Build a Game-Playing Agent
Heuristic Analysis

Abhi Ojha

March 15, 2017

1 Heuristic evaluation functions

I have implemented the following three heuristic evaluation functions in this project:

- **runaway()**: This heuristic function (implemented in lines 20 through 58 of the code in file *game_agent.py*) maximizes the distance between the player and the opponent. Player thus runs away from the opponent. We first get the location vectors of the player and the opponent using *game.get_player_location*(). runaway() returns the absolute value of difference between the sum of location vectors. Thus, higher scores have larger absolute differences.
- **runtowards()**: This heuristic function (implemented in lines 58 through 99 of the code in file *game_agent.py*) minimizes the distance between the player and the opponent. Player thus runs towards from the opponent. We first get the location vectors of the player and the opponent using *game.get_player_location*(). runtowards() returns the negative of absolute value of difference between the sum of location vectors. Thus, higher scores have smaller absolute differences.
- **custom_score()**: This function is implemented in lines 102 through 151 of the code in file *game_agent.py*. The idea is to calculate the number of moves available to the 2 players at each game state. We also want to penalize penalize the corner moves for max player and reward corner moves for min player, as it is observed that its not good to be stuck in corner towards the end game. The penalties are elevated as we proceed towards the end game by changing the board factor from 1 to 4.

All the three heuristic functions have following input parameters and return value:

- Input parameters:

  game: *isolation.Board*

  An instance of isolation board encoding the current state of the game (e.g., player location and blocked cells).

  player: *object*

  A player instance in the current game (i.e., an object corresponding to one of the player objects *game.__player_1__* or *game.__player_2__*).

- Return value: *float*
  The heuristic value of the current game state to the specified player.

On analysis it is found out that *custom_score*() gives the best performance compared to *run_towards*() and $run_away()$.

2 Performance Analysis

Table 1: ID_Improved vs Other agents

# Match	Player	Opponent	Win %
1	ID_Improved	Random	70
2	ID_Improved	MM_Null	90
3	ID_Improved	MM_Open	65
4	ID_Improved	MM_Improved	55
5	ID_Improved	AB_Null	80
6	ID_Improved	AB_Open	65
7	ID_Improved	AB_Improved	75

Table 2: Student vs Other agents

# Match	Player	Opponent	Win %
1	Student	Random	75
2	Student	MM_Null	65
3	Student	MM_Open	75
4	Student	MM_Improved	65
5	Student	AB_Null	65
6	Student	AB_Open	60
7	Student	AB_Improved	100

In addition to the provided evaluation functions *null_score*(), *open_move_score*() and *improved_score*() in the file *game_agent.py*. I have implemented three more heuristic functions as described in the section above.

- **runaway()**: This is a defensive strategy. The value returned is the Manhattan distance between the player and opponent.
- **runtowards()**: This is an aggressive strategy in which the player runs towards the opponent and tries to push him in the corner.
- **custom_score()**: This is the best strategy that I came up with. Its not aggressive or defensive, and is good in the sense that it penalizes the bad moves for player and rewards all the good moves. Performance analysis of this strategy against the other agents is shown in the Table II.

From the results of Table I and Table II, it can be seen that the $custom_score()$ (displayed as Student) is a good evaluation function as it has high chances of winning. This can be verified by running the code in file "tournament.py". Compared to the functions *runaway*() and *runtowards*(), I found *custom_score* to have better winning percentage.

Following are the advantages of choosing *custom_score*():

- It beats all the other evaluation functions in head to head competitions.
- Apart from the quantitative evidence provided in Table II, we can analyze the qualitative information provided by this function. The functions null or Random do not evaluate the state of the board at all. This is not good as we need to know the state of the board relative to opponents state to make a good move. *runaway*() and *runtowards*() do this by moving away and towards the opponent by evaluating the state of the board, and thus have more information in there score. *custom_score*() does a really good job in identifying whether the player is at the corner of board or towards the center. It penalizes all the bad moves and rewards the good moves. It also penalizes heavily if you move to the corner towards end game. Thus, *custom_score*() takes into account a lot of information before making the next move.
- All the moves performed by $custom_score()$ are within the time limit. Thus, we don't incur any loss due to the lack of time. This is also another advantage as it shows that the evaluation function is simple enough and can do the required calculations in real time.