Minmax vs MCTS on Connect4 - results

1 Minmax implementation

Minmax player was implemented in a customizable way:

- exploration range is controlled both via time given per decision and considered depth of a search
- state evaluation function is injected while creating player

Moreover, player uses transposition table so as to exploit previous research and redundant computation. The key in table is the state being considered. The values are 3-tuples: (*depth*, *value*, *best_move*). *depth* is the height of the tree that was taken into account while computing *value* for the key, resulting in selecting *best_move*. Of course, results from search in higher tree would overwrite weaker stored data.

Evaluation function firstly checks whether there exists a single game-ending move. If there is no such a move, then for each 4 adjacent fields in grid (in every 4 directions), it computes how many there are discs of a single color and multiplies it by a corresponding weight and a ± 1 factor (depending on the color).

To reduce branching and deduce as much as possible from already computed values, alpha-beta pruning was harnessed.

The exploration is performed in iterative-deepening manner. It enables us to have breadth-full view all the time in small memory space. Simultaneously it defines the order of moves being considered.

2 Monte Carlo Tree Search implementation

MCTS player was implemented according to the standard approach, i.e. with selection, expansion, rollout and back-propagation phases. In expansion stage, the best child option is chosen due to *UCB* value:

$$UCB(x_i) = \frac{w_i}{n_i} + 2\sqrt{\frac{ln\ N}{n_i}}$$

Both after expanding a leaf node and the rollout, the next considered node is chosen randomly from uniform distribution.

The nodes are stored in a hash map. The key is a pair of state and currently considered color. Values are 3-tuples: (*wins*, *visits*, *is_leaf*) - these are used for computing *UCB* and selecting moves.

Before any search tree, an initial check is performed to see whether there is a single move, that would finish the game.

Exploration range can be controlled via both timeout and number of iterations performed for a single decision.

The final decision made in the root of the search tree is simply choosing the most often visited node. After reasonable amount of exploration, *UCB* should more often select better nodes according to the first part in the formula enabling exploitation.

3 Results

Comparison was made adjusting 2 values:

- timeout given to both players - either equal or somehow biased
- whether players are memorable - i.e. whether new players were created for each single game, or there was a single player for all games

3.1 Quick movements (1 second per move)

```
Minmax vs MCTS winning rate (timeout: 1s, new player for each game): 0.4
Minmax vs MCTS winning rate (timeout: 1s, new player for each game): 0.34
Minmax vs MCTS winning rate (timeout: 1s, new player for each game): 0.3
Minmax vs MCTS winning rate (timeout: 1s, new player for each game): 0.48
Minmax vs MCTS winning rate (timeout: 1s, new player for each game): 0.4

Minmax vs MCTS winning rate (timeout: 1s, one player for all games): 0.34
Minmax vs MCTS winning rate (timeout: 1s, one player for all games): 0.24
Minmax vs MCTS winning rate (timeout: 1s, one player for all games): 0.27
Minmax vs MCTS winning rate (timeout: 1s, one player for all games): 0.28
Minmax vs MCTS winning rate (timeout: 1s, one player for all games): 0.22
```

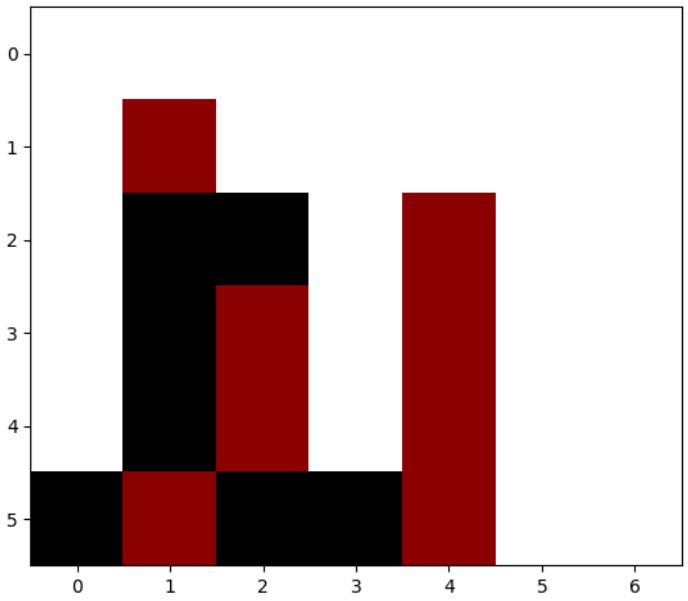

Figure 1: Sample 1sec-game (Minmax is red)

3.2 2 seconds per move

```
Minmax vs MCTS winning rate (timeout: 2s, new player for each game): 0.24
Minmax vs MCTS winning rate (timeout: 2s, new player for each game): 0.36
Minmax vs MCTS winning rate (timeout: 2s, new player for each game): 0.41
Minmax vs MCTS winning rate (timeout: 2s, new player for each game): 0.32
Minmax vs MCTS winning rate (timeout: 2s, new player for each game): 0.35

Minmax vs MCTS winning rate (timeout: 2s, one player for all games): 0.5
Minmax vs MCTS winning rate (timeout: 2s, one player for all games): 0.47
Minmax vs MCTS winning rate (timeout: 2s, one player for all games): 0.33
Minmax vs MCTS winning rate (timeout: 2s, one player for all games): 0.34
Minmax vs MCTS winning rate (timeout: 2s, one player for all games): 0.42
```

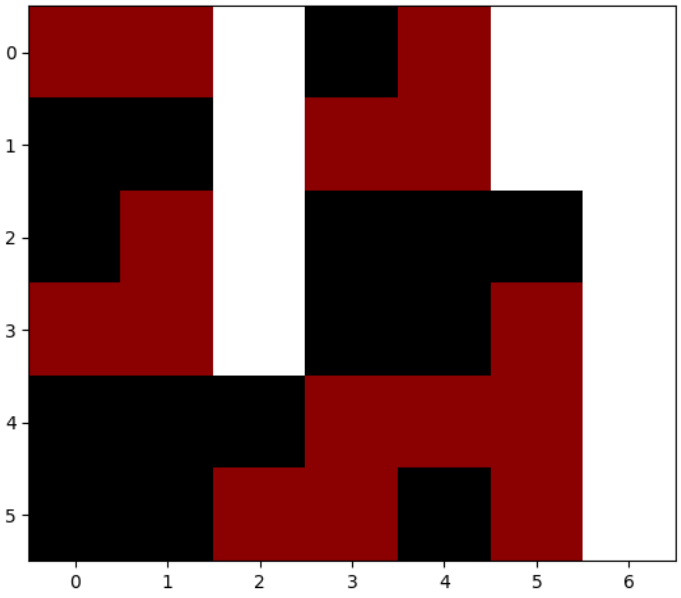

Figure 2: Sample 2sec-game (Minmax is red)

3.3 5 seconds per move

```
Minmax vs MCTS winning rate (timeout: 5s, new player for each game): 0.37
Minmax vs MCTS winning rate (timeout: 5s, new player for each game): 0.61
Minmax vs MCTS winning rate (timeout: 5s, new player for each game): 0.33
Minmax vs MCTS winning rate (timeout: 5s, new player for each game): 0.54

Minmax vs MCTS winning rate (timeout: 5s, one player for all games): 0.32
Minmax vs MCTS winning rate (timeout: 5s, one player for all games): 0.42
Minmax vs MCTS winning rate (timeout: 5s, one player for all games): 0.25
Minmax vs MCTS winning rate (timeout: 5s, one player for all games): 0.5
```

3.4 10 seconds per move

```
Minmax vs MCTS winning rate (timeout: 10s, new player for each game): 0.25
Minmax vs MCTS winning rate (timeout: 10s, new player for each game): 0.5
Minmax vs MCTS winning rate (timeout: 10s, new player for each game): 0.25

Minmax vs MCTS winning rate (timeout: 10s, one player for all games): 0.5
Minmax vs MCTS winning rate (timeout: 10s, one player for all games): 0.5
Minmax vs MCTS winning rate (timeout: 10s, one player for all games): 0.5
```

3.5 Remarks

MCTS could have done about 200-300 iterations per second (530-620 per 2s, 1340-1600 per 5s).

Minmax was reaching about 3 levels down in first second, next 2 levels in next second. Given 5 seconds it was able to reach 7th level (thanks to transposition table).

Test batches were of size 10-50 games, depending on the time given to players for decision.

3.6 Conclusion

It is easy to see, that both players, given either more time for decision or more games are converging to the optimum. However, it seems that MCTS performs slightly better in earlier learning stage and in quick games.

Charts show that time is the factor that enables both approaches to become more conscious and to better plan their moves.