Udacity Build an Adversarial Game Agent

Problem description

Experiment with adversarial search techniques by building an agent to play knights Isolation. Unlike the examples in lecture where the players control tokens that move like chess queens, this version of Isolation gives each agent control over a single token that moves in L-shaped movements--like a knight in chess.

Model implementation

From the 3 options offered to develop, the first one was considered developing a custom heuristic based on the alpha beta search as baseline based in the implementation exampled in the course lesson 4. In the case:

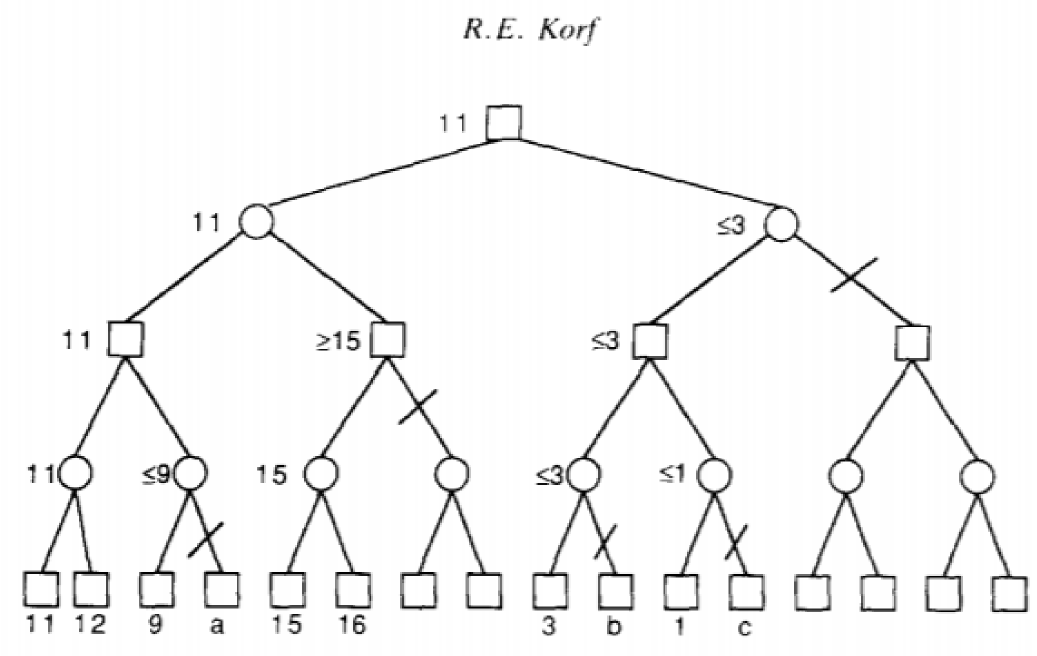

Fig. 1. Two-player alpha-beta pruning.

For this implementation, every directional minimax search algorithm must evaluate every leaf node evaluated by alpha-beta under the same ordering.

Experiments running

To run the experiments was used the interact command to execute the game:

- python run_match.py -r 50 -o GREEDY -t 150

For running the experiments were considered this parameters values:

- Time limit: 100 y 150 ms
- Opponent model: GREEDY, MINIMAX, SELF, RANDOM
- Depth: 3, 5, 7 (Parameter DEFAULT_DEPTH inside my_custom_player.py)
- Matches: 100 (50 rounds)

To facilitate the execution was created a shell script named experiments.sh that executes these calls:

```
python run_match.py -r 50 -o GREEDY -t 100
python run_match.py -r 50 -o MINIMAX -t 100
python run_match.py -r 50 -o SELF -t 100
python run_match.py -r 50 -o RANDOM -t 100
python run_match.py -r 50 -o GREEDY -t 150
python run_match.py -r 50 -o MINIMAX -t 150
python run_match.py -r 50 -o SELF -t 150
python run_match.py -r 50 -o RANDOM -t 150
```

is necessary assign the right permits to execute the script:

```
chmod u+x experiments.sh
```

Experiment Results

Experiment number	DEPTH	OPONNENT AGENT	AGENT WIN RATE	MATCHES	TIME LIMIT
1	3	MINIMAX	79%	100	100
2	3	GREEEDY	62%	100	100
3	3	RANDOM	98%	100	100
4	3	SELF	51%	100	150
5	3	MINIMAX	71%	100	150
6	3	GREEEDY	58%	100	150
7	3	RANDOM	48%	100	150
8	3	SELF	96%	100	100
9	5	MINIMAX	81%	100	100
10	5	GREEEDY	63%	100	100
11	5	RANDOM	99%	100	100

12	5	SELF	51%	100	150
13	5	MINIMAX	66%	100	150
14	5	GREEEDY	83%	100	150
15	5	RANDOM	95%	100	100
16	5	SELF	50%	100	100

9	7	MINIMAX	86%	100	100
10	7	GREEEDY	65%	100	100
11	7	RANDOM	99%	100	100
12	7	SELF	50%	100	150
13	7	MINIMAX	67%	100	150
15	7	GREEEDY	85%	100	150
16	7	RANDOM	98%	100	100
17	7	SELF	50%	100	100

As a result, deeper search extended the execution significantly, with SELF opponent the result is close to 50% as expected because the agent just copies the same search strategy. In all cases the custom agent over performs the random strategy. For the MINIMAX, the agent obtains a mean of 75% wins and GREEDY 69%. That shows that the agent strategy is not goof against the GREEDY agent and better in the matches with MINIMAX.

Questions

- **What features of the game does your heuristic incorporate, and why do you think those features matter in evaluating states during search?**

  The search incorporates some changes over the base search implementation in the lesson 3 for the beta search pruning. Some changes in the number of liberties from the player and player 2 using a weight parameter than change the bias between the two players.

- **Analyze the search depth your agent achieves using your custom heuristic. Does search speed matter more or less than accuracy to the performance of your heuristic?**

  The agent considers the depth parameter in the search. This parameter had an important impact on performance, but in the results, only achieved close to 10% improvement in some cases. Deeper search does not mean better result in all cases.