Ryland Atkins
Back2Back

Data Structures

The data structures I used were

- A *b2bPiece* class to represent each piece.
  - 2D Numpy arrays within this to represent piece and make rot90 easier to use.
- 2, 2D arrays to represent each side of the board (2 for the front, 2 for the back, I now realize using this was excessive).
  - One Numpy array to hold the depth of each location on the board (0, 1 or 2. Zero being empty, one being full, and 2 being full on the opposite side only).
  - One 2D array of "pieces." A 2D, one unit array for each location on the board--[[0]] or [[1]]. I used 2D here to make implementation smoother in conjunction with the actual game pieces.

Heuristics

Considering that the fastest way to reach a solution is to place each piece on the board with zero mis-steps, so that 11 moves are made, an admissible heuristic would not overestimate this. Since our search depth is 11, we have an upper bound of 11 moves, or 11-number of pieces placed, on our heuristic. But because we cannot get to the goal state faster than 11 moves (i.e. we cannot place two pieces in one move) we have a lower bound of 11 moves, or 11-number of pieces placed. So our heuristic must match or beat this. Since we technically cannot beat it, our heuristic is 11-# pieces already placed.