Artificial Intelligence Nanodegree Alexandru Korotcov
Project: Build a Game-Playing Agent March 14th, 2017

Heuristics analysis for the Isolation game agent

Isolation is a deterministic, two-player game of perfect information in which the players alternate turns moving a single piece from one cell to another on a board. Whenever either player occupies a cell, that cell becomes blocked for the remainder of the game. The first player with no remaining legal moves loses, and the opponent is declared the winner.

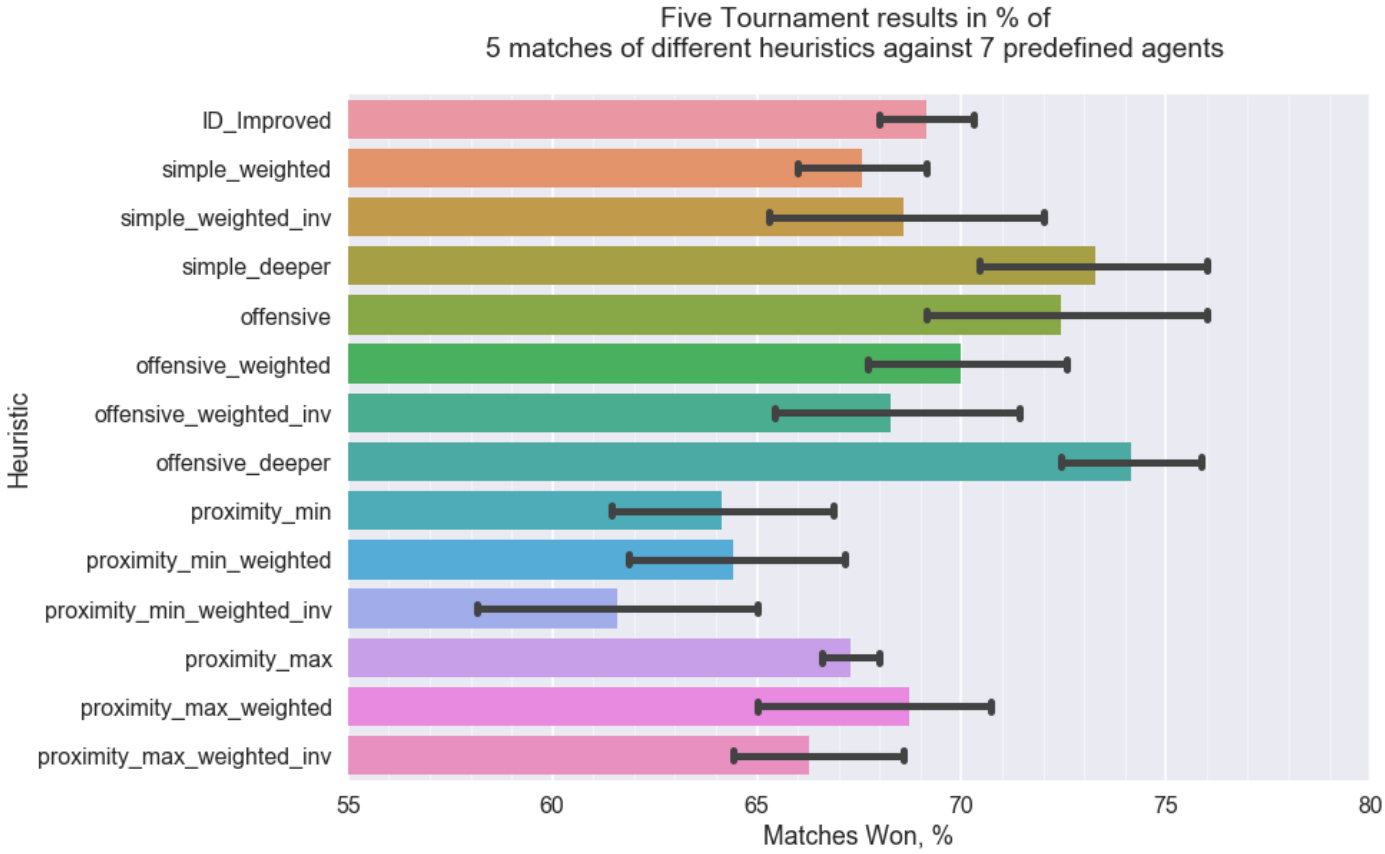
This project uses a version of Isolation where each agent is restricted to L-shaped movements (like a knight in chess) on a rectangular grid (like a chess or checkerboard). Additionally, agents have a fixed time limit each turn to search for the best move and respond. If the time limit expires during a player's turn, that player forfeits the match, and the opponent wins.

The *tournament_new.py* script have been used to evaluate the effectiveness of the programmed custom heuristics. The script measured relative performance of all programmed custom heuristics in a round-robin tournament against several other pre-defined agents. The script runs five tournaments for each heuristic of custom agent. The custom agent used time-limited Iterative Deepening and the following heuristics:

- ***simple_weighted*** evaluates a score equal to the difference in the number of moves available to the two players weighted to the number of empty spaces:
  (own_moves - opponent moves) / (num empty spaces + 1)
- ***simple_weighted_inv*** evaluates a score equal to the difference in the number of moves available to the two players weighted to the inverse number of empty spaces:
  (own_moves - opponent moves) * (num empty spaces + 1)
- ***simple_deeper*** evaluates a score equal to the difference in the number of sum of all moves available to the two players at one level deeper:
  (sum of own_moves at next depth) - (sum of opponent moves at next depth)
- ***offensive*** evaluates a score equal to the difference in the number of moves available to active player and double of available moves to the opponent player:
  (own_moves - 2 * opponent moves)
- ***offensive_weighted*** evaluates a score equal to the difference in the number of moves available to active player and double of available moves to the opponent player weighted to the number of empty spaces:
  (own_moves - 2 * opponent moves) / (num empty spaces + 1.)
- ***offensive_weighted_inv*** evaluates a score equal to the difference in the number of moves available to active player and double of available moves to the opponent player weighted to the inverse number of empty spaces:
  (own_moves - 2 * opponent moves) * (num empty spaces + 1.)
- *offensive_deeper* evaluates a score equal to the difference in the number of sum of all moves available to active player at one level deeper and double of available moves to the opponent player at one level deeper:
  (sum of own_moves at next depth) - (2 * sum of opponent moves at next depth)
- ***proximity_min*** evaluates a score equal to the inverse of the Euclidian distance between players:
  1. / sqrt((x2 - x1)^2c+ (y2 - y1)^2)

- ***proximity_min_weighted*** evaluates a score equal to the inverse of the Euclidian distance weighted by empty spaces:
  1.0 / ((num empty spaces + 1.) * sqrt((x2 - x1)^2 + (y2 - y1)^2))
- ***proximity_min_weighted_inv*** evaluates a score equal to the inverse of the Euclidian distance weighted by inverse number of empty spaces:
  (num empty spaces + 1.) / sqrt((x2 - x1)^2 + (y2 - y1)^2)
- ***proximity_max*** evaluates a score equal to the Euclidian distance:
  sqrt((x2 - x1)^2 + (y2 - y1)^2)
- ***proximity_max_weighted*** evaluates a core equal to the Euclidian distance weighted by of empty spaces:
  sqrt((x2 - x1)^2 + (y2 - y1)^2) / (num empty spaces + 1.)
- ***proximity_max_weighted_inv*** evaluates a Euclidian distance weighted by inverse number of empty spaces:
  sqrt((x2 - x1)^2 + (y2 - y1)^2) * (num empty spaces + 1.)

The results of the five tournaments (see *game_agent_testANDtournaments_results.ipynb*) are showed in the figure below.



Summary: The **most effective** heuristic was ***offensive_depper***. In addition to effectiveness, this heuristic **performs more consistently** than the other heuristics, thus in **average winning *74.1%*** of matches with only 2.2% of standard deviation (***ID_improved*** won ***69.1%*** *of matches*). The main concept of this heuristic is that it **evaluates one step forward (one level deeper)** of the total available moves for both players with **applying idea of having significantly more moves** for this custom agent rather than for the opponent **helping to avoid moving toward losing branches** of the game tree.