Artificial Intelligence Nanodegree Program

Project#3: Build an Adversarial Game Playing Agent

March 21st, 2022

(1) Game Agent Implementation

➔ **my_custom_player.py**
(minimax and alpha-beta search algorithm)

(2) Experimental Results & Report

(a)CustomAgent search function uses an advanced search technique

I chose Option 1: Develop a custom heuristic.

I implemented the following 3 heuristic functions.

(i) Difference between my moves and opponent's moves (default heuristic)

I implemented the default heuristic based on the lecture.

I also tuned the balance between my moves and opponent's moves.

```
136     # add score function
137     def score(self, state):
138         own_loc = state.locs[self.player_id]
139         opp_loc = state.locs[1 - self.player_id]
140         own_liberties = state.liberties(own_loc)
141         opp_liberties = state.liberties(opp_loc)
142         #return len(own_liberties) - len(opp_liberties) # heuristic 1 (original)
143         #return len(own_liberties) - 2 * len(opp_liberties) # heuristic 2
144         #return 2 * len(own_liberties) - len(opp_liberties) # heuristic 3
145         #return len(own_liberties) - 1.5 * len(opp_liberties) # heuristic 4
146         #return 1.5 * len(own_liberties) - len(opp_liberties) # heuristic 5
147         #return len(own_liberties) - 1.2 * len(opp_liberties) # heuristic 6
148         #return 1.2 * len(own_liberties) - len(opp_liberties) # heuristic 7
149         return 1.1 * len(own_liberties) - len(opp_liberties) # heuristic 8
```

(ii) How close from the center (original#1 heuristic)

Knight cannot move well near the corner, so I created an original heuristic that calculate how close from the center. I subtract it from the max distance: sqrt(4*4 + 5*5), so the bigger score the better result.

```
151     def score2(self, state):
152         _WIDTH = 11
153         own_loc = state.locs[self.player_id]
154         center_loc = 57
155         x, y = own_loc % (_WIDTH + 2), own_loc // (_WIDTH + 2)
156         cx, cy =  center_loc % (_WIDTH + 2), center_loc // (_WIDTH + 2)
157         center_distance = math.sqrt((cx-x)*(cx-x) + (cy-y)*(cy-y))
158         return  math.sqrt(41) - center_distance # how close from center point (farest point is 0) # heuristic 9
```

(iii) Combination of default heuristic and original heuristic (original#2 heuristic)

I combine the 2 heuristics by multiplying weights.

```
160     def score3(self, state):
161         _WIDTH = 11
162         own_loc = state.locs[self.player_id]
163         opp_loc = state.locs[1 - self.player_id]
164         own_liberties = state.liberties(own_loc)
165         opp_liberties = state.liberties(opp_loc)
166         center_loc = 57
167         x, y = own_loc % (_WIDTH + 2), own_loc // (_WIDTH + 2)
168         cx, cy =  center_loc % (_WIDTH + 2), center_loc // (_WIDTH + 2)
169         center_distance = math.sqrt((cx-x)*(cx-x) + (cy-y)*(cy-y))
170         #return 1*(len(own_liberties) - len(opp_liberties)) + 1*(math.sqrt(41) - center_distance) # combination # heuristic 10
171         #return 2*(len(own_liberties) - len(opp_liberties)) + 1*(math.sqrt(41) - center_distance) # combination # heuristic 11
172         return 1*(len(own_liberties) - len(opp_liberties)) + 2*(math.sqrt(41) - center_distance) # combination # heuristic 12
173         #return 1*(len(own_liberties) - len(opp_liberties)) + 3*(math.sqrt(41) - center_distance) # combination # heuristic 13
174         #return 1*(len(own_liberties) - len(opp_liberties)) + 1.5*(math.sqrt(41) - center_distance) # combination # heuristic 14
```

(b)Result

(i) Difference between my moves and opponent's moves (default heuristic)

I set the round = 10. At first, I checked the result of the default heuristic (minimax and alpha-beta pruning) with **depth=3**, which is the same depth as the opponent agent.

My custom agent can win against random agent 100%, and also can win against greedy agent about 80%. However, It can win only 55% against minimax agent. Alpha-beta pruning can only improve the calculation efficiency, so it's understandable that the result is almost even against the minimax agent.

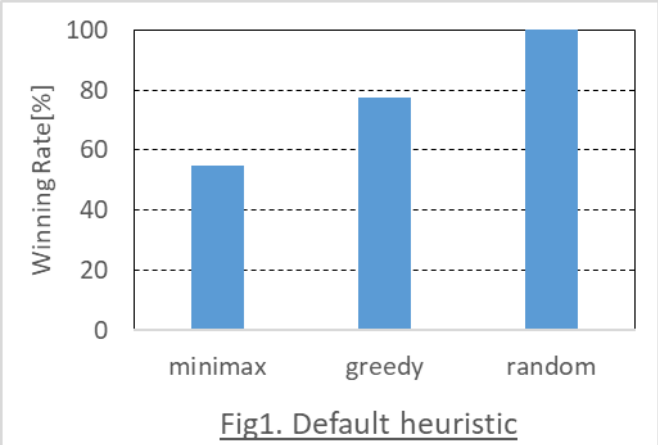

Fig1. Default heuristic

With the default heuristic, I also checked how depth change works.

I fix the opponent as minimax agent here.

I can win more with the depth=4, but the result will saturate after depth=5.

I will take more calculation time with deeper depth, so **I choose depth=4** as the best depth in this project for the following calculations.

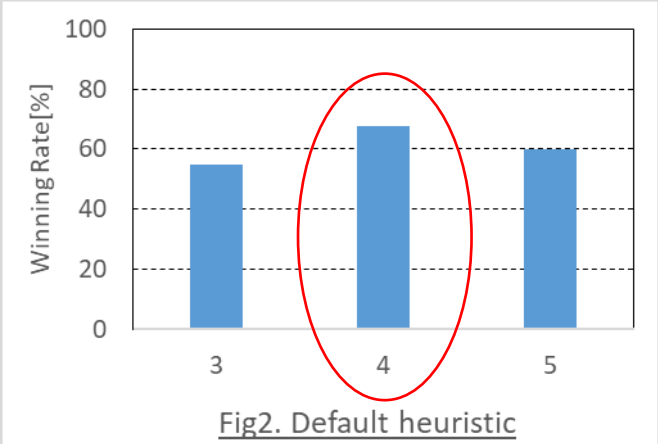

Fig2. Default heuristic

I changed the weight balance between my moves and opponent's moves and the result is shown below. X axis means the weight balance ratio. For example, the ratio of the heuristic "len(own_liberties) - 2 * len(opp_liberties)" is 2. It means that the bigger ratio, the more aggressive agent.

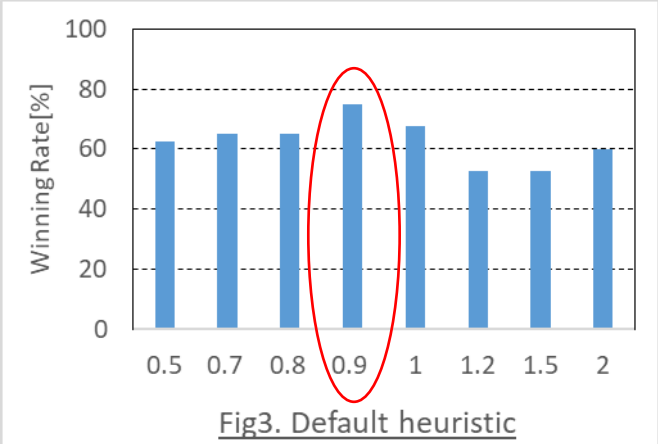

Fig3. Default heuristic

As the result, the ratio nearly equal 0.9: "1.1 * len(own_liberties) - len(opp_liberties)", a little less aggressive agent, has the **best result: 75%.**

(ii) How close from the center (original#1 heuristic)

Only using my original#1 heuristic: center-close doesn't work so well as below.

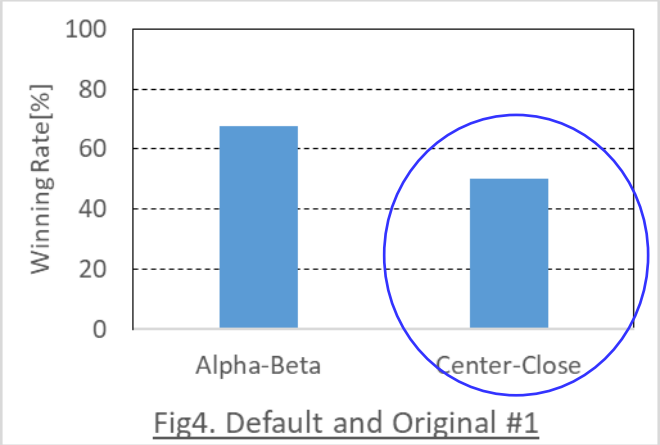

Fig4. Default and Original #1

(iii) Combination of default heuristic and original heuristic (original#2 heuristic)

I changed the weight balance between default heuristic (my moves and opponent's moves) and original#1 heuristic.

The result is shown below. X axis means the weight balance ratio. For example, the ratio of the heuristic "1*(len(own_liberties) - len(opp_liberties)) + 2*(math.sqrt(41) - center_distance)" is 2. It means that the bigger ratio, the more center-close agent.

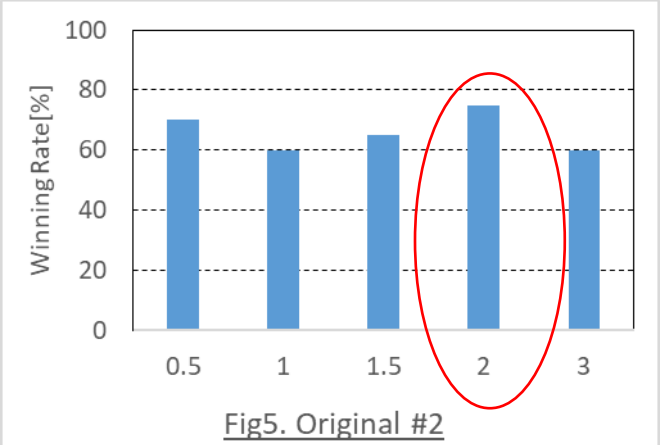

Fig5. Original #2

As the result, the ratio equal 2 : "1*(len(own_liberties) - len(opp_liberties)) + 2*(math.sqrt(41) - center_distance)" has the **best result: 75%.**

(c)Answer to the rubric questions

The answers of the Option1: Advanced Heuristic are show below.

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

  I incorporated the combination of the following 2 features in my heuristic.

  (i)Subtraction of my moves and opponent's moves

  (ii)How close to the center

  (i) is the default heuristic, and it's intuitively understandable to maximize the winning probability.

  (ii) is my original heuristic, and it's intuitively better not to go near to the board edges because the movement of Knight would be restricted.

  I thought it's better to combine these 2 heuristics and give a good balance weight by trying some couples. As a result, I got a good answer.

- 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?

  I changed the depth from 3 to 5 with the final setting.

  Same as Fig.2 above, the winning rate will be saturated after the depth=5.

  It's much slower than depth=4, so I think **depth=4 is the best** in this search problem.

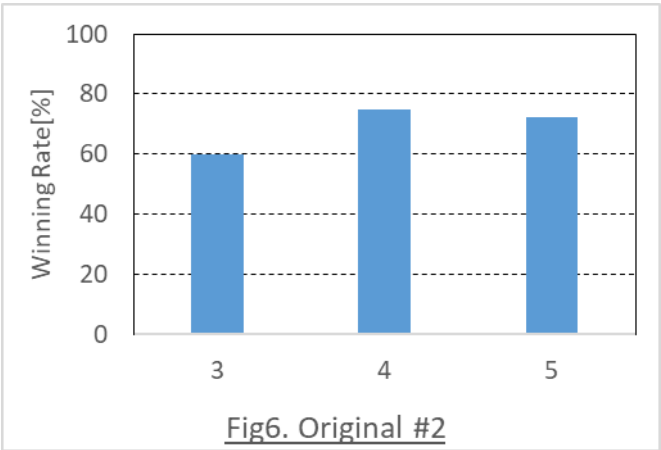

Fig6. Original #2