TCG HW2

B04705003 資工四 林子雋

1. How to compile your code into an agent

$ cd b04705003/source
$ make b04705003
And it will generate `b04705003` agent.

2. What algorithms and heuristics you've implemented

Setting:

At first, I was wondering if I need to **save 'Board' into each node**. Therefore, I code one version of saving 'Board' into each node and one version of not saving it. Then I find out that **saving 'Board' into each node turns out to be as fast as not saving 'Board'**. In the below setting, I use saving 'Board' into each node.

Heuristics:

In an end game board, I calculate the score as follow matrix:

	For Max node	For Min node
Win	6 + (Max node remaining cubes) – (Min node remaining cubes)	12 + (Min node remaining cubes) – (Max node remaining cubes)
Lose	-12 + (Max node remaining cubes) – (Min node remaining cubes)	-6 + (Min node remaining cubes) – (Max node remaining cubes)

The reason why I add 6 and subtract -12 is **to make Max node afraid of dying and make Min node more aggressive to kill Max node**.

Algorithm:

I implement:

- **b04705003** – MCTS + Progressive Pruning + Tree Preserving
  - **Tree Preserving** means that I will store the tree expanded during this phase.
  - To compile, run as command as first section.
- UCB score(win/lose) without tree search
  - See ./b04705003/source/ucb_winlose.cc
  - To compile, run:
    - $ cd ./b04705003/source
    - $ make ucb_winlose
    - And it will generate 'ucb_winlose' executable file.
- UCB score(win/lose) with tree search
  - See ./b04705003/source/ucb_winlose.cc
    - $ cd ./b04705003/source
    - $ make ucb_winlose_wtree
    - And it will generate 'ucb_winlose_wtree' executable file.
- **MCTS_pure** - UCB score(12~-18 end game board score) with tree search

- See ./b04705003/source/mcts.cc
- To compile, run:
  - $ cd ./b04705003/source
  - $ make mcts
  - And it will generate 'mcts' executable file.

- **MCTS_wpropru** - UCB score(12~-18 end game board score) with tree search with progressive pruning
  - See ./b04705003/source/mcts.cc
  - To compile, run:
    - $ cd ./b04705003/source
    - $ make mcts_wpropru
    - And it will generate 'mcts_wpropru' executable file.
- **MCTS_rave** - UCB score(12~-18 end game board score) with tree search with RAVE
  - Note that I only do linear combination of non-AMAF and AMAF on mean value(i.e. I didn't do linear combination on variance).
  - See ./b04705003/source/mcts.cc
  - To compile, run:
    - $ cd ./b04705003/source
    - $ make mcts_rave
    - And it will generate 'mcts_rave' executable file.
- **MCTS_alphabeta** – UCB score(12~-18 end game board score) with tree search and use leaf node simulation result to do alpha beta pruning.
  - See ./b04705003/source/mcts.cc
  - To compile, run:
    - $ cd ./b04705003/source
    - $ make mcts_alphabeta
    - And it will generate 'mcts_alphabeta' executable file.

3. Experiments and Analysis

Simulation Number Per Node Experiments

Experiment Intuition:

Because I find out that the scores among depth 1 children(depth 0 is root board) are close, I want to do some experiments to check whether tree expansion is more important than simulation on each node.

Experiment Setting:

- Simulation time = 8 seconds
- c = 1.18
- c1 = 2
- c2 = 3
- r_d = 1
- sigma_e = 0.7
- round = 100 (-r flag on ./game)

Experiment Results:

UCB score(win/lose) without tree search

None(Because there is no tree expansion here)

UCB score(win/lose) with tree search

Versus 'random' agent	Winning rate
Per node simulation = 1000	100%
1500	100%
2000	100%

Versus 'greedy' agent	Winning rate
Per node simulation = 1000	65%
1500	69%
2000	71%

MCTS_pure - UCB score(12~-18 end game board score) with tree search

Versus 'random' agent	Winning rate
Per node simulation = 1000	100
1500	100
2000	100

Versus 'greedy' agent	Winning rate
Per node simulation = 1000	98%
1500	98%
2000	99%

MCTS_wpropru - UCB score(12~-18 end game board score) with tree search with progressive pruning

Versus 'random' agent	Winning rate
Per node simulation = 1000	100%
1500	100%
2000	100%

Versus 'greedy' agent	Winning rate
Per node simulation = 1000	87%
1500	98%
2000	97%

MCTS_rave - UCB score(12~-18 end game board score) with tree search with RAVE

1. Versus 'random' agent	Winning rate
Per node simulation = 1000	99%
1500	100%
2000	100%

Versus ‘greedy’ agent	Winning rate
Per node simulation = 1000	92%
1500	91%
2000	93%

Analysis:
We can find out that the more simulation perform on one node, the better result against`random` and `greedy` agent.

Compared With My Friend’s Fast AlphaBeta Pruning Agent Experiment:

Experiment Intuition:
At first, I thought maybe more simulation per node is a good idea. But as I test my agent to compete alpha beta pruning agent, I finally figure out maybe more tree structure is more important than more simulation per node.

My friend’s alpha beta is super fast(can search 18 depths). I guess TA will use alpha beta to kill our agent. Therefore, I want to test which simulation number per node is suitable to compete with purely alpha beta pruning agent.

Experiment Setting:

- Simulation time = 8 seconds
- c = 1.18
- c1 = 2
- c2 = 3
- r_d = 9
- sigma_e = 0.2
- round = 100 (-r flag on ./game)

MCTS_pure - UCB score(12~-18 end game board score) with tree search

Versus ‘purely alpha beta’ agent	Winning rate
Per node simulation = 70	40.32%
80	32.14%
90	34.88%
100	36.5%
200	28.2%
300	21.5%
400	19.10%
500	17.0%
1000	14.9%
1500	22.2%
2000	17.4%

MCTS_alphabeta – UCB score(12~-18 end game board score) with tree search and use leaf node simulation result to do alpha beta pruning.

Versus 'purely alpha beta' agent	Winning rate
Per node simulation = 100	21.74%
200	28.88%
300	25.56%
400	26.37%

MCTS_wpropru - UCB score(12~-18 end game board score) with tree search with progressive pruning

Versus 'purely alpha beta' agent	Winning rate
Per node simulation = 90	32.89%
100	36.96%
200	22.99%

b04705003 – MCTS + Progressive Pruning + Tree Preserving

Versus 'purely alpha beta' agent	Winning rate
Per node simulation = 80	30.88%
90	33.82%
100	33.33%
200	32.79%

Analysis:

By the experiment, we can find out that tree structure is really important.