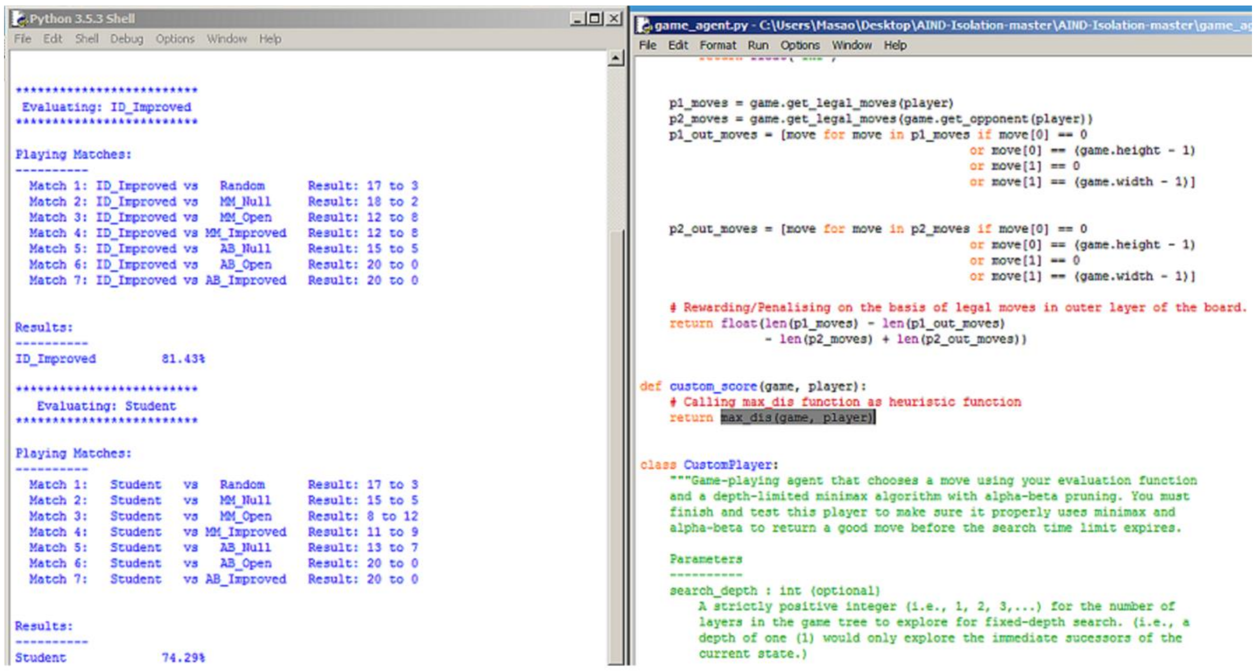
HEURISTIC FUNCTION ANALYSIS

max_dis() function:

I created this function in order to maximize the distance between the players, by returning the absolute difference between the sum of individual's location vector.



Python 3.5.3 Shell
File Edit Shell Debug Options Window Help
game_agent.py - C:\Users\Masao\Desktop\AIND-Isolation-master\AIND-Isolation-master\game_ag
File Edit Format Run Options Window Help

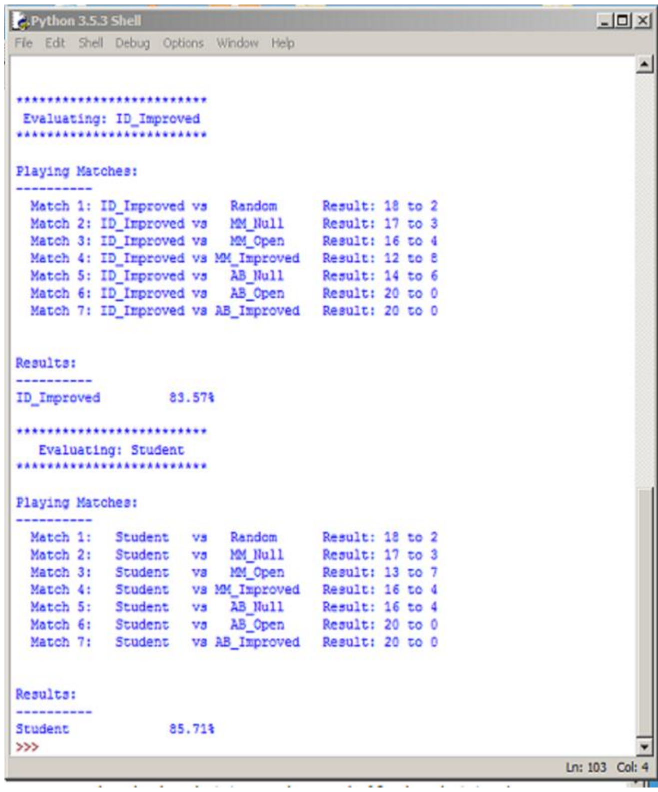
```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 17 to 3
  Match 2: ID_Improved vs   MM_Null     Result: 18 to 2
  Match 3: ID_Improved vs   MM_Open     Result: 12 to 8
  Match 4: ID_Improved vs MM_Improved   Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null     Result: 15 to 5
  Match 6: ID_Improved vs   AB_Open     Result: 20 to 0
  Match 7: ID_Improved vs AB_Improved   Result: 20 to 0


Results:
----------
ID_Improved         81.43%

*************************
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 17 to 3
  Match 2:   Student   vs   MM_Null     Result: 15 to 5
  Match 3:   Student   vs   MM_Open     Result: 8 to 12
  Match 4:   Student   vs MM_Improved   Result: 11 to 9
  Match 5:   Student   vs   AB_Null     Result: 13 to 7
  Match 6:   Student   vs   AB_Open     Result: 20 to 0
  Match 7:   Student   vs AB_Improved   Result: 20 to 0


Results:
----------
Student             74.29%
```

```python
    p1_moves = game.get_legal_moves(player)
    p2_moves = game.get_legal_moves(game.get_opponent(player))
    p1_out_moves = [move for move in p1_moves if move[0] == 0
                                              or move[0] == (game.height - 1)
                                              or move[1] == 0
                                              or move[1] == (game.width - 1)]


    p2_out_moves = [move for move in p2_moves if move[0] == 0
                                              or move[0] == (game.height - 1)
                                              or move[1] == 0
                                              or move[1] == (game.width - 1)]

    # Rewarding/Penalising on the basis of legal moves in outer layer of the board.
    return float(len(p1_moves) - len(p1_out_moves)
                 - len(p2_moves) + len(p2_out_moves))


def custom_score(game, player):
    # Calling max_dis function as heuristic function
    return max_dis(game, player)


class CustomPlayer:
    """Game-playing agent that chooses a move using your evaluation function
    and a depth-limited minimax algorithm with alpha-beta pruning. You must
    finish and test this player to make sure it properly uses minimax and
    alpha-beta to return a good move before the search time limit expires.

    Parameters
    ----------
    search_depth : int (optional)
        A strictly positive integer (i.e., 1, 2, 3,...) for the number of
        layers in the game tree to explore for fixed-depth search. (i.e., a
        depth of one (1) would only explore the immediate sucessors of the
        current state.)
```

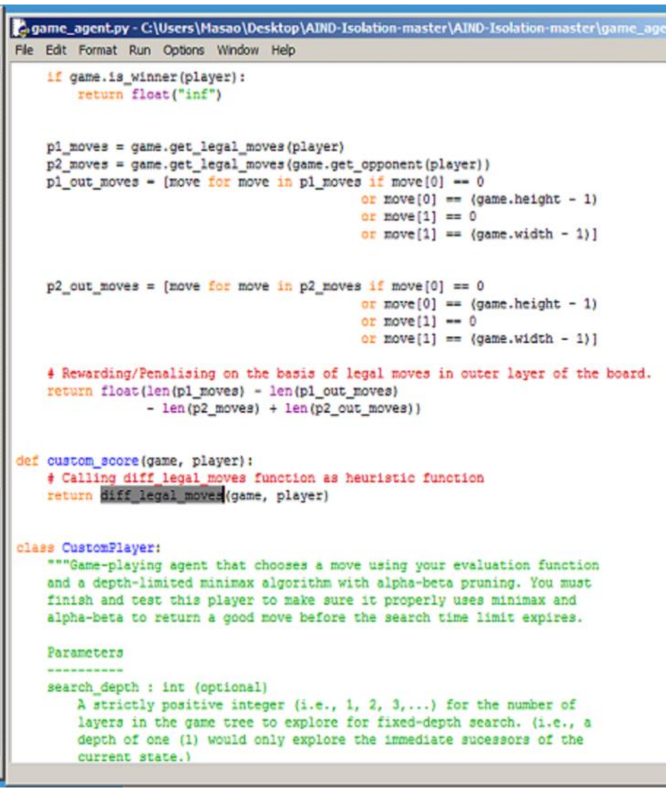
diff_legal_moves() function:

Created this function in order to return the difference in the number of moves available to the two players.

This gives an estimate to compare which player has better placed itself on the board depending upon the possible number of moves at each moment.



```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 18 to 2
  Match 2: ID_Improved vs   MM_Null     Result: 17 to 3
  Match 3: ID_Improved vs   MM_Open     Result: 16 to 4
  Match 4: ID_Improved vs MM_Improved   Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null     Result: 14 to 6
  Match 6: ID_Improved vs   AB_Open     Result: 20 to 0
  Match 7: ID_Improved vs AB_Improved   Result: 20 to 0


Results:
----------
ID_Improved         83.57%

*************************
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 18 to 2
  Match 2:   Student   vs   MM_Null     Result: 17 to 3
  Match 3:   Student   vs   MM_Open     Result: 13 to 7
  Match 4:   Student   vs MM_Improved   Result: 16 to 4
  Match 5:   Student   vs   AB_Null     Result: 16 to 4
  Match 6:   Student   vs   AB_Open     Result: 20 to 0
  Match 7:   Student   vs AB_Improved   Result: 20 to 0


Results:
----------
Student             85.71%
>>>
```


game_agent.py - C:\Users\Masao\Desktop\AIND-Isolation-master\AIND-Isolation-master\game_age
File Edit Format Run Options Window Help

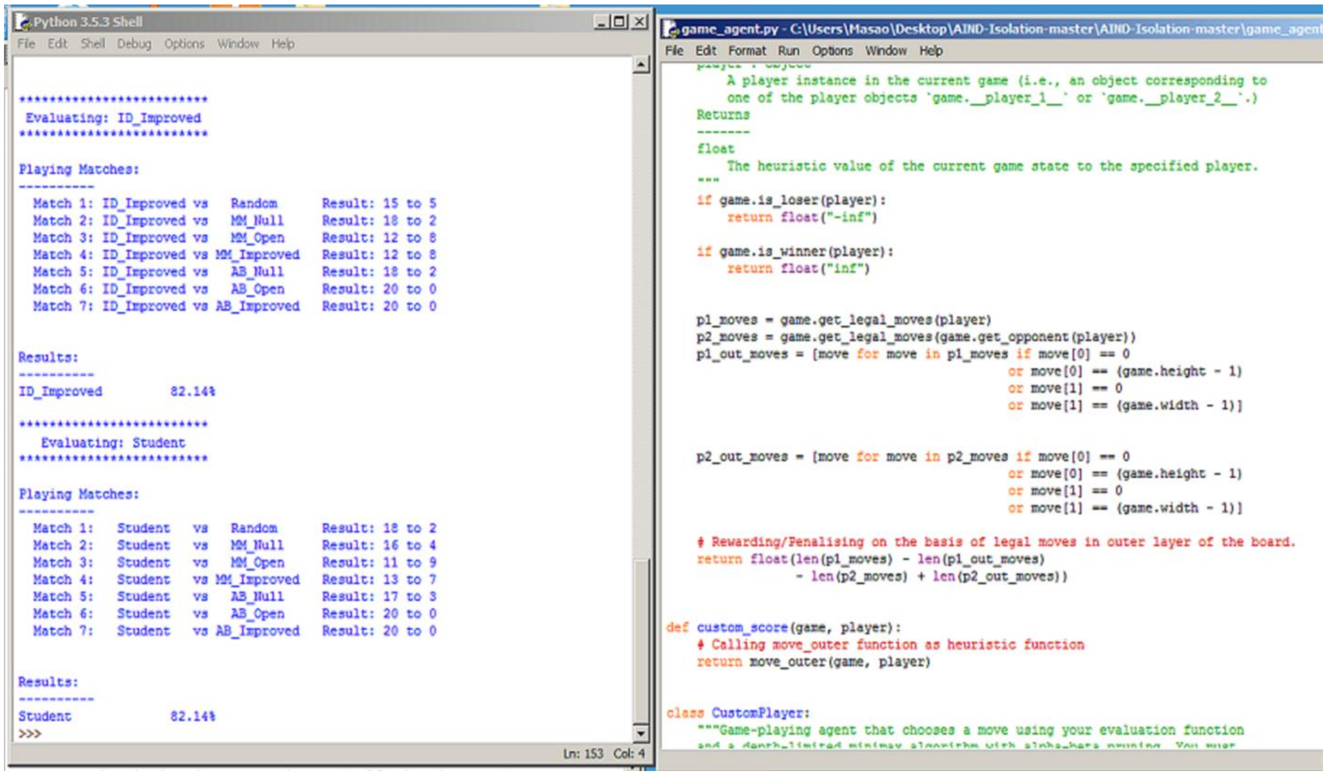
```python
    if game.is_winner(player):
        return float("inf")


    p1_moves = game.get_legal_moves(player)
    p2_moves = game.get_legal_moves(game.get_opponent(player))
    p1_out_moves = [move for move in p1_moves if move[0] == 0
                                                or move[0] == (game.height - 1)
                                                or move[1] == 0
                                                or move[1] == (game.width - 1)]


    p2_out_moves = [move for move in p2_moves if move[0] == 0
                                                or move[0] == (game.height - 1)
                                                or move[1] == 0
                                                or move[1] == (game.width - 1)]

    # Rewarding/Penalising on the basis of legal moves in outer layer of the board.
    return float(len(p1_moves) - len(p1_out_moves)
                 - len(p2_moves) + len(p2_out_moves))


def custom_score(game, player):
    # Calling diff_legal_moves function as heuristic function
    return diff_legal_moves(game, player)


class CustomPlayer:
    """Game-playing agent that chooses a move using your evaluation function
    and a depth-limited minimax algorithm with alpha-beta pruning. You must
    finish and test this player to make sure it properly uses minimax and
    alpha-beta to return a good move before the search time limit expires.

    Parameters
    ----------
    search_depth : int (optional)
        A strictly positive integer (i.e., 1, 2, 3,...) for the number of
        layers in the game tree to explore for fixed-depth search. (i.e., a
        depth of one (1) would only explore the immediate sucessors of the
        current state.)
```

move_outer() function:-

This function returns the difference in the number of moves available to the two players while penalizing the moves for the maximizing player against the wall and rewarding the moves for the opponent player against the wall.

The idea certainly came from previous function but Deep Blue Research paper gave it a broader understanding. As I used it to reward credit score when player has less number of legal places around the outer most layer of the board, while penalizing the opponent for the same situation.

Certainly, this heuristic function performs better than others because of Credit/Penalize mechanism like Deep Blue. This takes into account for good possible position on the board, at the same time considers bad possible positions for both the players.



Python 3.5.3 Shell

```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 15 to 5
  Match 2: ID_Improved vs   MM_Null     Result: 18 to 2
  Match 3: ID_Improved vs   MM_Open     Result: 12 to 8
  Match 4: ID_Improved vs MM_Improved   Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null     Result: 18 to 2
  Match 6: ID_Improved vs   AB_Open     Result: 20 to 0
  Match 7: ID_Improved vs AB_Improved   Result: 20 to 0


Results:
----------
ID_Improved         82.14%

*************************
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 18 to 2
  Match 2:   Student   vs   MM_Null     Result: 16 to 4
  Match 3:   Student   vs   MM_Open     Result: 11 to 9
  Match 4:   Student   vs MM_Improved   Result: 13 to 7
  Match 5:   Student   vs   AB_Null     Result: 17 to 3
  Match 6:   Student   vs   AB_Open     Result: 20 to 0
  Match 7:   Student   vs AB_Improved   Result: 20 to 0


Results:
----------
Student             82.14%
>>>
```

game_agent.py - C:\Users\Masao\Desktop\AIND-Isolation-master\AIND-Isolation-master\game_agent

```python
        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__`.)
    Returns
    -------
    float
        The heuristic value of the current game state to the specified player.
    """
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")


    p1_moves = game.get_legal_moves(player)
    p2_moves = game.get_legal_moves(game.get_opponent(player))
    p1_out_moves = [move for move in p1_moves if move[0] == 0
                                              or move[0] == (game.height - 1)
                                              or move[1] == 0
                                              or move[1] == (game.width - 1)]


    p2_out_moves = [move for move in p2_moves if move[0] == 0
                                              or move[0] == (game.height - 1)
                                              or move[1] == 0
                                              or move[1] == (game.width - 1)]

    # Rewarding/Penalising on the basis of legal moves in outer layer of the board.
    return float(len(p1_moves) - len(p1_out_moves)
                 - len(p2_moves) + len(p2_out_moves))


def custom_score(game, player):
    # Calling move_outer function as heuristic function
    return move_outer(game, player)


class CustomPlayer:
    """Game-playing agent that chooses a move using your evaluation function
```

Q. Why I selected the move_outer heuristic function over others?

A.

1) Strategy to score on the basis of legal moves left by each player outperform strategy of player's location vector to score.

2) Strategy to penalize our score for bad legal moves(boundary positions) helped to improve heuristic fns.

3)Strategy to reward our score for bad legal moves by the opponent helped to improve heuristic fns.

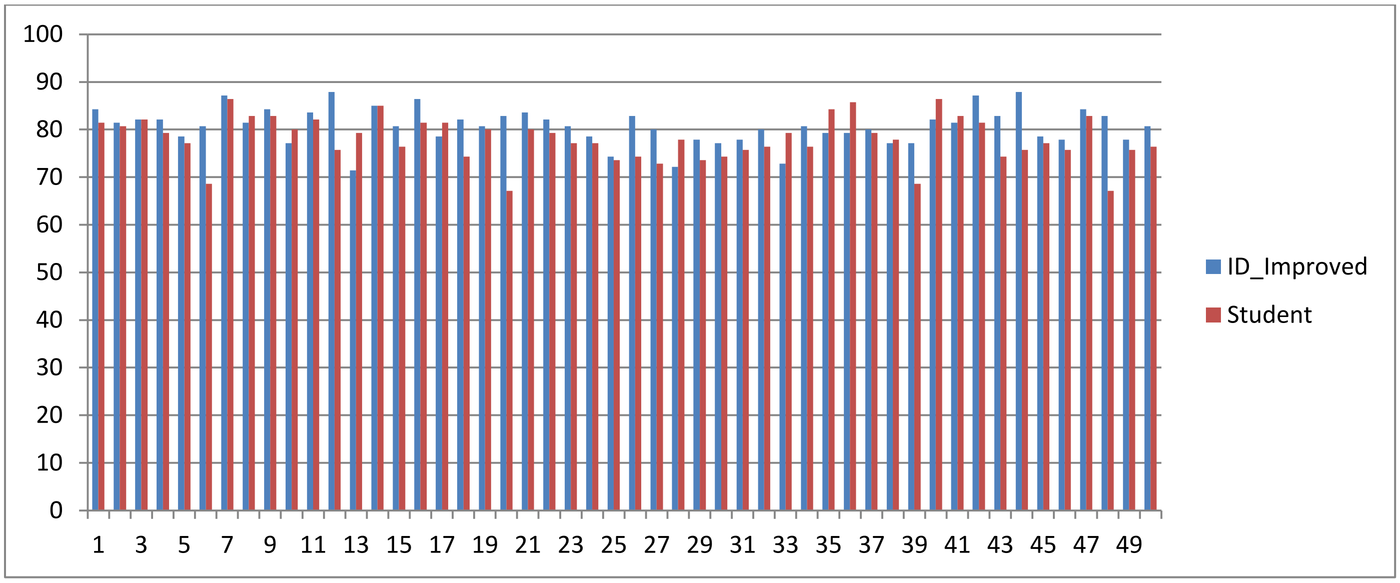


Chart- Tournament Analysis for 50 matches