGF files
- \texttt{hdf5\_file} – the name of the HDF5 where features will be saved
- \texttt{bd\_size} – side length of board of games that are loaded
- \texttt{ignore\_errors} – if True, issues a Warning when there is an unknown exception rather than halting. Note that \texttt{sgf\_ParseException} and \texttt{go\_IllegalMove} exceptions are always skipped.
- \texttt{verbose} – display setting

\textbf{Returns} None

\texttt{selfplay\_to\_hdf5} \((\text{sgf\_pkl\_files}, \text{hdf5\_file}, \text{bd\_size}=19, \text{ignore\_errors}=True, \text{verbose}=False)\)

Convert all files in the iterable \texttt{sgf\_files} into an hdf5 group to be stored in \texttt{hdf5\_file}.

The resulting file has the following properties:

- \texttt{states} : dataset with shape \((n\_data, n\_features, \text{board width}, \text{board height})\)
- \texttt{actions} : dataset with shape \((n\_data, 2)\) (actions are stored as \(x,y\) tuples of where the move was played)
- \texttt{results} : dataset with shape \((n\_data, 1)\), +1 if current player wins, -1 otherwise
- \texttt{file\_offsets} : group mapping from filenames to tuples of \((\text{index}, \text{length})\)

For example, to find what positions in the dataset come from ‘test.sgf’:

\begin{verbatim}
index, length = file_offsets['test.sgf']
test_states = states[index:index+length]
test_actions = actions[index:index+length]
\end{verbatim}

\textbf{Parameters}

- \texttt{sgf\_pkl\_files} – an iterable of relative or absolute paths to SGF and PKL files
- \texttt{hdf5\_file} – the name of the HDF5 where features will be saved

\section*{2.6. Data Processing}
• **bd_size** – side length of board of games that are loaded
• **ignore_errors** – if True, issues a Warning when there is an unknown
• **rather than halting. Note that sgf.ParseException and (exception)** –
• **exceptions are always skipped**(go.IllegalMove)** –
• **verbose** – display setting

Returns None

AlphaZero.processing.go.game_converter.run_game_converter(cmd_line_args=None)
Run conversions.

Parameters cmd_line_args – command-line args may be passed in as a list

Returns None

class AlphaZero.processing.state_converter.StateTensorConverter(config, feature_list=None)

a class to convert from AlphaGo GameState objects to tensors of one-hot features for NN inputs

get_board_history(state)
A feature encoding WHITE and BLACK on separate planes of recent history_length states

Parameters state – the game state

Returns numpy.ndarray

state_to_tensor(state)
Convert a GameState to a Theano-compatible tensor

Parameters state – the game state

Returns numpy.ndarray

class AlphaZero.processing.state_converter.TensorActionConverter(config)
a class to convert output tensors from NN to action tuples

tensor_to_action(tensor)

Parameters tensor – a 1D prob tensor with length flat_move_output

Returns a list of (action, prob)

Return type list

class AlphaZero.processing.state_converter.ReverseTransformer(config)

lr_reflection(action_prob)
Flips the coordinate of action probability vector like np.fliplr
Modification is made in place. Note that PASS_MOVE should be placed at the end of this vector. Condition check is disabled for efficiency.

Parameters action_prob – action probabilities

Returns None

reverse_nprot90(action_prob, transform_id)
Reverse the coordinate transform of np.rot90 performed in go.Gamestate.transform

Rotate the coordinates by Pi/4 * id clockwise

Parameters
• action_prob – action probability vector
• **transform_id** – argument passed to `np.rot90`

  **Returns** None

  **reverse_transform**(*action_prob, transform_id*)

  Reverse the coordinates for `go.GameState.transform` The function make modifications in place

  **Parameters**

  • **action_prob** – list of (action, prob)

  • **transform_id** – number used to perform the transform, range: [0, 7]

  **Returns** None

### 2.7 Search Algorithm

**class** AlphaZero.search.mcts.MCTreeNode(*parent, prior_prob*)

Tree Node in MCTS.

  **expand**(*policy, value*)

  Expand a leaf node according to the network evaluation. NO visit count is updated in this function, make sure it’s updated externally.

  **Parameters**

  • **policy** – a list of (action, prob) tuples returned by the network

  • **value** – the value of this node returned by the network

  **Returns** None

  **select**()

  Select the best child of this node.

    **Returns** A tuple of (action, next_node) with highest Q(s,a)+U(s,a)

    **Return type** tuple

  **update**(v)

  Update the three values

    **Parameters** v – value

    **Returns** None

  **get_selection_value**()

  Implements PUCT Algorithm’s formula for current node.

    **Returns** None

  **get_mean_action_value**()

  Calculates Q(s,a)

    **Returns** mean action value

    **Return type** real

  **visit**()

  Increment the visit count.

    **Returns** None
is_leaf()
Checks if it is a leaf node (i.e. no nodes below this have been expanded).

Returns if the current node is leaf.
Return type bool

is_root()
Checks if it is a root node.

Returns if the current node is root.
Return type bool

class AlphaZero.search.mcts.MCTSearch (evaluator, game_config, max_playout=1600)
Create a Monto Carlo search tree.

calc_move (state, dirichlet=False, prop_exp=True)
Calculates the best move

Parameters
• state – current state
• dirichlet – enable Dirichlet noise described in “Self-play” section
• prop_exp – select the final decision proportional to its exponential visit

Returns the calculated result (x, y)
Return type tuple

calc_move_with_probs (state, dirichlet=False)
Calculates the best move, and return the search probabilities. This function should only be used for self-play.

Parameters
• state – current state
• dirichlet – enable Dirichlet noise described in “Self-play” section

Returns the result (x, y) and a list of (action, probs)
Return type tuple

update_with_move (last_move)
Step forward in the tree, keeping everything we already know about the subtree, assuming that calc_move() has been called already. Siblings of the new root will be garbage-collected. :returns: None

AlphaZero.search.mcts.randint (low, high=None, size=None, dtype='l')
Return random integers from low (inclusive) to high (exclusive).

Return random integers from the “discrete uniform” distribution of the specified dtype in the “half-open” interval [low, high). If high is None (the default), then results are from [0, low).

Parameters
• low (int) – Lowest (signed) integer to be drawn from the distribution (unless high=None, in which case this parameter is one above the highest such integer).
• high (int, optional) – If provided, one above the largest (signed) integer to be drawn from the distribution (see above for behavior if high=None).
• **size** *(int or tuple of ints, optional)* – Output shape. If the given shape is, e.g., *(m, n, k)*, then *m* * n * *k* samples are drawn. Default is None, in which case a single value is returned.

• **dtype** *(dtype, optional)* – Desired dtype of the result. All dtypes are determined by their name, i.e., ‘int64’, ‘int’, etc, so byteorder is not available and a specific precision may have different C types depending on the platform. The default value is ‘np.int’.

New in version 1.11.0.

**Returns out** – *size*-shaped array of random integers from the appropriate distribution, or a single such random int if *size* not provided.

**Return type** int or ndarray of ints

See also:

**random.random_integers()** similar to **randint**, only for the closed interval *[low, high]*, and 1 is the lowest value if *high* is omitted. In particular, this other one is the one to use to generate uniformly distributed discrete non-integers.

**Examples**

```python
>>> np.random.randint(2, size=10)
array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0])

>>> np.random.randint(1, size=10)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Generate a 2 x 4 array of ints between 0 and 4, inclusive:

```python
>>> np.random.randint(5, size=(2, 4))
array([[4, 0, 2, 1],
       [3, 2, 2, 0]])
```

### 2.8 Reinforcement Learning

**class** AlphaZero.train.parallel.evaluator.Evaluator**(nn_eval_chal, nn_eval_best, r_conn, s_conn, game_config, ext_config)**

This class compares the performance of the up-to-date model and the best model so far by holding games between these two models.

**Parameters**

- **nn_eval_chal** – NNEvaluator instance storing the up-to-date model
- **nn_eval_best** – NNEvaluator instance storing the best model so far
- **r_conn** – Pipe to receive the message from optimizer
- **s_conn** – Pipe to send the model updating message to the self play module
- **game_config** – A dictionary of game environment configuration
- **ext_config** – A dictionary of system configuration

**eval_wrapper**(color_of_new)

Wrapper for a single game.
Parameters **color_of_new** – The color of the new model (challenger)

**run**()
The main evaluation process. It will launch games asynchronously and examine the winning rate.

**class** AlphaZero.train.parallel.selfplay.**Selfplay**(nn_eval, r_conn, data_queue, game_config, ext_config)
This class generates training data from self-play games.
Run only this file to start a remote self-play session.

**Example**

$ python -m AlphaZero.train.parallel.selfplay <master addr>

**Parameters**

- **nn_eval** – NNEvaluator instance storing the best model so far
- **r_conn** – Pipe to receive the model updating message
- **data_queue** – Queue to put the data
- **game_config** – A dictionary of game environment configuration
- **ext_config** – A dictionary of system configuration

**selfplay_wrapper()**
Wrapper for a single self-play game.

**run()**
The main data generation process. It will keep launching self-play games.

**model_update_handler()**
The handler for model updating. It will try to load new network parameters. If it is the master session, it will also notify the remote sessions to update.

**rcv_remote_data_handler()**
The handler for receiving data from remote sessions. Only the master session uses this handler.

**remote_update_handler()**
The handler for receiving the update notification from the master session. Only the remote sessions use this handler.

**class** AlphaZero.train.parallel.datapool.**DataPool**(ext_config)
This class stores the training data and handles data sending and receiving.

**Parameters** **ext_config** – A dictionary of system configuration

**serve()**
The listening process. It will first load the saved data and then run a loop to handle data getting and putting requests.

**merge_data**(data)
Put the new data into the array. Since the array is pre-allocated, this function will overwrite the old data with the new ones and record the ending index.

**Parameters** **data** – New data from self-play games

**put**(data)
Send the putting request. This function will be called by self-play games.

**Parameters** **data** – New data
**get** *(batch_size)*

Send the getting request. This function will be called by the optimizer.

- **Parameters** *batch_size* – The size of the minibatch
- **Returns** Minibatch of training data
Python Module Index

a
AlphaZero.env.go, 3
AlphaZero.env.mnk, 5
AlphaZero.env.reversi, 6
AlphaZero.evaluator.dummy_eval, 7
AlphaZero.evaluator.nn_eval_parallel, 7
AlphaZero.evaluator.nn_eval_seq, 7
AlphaZero.game.gameplay, 8
AlphaZero.network.main, 8
AlphaZero.network.model, 9
AlphaZero.player.cmd_player, 9
AlphaZero.player.mcts_player, 9
AlphaZero.player.nn_player, 10
AlphaZero.processing.go.game_converter, 10
AlphaZero.processing.selfplay2hdf, 12
AlphaZero.processing.state_converter, 12
AlphaZero.search.mcts, 13
AlphaZero.train.parallel.datapool, 16
AlphaZero.train.parallel.evaluator, 15
AlphaZero.train.parallel.selfplay, 16
Index

A
ack() (AlphaZero.player.cmd_player.Player method), 9
ack() (AlphaZero.player.mcts_player.Player method), 10
ack() (AlphaZero.player.nn_player.Player method), 10
AlphaZero.env.go (module), 3
AlphaZero.env.mnk (module), 5
AlphaZero.env.reversi (module), 6
AlphaZero.evaluator.dummy_eval (module), 7
AlphaZero.evaluator.nn_eval_parallel (module), 7
AlphaZero.evaluator.nn_eval_seq (module), 7
AlphaZero.game.gameplay (module), 8
AlphaZero.network.main (module), 8
AlphaZero.network.model (module), 9
AlphaZero.player.cmd_player (module), 9
AlphaZero.player.mcts_player (module), 9
AlphaZero.player.nn_player (module), 10
AlphaZero.processing.go.game_converter (module), 10
AlphaZero.processing.selfplay2hdf (module), 12
AlphaZero.processing.state_converter (module), 12
AlphaZero.search.mcts (module), 13
AlphaZero.train.parallel.datapool (module), 16
AlphaZero.train.parallel.evaluator (module), 15
AlphaZero.train.parallel.selfplay (module), 16

C
calc_move() (AlphaZero.search.mcts.MCTSearch method), 14
calc_move_with_probs() (AlphaZero.search.mcts.MCTSearch method), 14
convert_game() (AlphaZero.processing.go.game_converter.GameConverter method), 10
copy() (AlphaZero.env.go.GameState method), 4
copy() (AlphaZero.env.mnk.GameState method), 5
copy() (AlphaZero.env.reversi.GameState method), 6

D
DataPool (class in AlphaZero.train.parallel.datapool), 16
do_move() (AlphaZero.env.go.GameState method), 5
do_move() (AlphaZero.env.mnk.GameState method), 5
do_move() (AlphaZero.env.reversi.GameState method), 6

eval() (AlphaZero.evaluator.nn_eval_parallel.NNEvaluator method), 7
eval_wrapper() (AlphaZero.train.parallel.evaluator.Evaluator method), 15
evaluate() (AlphaZero.network.main.Network method), 9
Evaluator (class in AlphaZero.train.parallel.evaluator), 15
expand() (AlphaZero.search.mcts.MCTreeNode method), 13

G
Game (class in AlphaZero.game.gameplay), 8
GameConverter (class in AlphaZero.processing.go.game_converter), 10
GameState (class in AlphaZero.env.go), 3
GameState (class in AlphaZero.env.mnk), 5
GameState (class in AlphaZero.env.reversi), 6
get() (AlphaZero.train.parallel.datapool.DataPool method), 16
get_board_history() (AlphaZero.processing.state_converter.StateTensorConverter method), 12
get_current_player()  (AlphaZero.env.go.GameState method), 5
get_global_step()  (AlphaZero.network.main.Network method), 9
get_group()  (AlphaZero.env.go.GameState method), 3
get_groups_around()  (AlphaZero.env.go.GameState method), 3
global_step()  (AlphaZero.env.go.GameState method), 3
get_history()  (AlphaZero.game.gameplay.Game method), 8
get_legal_moves()  (AlphaZero.env.go.GameState method), 4
get_legal_moves()  (AlphaZero.env.mnk.GameState method), 5
get_legal_moves()  (AlphaZero.env.reversi.GameState method), 6
get_mean_action_value()  (AlphaZero.search.mcts.MCTreeNode method), 13
get_selection_value()  (AlphaZero.search.mcts.MCTreeNode method), 13
get_winner()  (AlphaZero.env.go.GameState method), 4
get_winner()  (AlphaZero.env.mnk.GameState method), 5
get_winner()  (AlphaZero.env.reversi.GameState method), 6
IllegalMove, 5–7
is_eye()  (AlphaZero.env.go.GameState method), 4
is_eyeish()  (AlphaZero.env.go.GameState method), 4
is_leaf()  (AlphaZero.search.mcts.MCTreeNode method), 13
is_legal()  (AlphaZero.env.go.GameState method), 4
is_legal()  (AlphaZero.env.mnk.GameState method), 5
is_legal()  (AlphaZero.env.reversi.GameState method), 6
is_positional_superko()  (AlphaZero.env.go.GameState method), 4
is_root()  (AlphaZero.search.mcts.MCTreeNode method), 14
is_suicide()  (AlphaZero.env.go.GameState method), 4

L
listen()  (AlphaZero.evaluator.nn_eval_parallel.NNEvaluator method), 7
load()  (AlphaZero.network.main.Network method), 9
lr_reflection()  (AlphaZero.processing.state_converter.ReverseTransformer method), 12
M
MCTreeNode (class in AlphaZero.search.mcts), 13
MCTSearch (class in AlphaZero.search.mcts), 14
merge_data()  (AlphaZero.train.parallel.datapool.DataPool method), 16
Model (class in AlphaZero.network.model), 9
model_update_handler()  (AlphaZero.train.parallel.selfplay.Selfplay method), 16
N
Network (class in AlphaZero.network.main), 8
NNEvaluator (class in AlphaZero.evaluator.nn_eval_parallel), 7
NoResultError, 10
P
place_handicap_stone()  (AlphaZero.env.go.GameState method), 4
place_handicaps()  (AlphaZero.env.go.GameState method), 4
Player (class in AlphaZero.player.cmd_player), 9
Player (class in AlphaZero.player.mcts_player), 9
Player (class in AlphaZero.player.nn_player), 10
put()  (AlphaZero.train.parallel.datapool.DataPool method), 16
R
randint()  (in module AlphaZero.search.mcts), 14
rcv_remote_data_handler()  (AlphaZero.train.parallel.selfplay.Selfplay method), 16
remote_update_handler()  (AlphaZero.train.parallel.selfplay.Selfplay method), 16
response()  (AlphaZero.network.main.Network method), 8
reverse_nprot90()  (AlphaZero.processing.state_converter.ReverseTransformer method), 12
reverse_transform()  (AlphaZero.processing.state_converter.ReverseTransformer method), 13
ReverseTransformer (class in AlphaZero.processing.state_converter), 12
run()  (AlphaZero.train.parallel.evaluator.Evaluator method), 16

22 Index