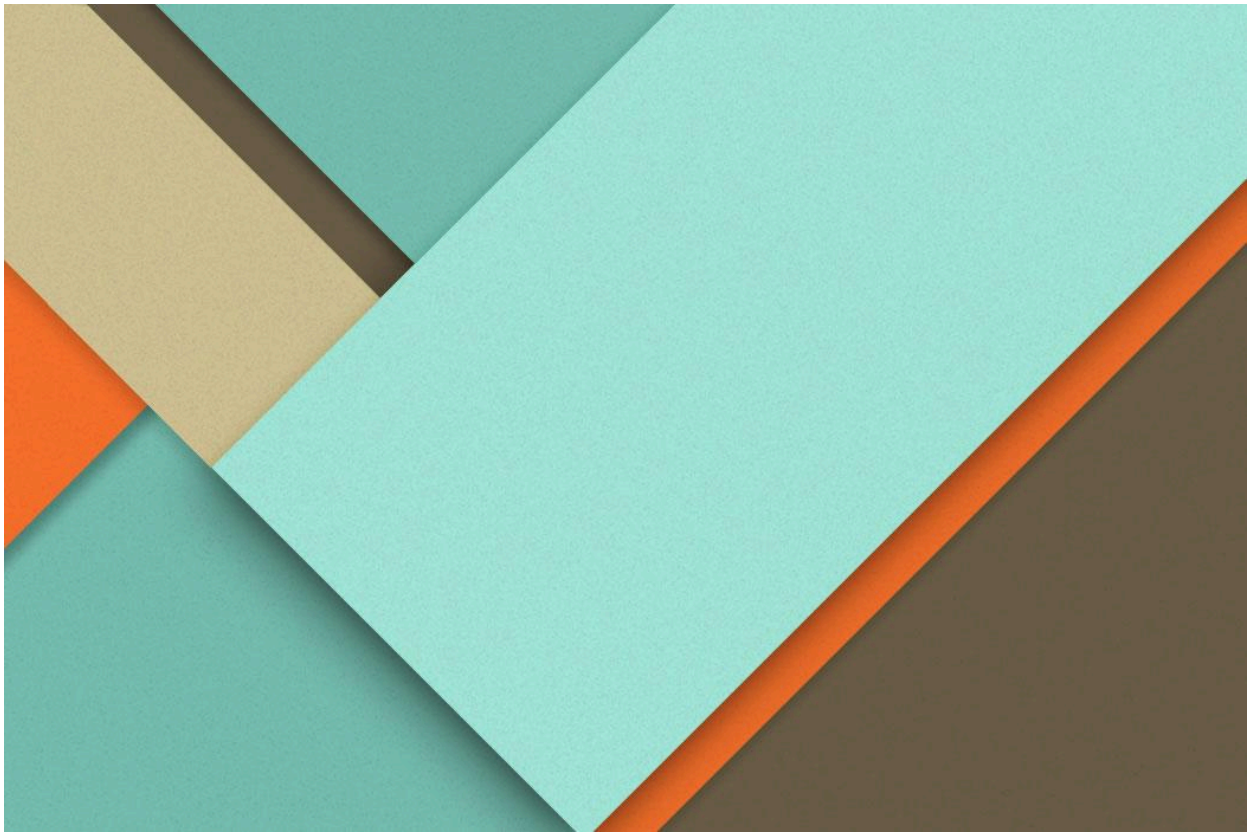

PROJECT 3: BUILD AN ADVERSARIAL GAME PLAYING AGENT

July 2024

TUAN-ANH VO

Custom Heuristics Analysis:

Question 1: What features of the game does your heuristic incorporate, and why do you think those features matter in evaluating states during search?

I picked the experiment of developing the custom heuristic. To evaluate the performance and effectiveness of this custom heuristic, I compare with a baseline heuristic with very simple, naive method: using the *#my_moves - #opponent_moves* heuristic from lecture .

Additionally, I integrated minimax search with alpha-beta pruning and iterative deepening to optimize the search

Here is the detailed explanation of the custom heuristics

- The baseline heuristics: is explained as instruction as the difference between 'our moves left' and 'opponent moves left'.

```
def baseline(self, state):
    """ baseline function given in project introduction using the #my_moves - #opponent_moves
    heuristics."""
    my_liberties = len(state.liberties(state.locs[self.player_id]))
    opponent_liberties = len(state.liberties(state.locs[1 - self.player_id]))
    return my_liberties - opponent_liberties
```

-Central Heuristic:

```
def central(self, state):
    """Evaluate the state based on proximity to the center and edge effects."""

    # Get player positions
    pos_1 = state.locs[self.player_id]
    pos_2 = state.locs[1 - self.player_id]

    # Convert linear position to 2D coordinates (assuming an 11x9 board)
    x1, y1 = (pos_1%(11+2), pos_1//(9+2))  # Coordinates of player 1
    x2, y2 = (pos_2%(11+2), pos_2//(9+2))  # Coordinates of player 2

    # Center of the board
    center_x, center_y = [(11-1)/2, (9-1)/2]

    # Calculate distance to the center
    distance1 = ((x1 - center_x) ** 2 + (y1 - center_y) ** 2)
    distance2 = ((x2 - center_x) ** 2 + (y2 - center_y) ** 2)

    # Penalize positions close to the edges
    edge_penalty1 = max(abs(x1 - 0), abs(x1 - 10), abs(y1 - 0), abs(y1 - 8))
    edge_penalty2 = max(abs(x2 - 0), abs(x2 - 10), abs(y2 - 0), abs(y2 - 8))

    # Weighted centrality
    center_value = -(distance1 - 2 * edge_penalty1) + (distance2 - 2 * edge_penalty2)

    return center_value
```

+ The key point of the central heuristic is the sum of straight line distance between current locations of each player and the center-most cell of the board [5, 4]. One of the noticeable strategy in winning chess is trying to locate the piece as close to the center of the board as possible because they have more available cells to explore in various directions, unlike the edges where one side is completely blocked off.
+ Board boundaries and Edge effects then is also important to ensure that the heuristic accounts for the fact that the board is finite and that being too close to the edges or corners might be disadvantageous. So I adjust the heuristic to penalize positions that are close to the board edges more heavily.
+ Depending on the game progression, the “center” of interest might change. For instance, as players move, the central area of tactical importance might shift. Different areas of the board might have different strategic values so we could use a non-uniform grid where distances from different parts of the board are weighted differently, reflecting their relative importance. So I adapt the center coordinates dynamically based on the game state or player positions with something called weighted centrality

Combined Heuristic (final heuristic):

```
def combined(self, state):
    """Combine baseline and central heuristics with weighted coefficients."""
    return 0.5 * self.baseline(state) + 1.5 * self.central(state)
```

+ This heuristic tries to integrate the above-mentioned heuristics, by assigning certain weights to them individually , as baseline heuristic is multiplied by 0.5 and central heuristic is multiplied by 1.5, and adding them up together.
+ The values 0.5 and 1.5 were chosen by initially selecting them as 1 and then gradually changing them randomly to check which values give a higher success rate.

HEURISTIC ANALYSIS (10 games)		
	Baseline Heuristic result (%)	Custom Heuristic result (%)
RANDOM	85	90
GREEDY	55	70
MINIMAX	30	40
SELF	50	65

We can see that the result increase significantly with custom heuristic in all 4 tested algorithms

Question 2: Analyze the search depth your agent achieves using your custom heuristic. Does search speed matter more or less than accuracy to the performance of your heuristic?

Search depth:

For the depth of the agent , the maximum depth achieved was 18, and the minimum being 8.

Time limit

+ By reducing the time limit to 50ms, the agent's success rate was significantly improved as compared to 150ms - 100%, running for Random with fair_matches enabled for 10 games. But it reduce to 55% in case of greedy running with the same condition.
+ When I increasing it to 500ms, the success rate increased to 95% to the first case and 60% for the second case.

  The change in time limit seem change the success rate not in a linear way but depend on each algorithm instead