PROJECT REPORT

Date: May 17, 2020
Project: Build an Adversarial Game Playing Agent
School: Udacity
Written By: Hardik Trivedi

KNIGHTS ISOLATION

This project is about experimenting with various adversarial search techniques do build an agent that plays the knights isolation game. The knights move in L-shaped moves similar to knights in chess. The goal of the game playing agent is to isolate the other player such that the opponent has no more moves left and the agent has at least one move left. There are several algorithms that are already provided, the project requires completion of one of three options in order to complete the project.

The options are as follows:
1. Develop custom/advanced heuristics while using the Alpha-beta search with iterative deepening.
2. Develop an opening book
3. Develop an agent using advanced search techniques (MCTS, principal variation seach, etc.)

CORE CONCEPTS

Isolation: A deterministic, two-player game in which players alternate turns moving a single piece on the game board. The game board, in this case, is a 9 x 11 grid.

Knights isolation: The game consists of two knights that move in L-shaped (2 rows, 1 column OR 2 columns, 1 row) fashion, while each player alternates turns. The game ends when one of two players have no remaining moves.

Search Algorithms: There are three search algorithms given including Random, Greedy and Minimax that the player needs to compare with the Alpha-beta (pruning) search algorithm with iterative deepening using a baseline and custom/advanced heuristics.

Minimax Algorithm: A backtracking algorithm that creates a search tree starting with the root node. The algorithm consists of alternating max_value and min_value function at each level of the tree. Starting with the bottom level of the tree, this search algorithm uses the max_value and min_value function alternatively at each level until it reaches the root to find the best moves for the agent to take at each level.

Alpha-beta Pruning: An optimization technique for the minimax algorithm. Alpha-beta pruning cuts off branches in a game tree that are not needed to be searched because a better move has already been found. It includes two extra parameters alpha and beta with initial values of -infinity and +infinity respectively. The condition for pruning is beta <= alpha. Alpha and beta are best values guaranteed by the max_value and min_value functions at that level or above.

Iterative Deepening: A techniques that starts its search at depth one and continues to move to the next depth of two and so on, until a set time limit is reached. The algorithm saves the best move at each depth and returns the best available move when the time limit is reached. This techniques guarantees that there is always a best move available even if the search is cut off as it runs out of time and was unable to go through the entire tree to find the next best move.

Heuristics: A technique designed to solve problems more quickly compared to classical methods, when classical methods are too slow. OR A technique to find an approximate solution when classical methods fail to find an exact solution. For this project, the baseline heuristic is #my_moves - #opponent_moves and the custom heuristic is offensive-to-defensive. There are other techniques available for custom heuristic but offensive-to-defensive is expected to yield the best results.

FUNCTIONS TO IMPLEMENT

I have chosen option 1 - develop custom/advanced heuristics while using the Alpha-Beta Search with Iterative Deepening. This will require implementing a CustomPlayer class with several methods/functions including:

get_action(): this function is called once per turn and puts the available moves on the queue.

alpha_beta_search(): this function implements the alpha_beta search algorithm and consists of two additional functions **min_value()** and **max_value()** from the original Minimax algorithm.

score(): this function implements the advanced and baseline heuristics.

TABLE OF CUSTOM AGENT WINS

Custom Player Win Rates	Self	Minimax	Greedy	Random
Baseline Heuristics (#my_moves - #opponent_moves)	47.50%	57.50%	97.50%	92.50%
Custom Heuristics (Offensive-to-defensive using depth)	52.50%	80.00%	100.00%	92.50%

Certain things stand out as we look at the results above:
1. Alpha-beta pruning with iterative deepening and custom heuristics increase the win rate for our playing agent even with fair matches enabled.
2. Baseline heuristic performance is not ideal as better heuristics exist.
3. Offensive-to-defensive heuristic, where the agent plays in offensively first and then switching to defensive play gives the best results.

QUESTIONS

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

Answer: A combination of liberties and depth. Depth matters because we are searching the game tree using iterative deepening and searching for best possible moves at each depth combined with the liberties available for the agent is the best custom heuristic available.

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?

Answer: It appears from the results that while the number of actions and hence the depth increases with the implementation of a custom heuristic, the entire tree is still not able to be searched in the time limit given. If we were to increase the time limit, the entire tree might be searched. Speed matters more than accuracy as the game begins and accuracy matters more towards the end of the game. Overall however, speed seems to matter more given that moves have to be selected in a set time limit.

ACTUAL RESULTS

Custom Heuristics (Offensive to Defensive):

```
root@b7b63e1db039:/home/workspace# python run_match.py -f -r 10 -o SELF
Running 20 games:
-+--+-+-+-+--+-+-+-+-+
Running 20 games:
+-+-+-+-+++-+-+-+-+-
Your agent won 52.5% of matches against Custom TestAgent
```

root@b7b63e1db039:/home/workspace# python run_match.py -f -r 10 -o MINIMAX
Running 20 games:
+++++++--++-++++++++-
Running 20 games:
+++-++-++++++-+++-+++
Your agent won 80.0% of matches against Minimax Agent

root@b7b63e1db039:/home/workspace# python run_match.py -f -r 10 -o GREEDY
Running 20 games:
++++++++++++++++++++
Running 20 games:
++++++++++++++++++++
Your agent won 100.0% of matches against Greedy Agent

root@b7b63e1db039:/home/workspace# python run_match.py -f -r 10 -o RANDOM
Running 20 games:
++++++++-++++++++++-+
Running 20 games:
+++-++++++++++++++++
Your agent won 92.5% of matches against Random Agent

Baseline Heuristics (#my_moves - #opp_moves):

root@b7b63e1db039:/home/workspace# python run_match.py -f -r 10 -o SELF
Running 20 games:
-+-+-+-+-+-+-+-+-+-+
Running 20 games:
+-+-+-+---+-+-+-+-+-
Your agent won 47.5% of matches against Custom TestAgent

root@b7b63e1db039:/home/workspace# python run_match.py -f -r 10 -o MINIMAX
Running 20 games:
+++----+++++-++++-+-
Running 20 games:
+-++-+-+--++-----+++
Your agent won 57.5% of matches against Minimax Agent

root@b7b63e1db039:/home/workspace# python run_match.py -f -r 10 -o GREEDY
Running 20 games:
++++++++++++++++++++
Running 20 games:
+-++++++++++++++++++
Your agent won 97.5% of matches against Greedy Agent

root@b7b63e1db039:/home/workspace# python run_match.py -f -r 10 -o RANDOM
Running 20 games:
++++-+++++++++-+++++++
Running 20 games:
++++++++++++-++++++
Your agent won 92.5% of matches against Random Agent