Defensive and attacking tactics

First intuition in designing evaluation function for deterministic game is making an agent that uses defensive tactics. In this game defensive tactic will be choosing a move that maximizes number of your open moves, so that you have always somewhere to move. Such “defensive” evaluation function is already implemented - it is “open_score_move” in “sample_players.py”.

Next intuition is designing an agent that uses attacking tactic - its evaluation function will minimize opponent’s open moves. After testing it - the results were similar to defensive tactics (open moves) but slightly lower.

After running tests on defensive tactics (number of player’s open moves), attacking tactics (number of opponent’s open moves) and their combined version (“Improved” heuristic function from lecture) the results are 71.29, 70.57, 74.29 percents of win rate accordingly.

Deeper “improved” evaluation function

So, the improved score is the best. Let’s try to improve it more and evaluate not on just number of open moves, but how good these open moves are. So our state of game will be evaluated by summing the scores of player’s currently available moves. Goodness of moves we evaluate the same way as original “improved” score - by counting the number of open moves the player will have if we choose that move and subtracting number of opponent’s moves. Code looks like this:

```
score = 0
for move in game.get_legal_moves(player):
   score += len(game.forecast_move(move).get_legal_moves(player))
   score -=
len(game.forecast_move(move).get_legal_moves(game.forecast_move(move).get_opponent(player)))
return float(score)
```

The result is worse than standard ID_Improved heuristic. The reason should be that this evaluation function is heavier to process, so the iterative deepening algorithm stops at lower depths compared to players using less complex heuristic function.

Centering technique

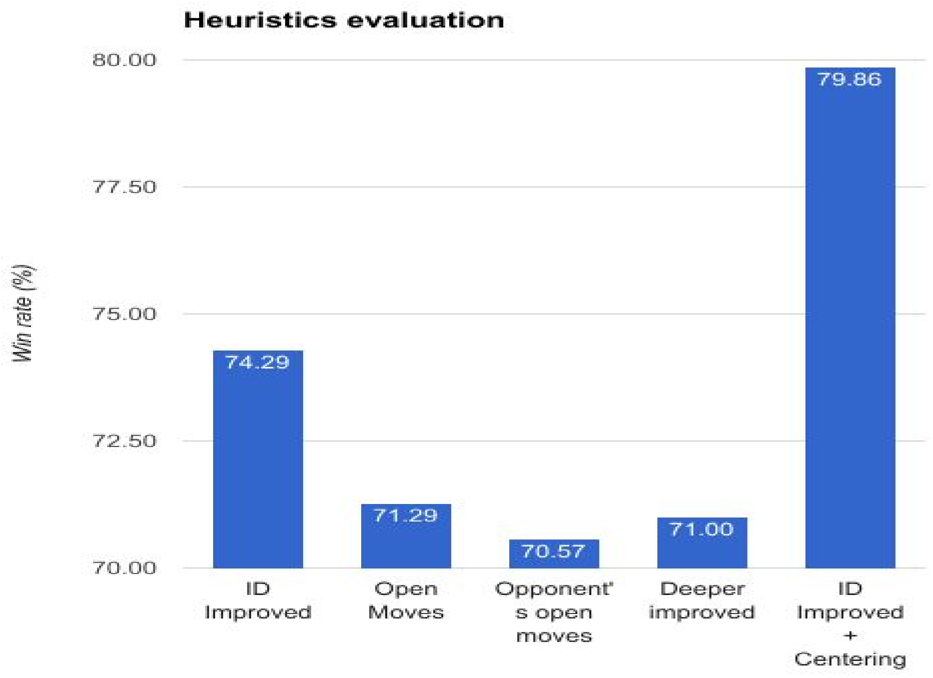
Another idea of evaluation function came from the fact that the player better not be on the edge of the board - because it limits player’s mobility. So I added condition that if player is closer to the center of the board - the better the evaluation function result. Algorithm is implemented as follows:

```
# standard open_moves_score
score = len(game.get_legal_moves(player))
# Lean towards center of the board
center_x = game.width/2
center_y = game.height/2
row, col = game.get_player_location(player)
score -= abs(row-center_x) * 0.1
score -= abs(col-center_y) * 0.1
return float(score)
```

Centering technique is not important in choosing the right move, compared to standard improved score, so its results are lowered 10 times. Experimental results show that centering technique improves “improved” score.

Results

The final results against standard players in tournament.py are:*



	Number of wins, out of 100 games played

	vs Random	vs MM_Null	vs MM_Open	vs MM_Improved	vs AB_Null	vs AB_Open	vs AB_Improved	**Overall**
ID Improved	92	88	71	67	79	63	60	**74.29**
Open Moves	92	79	68	65	76	64	55	**71.29**
Opponent's open moves	89	84	68	54	77	67	55	**70.57**
Deeper improved	93	77	70	66	75	54	62	**71.00**
ID Improved + Centering	94	85	84	75	81	73	67	**79.86**

In “ID_Improved” vs “ID_Improved + Centering” the results are 54% win for “ID_Improved + Centering”.

*All custom players used minimax search with alpha beta pruning and iterative deepening.