Adversarial Search

Michal Janczyk

June 6, 2020

1 Introduction

1.1 Abstract

In the science of the search algorithm is an algorithm that will be able to solve the search problem, specifically, more accurately the information stored in a particular data structure or determined in the search domain of the problem domain, with discrete or continuous values. Specific applications of search algorithms can be adversarial search. Adversarial search, which explains how to proceed when other agents are our opponents, i.e. their goals are opposed.

2 Research Design

In the science of the search algorithm is an algorithm that will be able to solve the search problem, specifically, more accurately the information stored in a particular data structure or determined in the search domain of the problem domain, with discrete or continuous values. Specific applications of search algorithms can be adversarial search. Adversarial search, which explains how to proceed when other agents are our opponents, i.e. their goals are opposed.

2.1 What are the games in the concept of AI?

Games are a form of multi-agent environment.

- What is the purpose and how do other agents behave? Do they influence our decisions?
- A cooperative and competing version of the multiagent environment

In games, we deal with an "unpredictable" opponent.

2.1.1 Game types (according to game theory)

- Two-player game (two agents)
- An antagonistic (zero-sum) game (values of the utility functions of agents at the end of the game are equal or opposite)
- A game with a finite number of strategies (deterministic environment)
- Non-competition game
- Full information game (fully observable environment)

2.1.2 Games versus search problems

Games - without the opponent.

- Solution: find the target state
- Heuristics make it possible to find a suboptimal and sometimes optimal solution
- Assessment function: estimating the cost of moving from the starting point to the solution by the node
- Examples: road planning, scheduling

Games - against the opponent

- The solution is the strategy (it determines the traffic for each opponent's possible response)
- Time limits enforce an approximate solution. In a few simple games, you can give the optimal solution
- Evaluation function: estimating the 'quality' of items in the game.
- Examples: chess, checkers, backgammon

2.1.3 Assumptions of Two players games: MAX and MIN

MAX makes the move first and then moves alternately with the opponent (one player's move is a half move) until the end of the game. The winner receives a prize for a lost penalty. The game as a problem of searching state spaces:

- Initial state: eg starting position on the board and determining which player will make the first move

- Function generating the next state: creates a list of pairs (movement, state) defining the allowed movements and the states received by this movement
- Termination condition: which determines when the game is over. The states in which the game ends are called terminal states
- Utility (payout, purpose) function: gives the numerical value of the terminal node, eg win (+1), lose (-1), draw (0) (tic-tac-toe)

The initial state and legal movement are defined by the game tree. MAX will use the tree and search it to determine the next move. The optimal strategy for MAX assumes the infallibility of the opponent MIN. This means that we assume that the opponent is reasonable and will not make moves that are unfavorable to him. The example will demonstrate how to find the optimal strategy, although for harder games than noughts and crosses it will not be possible to calculate. Assumption: Both opponents play optimally. For a given game tree, the optimal strategy will be determined by checking the minimax value (MINIMAX-VALUE (n)) of each node. Idea: choose a move to the position with the best minimax value that is equal to the best achievable payout function in the game with the optimal opponent. Hence MAX will select states with maximum values and MIN with minimum values.

3 Analysis

This project aimed to experiment with adversarial search techniques by building an agent to play knights Isolation.

All calculations were done on my private workstation. All the tables below present 4 problems of the game run that ware conducted during the search process for each of the search functions. The Adversarial Search class presented in the project - CustomAgent implements what the search technique presented in the report requirements. The heuristic must meet several requirements, the selected heuristics (it cannot be one of the lectures heuristics and cannot be only a combination of the number of freedoms available for each agent). Should have at least 4 layers. As an example, I chose to implement an advanced technique not included in the lecture by Principal Variation Search. All the tables below present 4 problems of the game run that ware conducted during the search process for each of the search functions, according to the equation. Run 40 games (10 rounds = 20 games x2 for fair matches = 40 games) against. As the experiment plan, I chose the third option.

Variant	Alpha-Betha Pruning	Principal Variation Search
SELF	45.00%	52.50%
GREEDY	15.00%	22.50%
RANDOM	37.50%	70.00%
MINIMAX	8.00%	10.00%

Table 1: Option 3

4 Conclusion

- How much performance difference does your agent show compared to the baseline?

AI is an experimental filed of science, where the ability to do more experiments directly correlates with progress so to get the most accurate results you would have to increase your research sample. In my experiment, I managed to achieve suboptimal growth relative to the base. Three variants of the experiment (self, greedy, minimax) indicate an increase in performance not exceeding 10%. One of the experiments (random) is characterized by a large increase in performance that can testify to its erroneous introduction. This test should be repeated to get the optimal result.

- Why do you think the technique you chose was more (or less) effective than the baseline?

Is game dependent heuristics. Although strategies that are independent of the game are preferred, in practice, information specific to the type of game is often used to increase the effectiveness of heuristics. For example, in checkers, you may prefer moves that capture more opponent pieces, etc. Principal variation search consists of searching branches outside principal variation with the window [localalfa, localalfa + 1]. It is assumed that movements outside principal variation are unlikely to be good (assuming perfect ordering).