The handle [https://hdl.handle.net/1887/3209232](https://hdl.handle.net/1887/3209232) holds various files of this Leiden University dissertation.

**Author**: Wang, H.
**Title**: Searching by learning: Exploring artificial general intelligence on small board games by deep reinforcement learning
**Issue Date**: 2021-09-07
## Appendix A

### A.1 Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Description</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>$0 \leq \gamma \leq 1$</td>
<td>the discount factor of $\max_a Q(s', a')$</td>
<td>Eq. (2.1)</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$0 \leq \alpha \leq 1$</td>
<td>the learning rate of Q-learning</td>
<td>Eq. (2.2)</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>$0 \leq \epsilon \leq 1$</td>
<td>$\epsilon$-greedy for exploration and exploitation</td>
<td>Eq. (2.3)</td>
</tr>
<tr>
<td>$l$</td>
<td>$\mathbb{N}^+$</td>
<td>match number used to control decaying speed of $\epsilon$</td>
<td>Eq. (2.3)</td>
</tr>
<tr>
<td>$d$</td>
<td>$\mathbb{N}^+$</td>
<td>dimension of action space</td>
<td>Eq. (3.1)</td>
</tr>
<tr>
<td>$p$</td>
<td>$\mathbb{R}^d$</td>
<td>policy provided by the neural network</td>
<td>Eq. (3.1)</td>
</tr>
<tr>
<td>$\pi$</td>
<td>$\mathbb{R}^d$</td>
<td>improved estimate policy after performing MCTS</td>
<td>Eq. (3.1)</td>
</tr>
<tr>
<td>$v$</td>
<td>$\mathbb{R}$</td>
<td>state value prediction</td>
<td>Eq. (3.1)</td>
</tr>
<tr>
<td>$z$</td>
<td>${-1, 0, 1}$</td>
<td>real game end reward</td>
<td>Eq. (3.1)</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>$0 \leq \lambda \leq 1$</td>
<td>a weight to balance policy and value loss function</td>
<td>Eq. (4.1)</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\mathbb{R}$</td>
<td>a weight number to balance $U(s, a)$ and $U_{rave}(s, a)$</td>
<td>Eq. (5.4)</td>
</tr>
<tr>
<td>$L$</td>
<td>$\mathbb{N}^+$</td>
<td>the length of the reward list</td>
<td>Eq. (7.1)</td>
</tr>
<tr>
<td>$\tau$</td>
<td>$0 \leq \tau \leq 1$</td>
<td>a ratio to locate game length threshold in reward list</td>
<td>Eq. (7.1)</td>
</tr>
<tr>
<td>$r_\tau$</td>
<td>$\mathbb{N}^+$</td>
<td>threshold reward of game length to judge win or loss</td>
<td>Eq. (7.1)</td>
</tr>
</tbody>
</table>
### APPENDIX

#### A.2 Abbreviations

<table>
<thead>
<tr>
<th>Abb.</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>AGI</td>
<td>Artificial General Intelligence</td>
</tr>
<tr>
<td>MCS</td>
<td>Monte Carlo Search</td>
</tr>
<tr>
<td>MCTS</td>
<td>Monte Carlo Tree Search</td>
</tr>
<tr>
<td>GGP</td>
<td>General game playing</td>
</tr>
<tr>
<td>RAVE</td>
<td>Rapid Action Value Estimation</td>
</tr>
<tr>
<td>DQN</td>
<td>Deep Q-networks</td>
</tr>
<tr>
<td>GM</td>
<td>Game Manager</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>UCT</td>
<td>Upper Confidence bound applied to Trees</td>
</tr>
<tr>
<td>AMAF</td>
<td>All Moves As First</td>
</tr>
<tr>
<td>RHEA</td>
<td>Rolling Horizon Evolutionary Algorithm</td>
</tr>
<tr>
<td>P-UCT</td>
<td>Policy-Upper Confidence bound applied to Trees</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>TSP</td>
<td>Travelling Salesman Problems</td>
</tr>
<tr>
<td>BPP</td>
<td>Bin Packing Problems</td>
</tr>
<tr>
<td>R2</td>
<td>Ranked Reward</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Process</td>
</tr>
<tr>
<td>NRPA</td>
<td>Nested Rollout Policy Adaptation</td>
</tr>
</tbody>
</table>
A.3 Algorithms

Algorithm 8 Time Limited Monte Carlo Search Algorithm

1: function MONTECARLOSEARCH(time_limit)
2:     get legal actions set $A$ of current state $s$
3:     get next states set $S'$ where $s' \in S'$
4:     $z(s')=0$, $\text{count}(s')=0$
5:     while time_cost $\leq$ time_limit do
6:         for each $s'$ in $S'$ do
7:             outcome($s'$)$\leftarrow$random simulation from $s'$ to game end.
8:             $z(s')+=\text{outcome}(s')$
9:             $\text{count}(s')+=1$
10:            selected_action$\leftarrow$ getActionFromStates($s$, $\arg\max_{s'\in S'} \frac{z(s')}{\text{count}(s')}\}$
11:     return selected_action
Algorithm 9 QM-learning Enhancement

1: function QMPLAYER(current state $s$, learning rate $\alpha$, discount factor $\gamma$, Q table: $Q(S, A)$)
2:     for each match do
3:         if $s$ terminates then
4:             for each $(s, a)$ from end to the start in current match record do
5:                 $R(s, a) = s'$ is terminal state? getGoal($s'$, myrole) : 0
6:                 Update $Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha (R(s, a) + \gamma \max_a Q(s', a'))$
7:         else
8:             if $\epsilon$-greedy is enabled then
9:                 selected_action = Random()
10:            else
11:                selected_action = SelectFromQTable()
12:                if no $s$ record in $Q(S, A)$ then
13:                   MonteCarloSearch(time_limit)
14:                performAction($s$, selected_action)
15:     return $Q(S, A)$
Algorithm 10 Neural Network Based MCTS with Only Rollout Simulation Value

1: function ROLLOUT(s, fθ)
2:     Search(s)
3:     πs ← normalize(Q(s, ⋅))
4:     return πs
5: function SEARCH(s)
6:     Return game end result if s is a terminal state
7:     if s is not in the Tree then
8:         Add s to the Tree, initialize Q(s, ⋅) and N(s, ⋅) to 0
9:         Get P(s, ⋅) and v(s) by looking up fθ(s)
10:        Get result v(s) by performing random rollout until the game ends
11:        return v(s)
12:     else
13:         Select an action a with highest UCT value
14:         s′ ← getNextState(s, a)
15:         v ← Search(s′)
16:         Q(s, a) ← \frac{N(s, a) \cdot Q(s, a) + v}{N(s, a) + 1}
17:         N(s, a) ← N(s, a) + 1
18:     return v;
Algorithm 11 Neural Network Based MCTS with Only RAVE Value

1: function RAVE(s, fθ)
2: Search(s)
3: πs ← normalize(Qrave(s, ·))
4: return πs
5: function Search(s)
6: Return game end result if s is a terminal state
7: if s is not in the Tree then
8: Add s to the Tree,
9: Initialize Q(s, ·), N(s, ·), Qrave(s, ·) and Nrave(s, ·) to 0.
10: Get P(s, ·) and v(s) by looking up fθ(s)
11: return v(s)
12: else
13: Select an action a with highest UCTrave value
14: st ← getNextState(s, a)
15: v ← Search(st)
16: Q(s, a) ← \frac{N(s,a)Q(s,a)+v}{N(s,a)+1}
17: N(s, a) ← N(s, a) + 1
18: Nrave(st1, at1) ← Nrave(st1, at1) + 1
19: Qrave(st1, at1) ← \frac{Nrave(st1, at1)Qrave(st1, at1)+v}{Nrave(st1, at2)+1}
20: ▷ where st1 ∈ VisitedPath, and at1 ∈ A(st1), and for ∀t < t2, at ≠ at2
21: return v;
A.3 Algorithms

**Algorithm 12** Neural Network Based MCTS with Rollout Simulation and RAVE Value

1: function `RoRa(s, f_θ)`
2: Search(s)
3: \( π_s ← \text{normalize}(Q_{rave}(s, \cdot)) \)
4: return \( π_s \)
5: function `Search(s)`
6: Return game end result if \( s \) is a terminal state
7: if \( s \) is not in the Tree then
8: Add \( s \) to the Tree,
9: Initialize \( Q(s, \cdot), N(s, \cdot), Q_{rave}(s, \cdot) \) and \( N_{rave}(s, \cdot) \) to 0.
10: Get \( P(s, \cdot) \) and \( v(s) \) by looking up \( f_θ(s) \)
11: Get result \( v(s) \) by performing random rollout until the game ends
12: return \( v(s) \)
else
13: Select an action \( a \) with highest UCT\(_{rave} \) value
14: \( s' ← \text{getNextState}(s, a) \)
15: \( v ← \text{Search}(s') \)
16: \( Q(s, a) ← \frac{N(s,a)\cdot Q(s,a)+v}{N(s,a)+1} \)
17: \( N(s,a) ← N(s,a) + 1 \)
18: \( N_{rave}(s_{t1}, a_{t2}) ← N_{rave}(s_{t1}, a_{t2}) + 1 \)
19: \( Q_{rave}(s_{t1}, a_{t2}) ← \frac{N_{rave}(s_{t1}, a_{t2})\cdot Q_{rave}(s_{t1}, a_{t2})+v}{N_{rave}(s_{t1}, a_{t2})+1} \)
20: \( \triangleright \text{where } s_{t1} ∈ VisitedPath, \text{ and } a_{t2} ∈ A(s_{t1}), \text{ and for } ∀t < t_2, a_t ≠ a_{t2} \)
21: return \( v; \)
APPENDIX

Algorithm 13 Neural Network Based MCTS with Neural Network and Rollout Simulation Value

1: function WRO($s, f_\theta$)
2:    Search($s$)
3:    $\pi_s \leftarrow \text{normalize}(Q(s, \cdot))$
4:    return $\pi_s$
5: function Search($s$)
6:    Return game end result if $s$ is a terminal state
7:    if $s$ is not in the Tree then
8:        Add $s$ to the Tree, initialize $Q(s, \cdot)$ and $N(s, \cdot)$ to 0
9:        Get $P(s, \cdot)$ and $v(s)_{\text{network}}$ by looking up $f_\theta(s)$
10:       Get result $v(s)_{\text{rollout}}$ by performing random rollout until the game ends
11:       $v(s) = (1 - \text{weight}) \times v_{\text{network}} + \text{weight} \times v_{\text{rollout}}$
12:       return $v(s)$
13:    else
14:        Select an action $a$ with highest UCT value
15:        $s' \leftarrow \text{getNextState}(s, a)$
16:        $v \leftarrow \text{Search}(s')$
17:        $Q(s, a) \leftarrow \frac{N(s,a)Q(s,a)+v}{N(s,a)+1}$
18:        $N(s,a) \leftarrow N(s,a) + 1$
19:    return $v$;
Algorithm 14 Neural Network Based MCTS with Neural Network, Rave and Rollout Simulation Value

1: function \text{WRoRa}(s, f_{\theta})
2: \hspace{1em} \text{Search}(s)
3: \hspace{1em} \pi_s \leftarrow \text{normalize} (Q(s, \cdot))
4: \hspace{1em} \text{return} \pi_s
5: function \text{Search}(s)
6: \hspace{1em} \text{Return game end result if } s \text{ is a terminal state}
7: \hspace{1em} \text{if } s \text{ is not in the Tree then}
8: \hspace{2em} \text{Add } s \text{ to the Tree,}
9: \hspace{2em} \text{Initialize } Q(s, \cdot), N(s, \cdot), Q_{\text{rave}}(s, \cdot) \text{ and } N_{\text{rave}}(s, \cdot) \text{ to 0.}
10: \hspace{2em} \text{Get } P(s, \cdot) \text{ and } v(s)_{\text{network}} \text{ by looking up } f_{\theta}(s)
11: \hspace{2em} \text{Get result } v(s)_{\text{rollout}} \text{ by performing random rollout until the game ends}
12: \hspace{2em} \text{random rollout path added to } \text{VisitedPath}
13: \hspace{2em} v(s) = (1 - \text{weight}) \ast v_{\text{network}} + \text{weight} \ast v_{\text{rollout}}
14: \hspace{2em} \text{return } v(s)
15: \hspace{1em} \text{else}
16: \hspace{2em} \text{Select an action } a \text{ with highest } UCT_{\text{rave}} \text{ value}
17: \hspace{2em} s' \leftarrow \text{getNextState}(s, a)
18: \hspace{2em} v \leftarrow \text{Search}(s')
19: \hspace{2em} Q(s, a) \leftarrow \frac{N(s,a) \ast Q(s,a)+v}{N(s,a)+1}
20: \hspace{2em} N(s, a) \leftarrow N(s, a) + 1
21: \hspace{2em} N_{\text{rave}}(s_t, a_{t_2}) \leftarrow N_{\text{rave}}(s_t, a_{t_2}) + 1
22: \hspace{2em} Q_{\text{rave}}(s_t, a_{t_2}) \leftarrow \frac{N_{\text{rave}}(s_t, a_{t_2}) \ast Q_{\text{rave}}(s_t, a_{t_2})+v}{N_{\text{rave}}(s_t, a_{t_2})+1}
23: \hspace{2em} \triangleright \text{ where } s_t \in \text{VisitedPath}, \text{ and } a_{t_2} \in A(s_t), \text{ and for } \forall t < t_2, a_t \neq a_{t_2}
24: \hspace{2em} \text{return } v;
Algorithm 15 Rolling Horizon Evolutionary Algorithm

1: function RHEA(s, time\_limit)
2:     Set up population of $n$ valid action sequences of length $l$: $A_{n,l}$
3:     for all $A_{i<n}$ do Evaluate($A_i$)
4:         repeat
5:             new action sequence $A_j$ = mutate one randomly chosen action sequence
6:             by changing every move with a small random chance
7:             $f(A_j) = $ Evaluate($A_j$)
8:             add $A_j$ to population
9:             remove $A_i$ with worst $f(A_i)$ from population
10:         until time\_cost $\geq$ time\_limit
11:     return first action of best sequence in population
12: function Evaluate($A_i$)
13:     repeat
14:         Play action sequence in $A_i$
15:         Get result success, game\_steps by performing random rollout until
16:         the game ends
17:     until repetitions $\geq$ 2
18:     compute fitness $f(A_i)$ from average success probability with sequence
19:         length penalty (line 117)
20:     return $f(A_i)$
A.4 Elo Computation

In this dissertation, like AlphaZero series papers did, a whole history Bayesian Elo computation \[82\] is also employed to present the relative competence of playing the game of different trained models instead of a win or loss rate. In this section, a full computation process will be described in detail based on the Bayesian Elo computation system (called Bayeselo) provided on github \[131\].

Bayeselo is a free software tool to compute Elo ratings. It receives a file containing game records written in PGN (Portable Game Notation) format \[132\], and produces a rating list \[131\]. Therefore, a full process can be simply described in Fig. A.1.

![Figure A.1: a full Bayesian Elo Computation Process.](image)

An example of part of a PGN file (arena_othello_final.pgn) generated based on win/loss results recorded during AlphaZero-like self-play arena competition is shown as Fig. A.2.
Figure A.2: Small Part of PGN file arena_othello_final.pgn. The file contains much such format iterative arena competition information. Each pair of White and Black players played 20 rounds.

An example of elo rating list generated by operating Bayeselo system with PGN file (arena_othello_final.pgn) as input is shown as follows. See Fig. A.3. The figures of elo ratings in this dissertation are visualized based on such elo rating lists.
### Table A.3: An example of Generated Elo Rating List by Bayeselo

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Elo</th>
<th>+</th>
<th>-</th>
<th>games</th>
<th>score</th>
<th>oppo.</th>
<th>draws</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bestmodel_mcts_rave_rollout_run6</td>
<td>117</td>
<td>22</td>
<td>22</td>
<td>960</td>
<td>62%</td>
<td>-2</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>bestmodel_weight_mcts_rave_rollout_run5</td>
<td>102</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>57%</td>
<td>-2</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>bestmodel_weight_mcts_rave_rollout_run2</td>
<td>100</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>58%</td>
<td>-2</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>bestmodel_weight_mcts_rollout_run5</td>
<td>97</td>
<td>22</td>
<td>21</td>
<td>960</td>
<td>58%</td>
<td>-2</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>bestmodel_mcts_rave_run1</td>
<td>86</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>58%</td>
<td>-2</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>bestmodel_weight_mcts_rave_rollout_run8</td>
<td>81</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>55%</td>
<td>-2</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>bestmodel_pi_v_run1</td>
<td>77</td>
<td>22</td>
<td>21</td>
<td>960</td>
<td>61%</td>
<td>-2</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>bestmodel_weight_mcts_rave_rollout_run3</td>
<td>71</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>54%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>9</td>
<td>bestmodel_mcts_rave_rollout_run2</td>
<td>62</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>57%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>10</td>
<td>bestmodel_weight_mcts_rave_rollout_run1</td>
<td>54</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>53%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>11</td>
<td>bestmodel_weight_mcts_rave_rollout_run6</td>
<td>53</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>52%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>12</td>
<td>bestmodel_mcts_rave_run4</td>
<td>53</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>54%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>13</td>
<td>bestmodel_weight_mcts_rave_rollout_run4</td>
<td>52</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>52%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>14</td>
<td>bestmodel_weight_mcts_rollout_run3</td>
<td>44</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>52%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>15</td>
<td>bestmodel_weight_mcts_rollout_run8</td>
<td>39</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>51%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>16</td>
<td>bestmodel_pi_v_run4</td>
<td>39</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>56%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>17</td>
<td>bestmodel_weight_mcts_rave_rollout_run7</td>
<td>36</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>50%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>18</td>
<td>bestmodel_weight_mcts_rollout_run7</td>
<td>34</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>51%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>19</td>
<td>bestmodel_mcts_rave_run7</td>
<td>32</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>51%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>20</td>
<td>bestmodel_weight_mcts_rollout_run2</td>
<td>30</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>51%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>21</td>
<td>bestmodel_mcts_rollout_run5</td>
<td>28</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>52%</td>
<td>-1</td>
<td>0%</td>
</tr>
<tr>
<td>22</td>
<td>bestmodel_mcts_rave_run2</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>51%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>23</td>
<td>bestmodel_weight_mcts_rollout_run4</td>
<td>20</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>49%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>24</td>
<td>bestmodel_weight_mcts_rollout_run1</td>
<td>20</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>50%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>25</td>
<td>bestmodel_pi_v_run6</td>
<td>18</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>53%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>26</td>
<td>bestmodel_mcts_rollout_run3</td>
<td>17</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>53%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>27</td>
<td>bestmodel_mcts_rave_rollout_run7</td>
<td>15</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>51%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>28</td>
<td>bestmodel_mcts_rollout_run1</td>
<td>11</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>52%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>29</td>
<td>bestmodel_mcts_rave_run8</td>
<td>10</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>49%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>30</td>
<td>bestmodel_weight_mcts_rollout_run6</td>
<td>8</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>48%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>31</td>
<td>bestmodel_mcts_rollout_run2</td>
<td>7</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>52%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>32</td>
<td>bestmodel_mcts_rave_rollout_run8</td>
<td>6</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>49%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>33</td>
<td>bestmodel_mcts_rave_run5</td>
<td>5</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>49%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>34</td>
<td>bestmodel_mcts_rave_run6</td>
<td>4</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>48%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>35</td>
<td>bestmodel_mcts_rave_rollout_run3</td>
<td>2</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>50%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>36</td>
<td>bestmodel_pi_v_run7</td>
<td>-2</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>51%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>37</td>
<td>bestmodel_mcts_rollout_run6</td>
<td>-6</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>49%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>38</td>
<td>bestmodel_pi_v_run3</td>
<td>-7</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>51%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>39</td>
<td>bestmodel_mcts_rollout_run5</td>
<td>-8</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>49%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>40</td>
<td>bestmodel_mcts_rave_rollout_run4</td>
<td>-10</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>48%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>41</td>
<td>bestmodel_mcts_rollout_run7</td>
<td>-11</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>49%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>42</td>
<td>bestmodel_mcts_rave_rollout_run1</td>
<td>-17</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>48%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>43</td>
<td>bestmodel_pi_v_run8</td>
<td>-17</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>49%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>44</td>
<td>bestmodel_mcts_rollout_run8</td>
<td>-43</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>45%</td>
<td>1</td>
<td>0%</td>
</tr>
<tr>
<td>45</td>
<td>bestmodel_pi_v_run5</td>
<td>-65</td>
<td>21</td>
<td>21</td>
<td>960</td>
<td>44%</td>
<td>1</td>
<td>0%</td>
</tr>
<tr>
<td>46</td>
<td>bestmodel_mcts_rave_run3</td>
<td>-66</td>
<td>22</td>
<td>22</td>
<td>960</td>
<td>41%</td>
<td>1</td>
<td>0%</td>
</tr>
<tr>
<td>47</td>
<td>bestmodel_pi_v_run2</td>
<td>-100</td>
<td>22</td>
<td>22</td>
<td>960</td>
<td>41%</td>
<td>2</td>
<td>0%</td>
</tr>
<tr>
<td>48</td>
<td>bestmodel_mcts_rollout_run4</td>
<td>-109</td>
<td>22</td>
<td>22</td>
<td>960</td>
<td>38%</td>
<td>2</td>
<td>0%</td>
</tr>
<tr>
<td>49</td>
<td>randomplayer</td>
<td>-988</td>
<td>124</td>
<td>204</td>
<td>960</td>
<td>0%</td>
<td>21</td>
<td>0%</td>
</tr>
</tbody>
</table>
Bibliography


BIBLIOGRAPHY


