Project 4: Build an Adversarial Game Playing Agent
By Neel Gandhi

Introduction

In this project, we experiment with adversarial search techniques by building an agent to play knights Isolation. Unlike the examples in lecture where the players control tokens that move like chess queens, this version of Isolation gives each agent control over a single token that moves in L-shaped movements--like a knight in chess. The project is played on a 9x11 grid.

This report summarizes my experiments with Advanced Heuristics in Isolation. After testing four different heuristics, two of which were proposed in class and two that were variations of those including search depth as an argument, I found that adding search depth as an argument resulted in slightly inferior results than simple aggressive play (Heuristic #2).

Experiments

My experiments with Advanced Heuristics involved implementing a baseline agent and four different heuristics and having them play each other for 100 fair games. A fair game, as I define it here, is a set of two games where players swap initial positions to mitigate any advantages that would be resulting of that positional advantage.

Baseline: MinimaxPlayer with Alpha-Beta Pruning + Iterative Deepening

Before implementing anything novel, I decided to start off by cloning the `MinimaxPlayer` seen in `sample_players.py` and augmenting it with the Alpha-Beta Pruning and Iterative Deepening techniques presented in the Classroom. The `#my_moves - #opponent_moves` heuristic from lecture is maintained, as it was used in the `MinimaxPlayer` agent.

Heuristic #1: Defensive

Just for the sake of testing out different heuristics proposed in class, I defined Heuristic #1 as #my_moves, which is arguably more defensive because it aims to only maximize the number of moves available to the agent.

Heuristic #2: Aggressive

Also to compare heuristics presented in class, I defined Heuristic #2 as #my_moves - 3*#opponent_moves, where the greater coefficient multiplying #opponent_moves would lead to a greater intent to reduce the number of moves available to the opponent.

Heuristic #3: Progressively Aggressive

As a simple variation of the heuristics presented in class, I started thinking that maybe it would make sense for the agent to start like the baseline and become progressively more aggressive. The way I found to encode it was to add search depth as the multiplicative coefficient, defining Heuristic #3 as `#my_moves - depth*#opponent_moves`.

Heuristic #4: Regressively Aggressive

I then thought it would be interesting to see whether the reverse approach - to start playing aggressively and then become more defensive - would lead to different results. To encode that behavior, I defined Heuristic #4 as `#my_moves - MAX(1,4-depth)*#opponent_moves`.

Results

Table 1 summarizes the results seen after 100 fair games were played between agents running each of the proposed heuristics. Since even agents playing against themselves are susceptible to an uneven 50/50 split in their win rates, I took those numbers (in yellow) as my reference for what a 50/50 split would be for Player 1 (rows) when playing against Player 2 (columns). When Player 1 has a higher rate than that reference against Player 2 running a specific heuristic, I colored that cell green. Otherwise, the cell is colored red.

Table 1 - Heuristic win rates after 100 fair games

	Against Baseline	Against Heuristic #1	Against Heuristic #2	Against Heuristic #3	Against Heuristic #4
Baseline	54.5%	66.5%	50.0%	49.0%	53.5%
Heuristic #1	41.0%	53.0%	31.0%	46.5%	37.0%
Heuristic #2	55.5%	59.0%	48.0%	56.0%	53.5%
Heuristic #3	56.0%	64.5%	51.0%	53.5%	54.5%
Heuristic #4	52.0%	63.5%	48.0%	53%	49.5%

Table 2 - Heuristic win/loss count for each heuristic Player 1

	Baseline	Heuristic #1	Heuristic #2	Heuristic #3	Heuristic #4
Wins	1	0	4	3	3
Losses	3	4	0	1	1

Discussion

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

I incorporated the idea of depth into my heuristic, as it encodes how far into the game the agent would be in. I think this feature might matter because it might be worth it to differentiate the opening strategy from the endgame.

I used the idea of multiplying #opponent_moves by a given number as a way to make gameplay more aggressive, as seen in concept “Evaluating Evaluation Functions” of the classroom.

With Heuristic #3 and #4, I wanted to check whether it would be better to start defensive and get more aggressive as the game progressed, or start aggressive and end the game more defensively.

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?

My agent reached a minimum of 6 and a maximum of 20 plies deep. I believe neither speed nor accuracy were definitive in my heuristics’ performance, but that their definition was ill-conditioned. The multitplicative factor, specially in Heuristic #3, was too high - so a higher search speed would actually lead to poorer decision-making. Even still, it did not perform as badly as it could hav, considering that point.