Project Report: Adversarial Search Agent

Ramy Rashad

In this project, we implement and study an adversarial search agent in an attempt to win the game of Knights Isolation. We explore various advanced heuristic functions using both iterative deepening and alpha-beta pruning.

1 Definition of Heuristics

1.1 Baseline Heuristic

The first **baseline** heuristic function is the number of moves available to the active player minus the number of moves available to the opponent. This is a very simple that is simply a measure of the number of moves available to each player, weighted equally.

$$h_1 = p - o \tag{1}$$

where:

- p = number of moves available to active player
- o = number of moves available to opponent

Note: The baseline heuristic is functionally and strategically equivalent to $m = 0.5$ in the advanced heuristics described below.

1.2 Advanced Heuristics

The **advanced** heuristic functions ($h2$, $h3$, and $h4$) are defined based on a weighting scheme between the number of moves available to each player.

$$h_i = m \cdot p - (1 - m) \cdot o \tag{2}$$

Where the weights, m, are provided in Table 1.

Table 1: Heuristic Strategies and Weights

Label	Strategy	m
h1	Baseline Heuristic	n/a
h2	Offensive Strategy	0.25
h3	Defensive Strategy	0.75
h4	Offensive to Defensive Strategy	varying

In Equation (2), when m=0, the heuristic function is equivalent to the number of moves available to the opponent. When m=1, the heuristic function is equivalent to the number of moves available to the active player. Note that the baseline heuristic is equivalent to m=0.5, however, we do not explicitly define it as such.

The h_2 and h_3 heuristic functions use a fixed weighting that represent an **offensive strategy** and **defensive strategy**, respectively. The last heuristic (h_4) is an **offensive to defensive strategy** provided by dynamic weighting that changes throughout the game based on the ratio of the number of moves played so far (`state.ply_count`) to the board size (`state.board_size()`), as follows.

$$m_4 = \frac{\text{currentmove}}{\text{boardsize}} \tag{3}$$

This dynamic weighting provides a **smooth** means of transitioning from a more offensive strategy (toward the beginning of the game) to a more defensive strategy (toward the end of the game).

Note: The function `state.board_size()` is a new function that was added to return the number of squares on the board, corresponding to the `_SIZE` variable in `isolation.py`.

2 Results

The Knights Isolation game was played in two modes, with and without the fair matches flag, `-f`. Note that when running the matches with the `-f` flag, the game engine will ensure that the starting positions for both players are fair and symmetric, as opposed to randomly assigned. For both mode, the game was played with each of the four heuristic functions and a depth limit of 5, for a total of 20 rounds. The results of the games are presented in Table 2 in the form of a win percentage **against the Minimax Agent**.

Table 2: Results of Knights Isolation Game (Depth Limit=5)

Heuristic Title	Win % w/out `-f`	Win % w/ `-f`
h1	57.5	62.5
h2	55.0	63.8
h3	65.0	67.5
h4	70.0	73.8

The results show that the both the defensive and the adaptive advanced heuristics (h_3 and h_4) generally outperform the baseline heuristic (h_1) in both modes of play. The results also show that the all heuristics perform better with `-f` enabled than without, presumably because the randomness can be occasionally advantageous to an inferior heuristic.

3 Q & A

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

The advanced heuristics (h_2, h_3, and h_4) incorporate different strategies. While h_2 and h_3 are fixed throughout the game, the reason I believe that h_4 is superior is because it provides a smooth transition from an offensive strategy to a defensive strategy. This is important because the game is dynamic and the optimal strategy changes throughout the game. In essence, the dynamic but predictable nature of the h_4 heuristic gives the strategy a sense of time. For example, at the beginning of the game, it seems advantageous to be more offensive and try to limit the opponent's moves. However, toward the end of the game, it is seems more advantageous to be defensive and try to maximize the number of moves available to the custom agent. The dynamic weighting of h_4 provides a smooth transition between these two strategies.

3.2 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?

The search depth that may be achieved using a custom heuristic depends primarily on the average branching factor, the available time limit for each move, and the time required to evaluate the heuristic. In the case of Knights Isolation, the maximum number of branches is 8 and the default time limit is 150 milliseconds. As such, it was assumed that the search depth that could be achieved by the custom agent is equal to or greater than 5. The computational expense of computing all of the heuristics investigated in this project is essentially identical for all intents and purposes, particularly when $h1$ is equivalent to $m = 0.5$. As such, the search speed is not a key differentiator and does not seem to matter as much as the accuracy or strategy of the heuristic (in the studies herein). Furthermore, since the depth limit was restricted to 5 and the players were observed to consistently explore to the full depth limit at each turn, the accuracy of the heuristic seems to be more important than the speed of the search.