Reinforcement Learning 2021
Practical Assignment 1: Search

1 Introduction

A friend of you has challenged you to participate in a virtual tournament of the two-player board game called Hex, created by John Nash. Figure 1 shows an example of a hex board with a size of 11×11. In this game, two players, red and blue, attempt to build a "bridge" across the board, either horizontally (blue), or vertically (red). The first player to succeeds in doing so, wins.

Naturally, the first player to move has a slight advantage. For this, the Pie rule was invented, which allows the second player to pick a color after the first move has been made. For simplicity, we do *not* use this rule.

Your friend finds an 11×11 board a bit too challenging, and proposed to do a tournament on boards of smaller sizes. In all spontaneity, you agreed to this challenge. However, you know that you are probably not going to win, as your friend is much more experienced. Desperate as you are, you suddenly recall that you are a good AI programmer! Shortly after, you decide to write a program *in Python 3* that plays the game for you!

After brainstorming for a bit, you came up with a plan.

2 Alpha-Beta

The first algorithm that you want to implement is *alpha-beta* as it is optimal. You split the implementation into four parts:

- Read chapter 4 from the book Learning to Play [2]
- Write a move-generator function
- Implement a dummy evaluation function in the form of a random number generator. Alternatively, you could use a deterministic one that always returns the same number. With this dummy evaluation function, your program is a blind searcher.

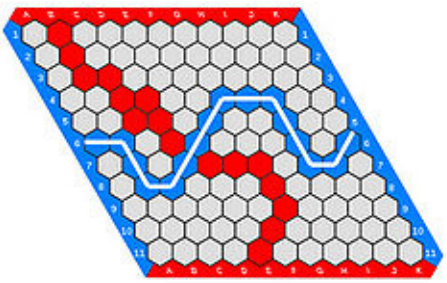

Figure 1: An empty 11×11 hex board, source: [5]

- Build a text-based interface which allows you to interactively play games of Hex with variable board sizes.

Since you no longer have the algorithm in your mind too sharply, you asked some tips from an expert. He told you the following: start with small board sizes, such as 2 or 3. Extra functions that you may want to consider using are getMoveList, makeMove, unMakeMove, various checks for empty tiles to encapsulate the raw board data structure management, and of course functions to help debugging, such as board printers. For the board data structure you could consider numpy arrays or Python dictionaries. Color to move can be incorporated into the board, or carried as a separate parameter in the function calls. Print how many nodes are searched. Print cutoffs.

After you have implemented all this, you decide to implement a heuristic evaluation function that uses Dijkstra's shortest path algorithm to determine how far the bridge is from the edge. For this, you can use [3].

You make sure to run many experiments with your program to evaluate its strength. For this, you use TrueSkill [4]. More specifically, you let the following three agents play against each other:

- Search depth 3 with random evaluation
- Search depth 3 with Dijkstra evaluation
- Search depth 4 with Dijkstra evaluation

You determine the Elo rating of these three agents. Also, you investigate how many games should be played for the ratings to stabilize. You ask yourself whether you can compute this number. Lastly, you measure the speed of your program, and decide on the board size that you will use.

Iterative deepening and transposition tables

Iterative deepening and transposition tables are perhaps the most important enhancements of alpha-beta programs. They have three advantages:

1. Alpha-beta becomes an anytime algorithm. It can be stopped at anytime, and an approximation of the optimal policy and the value function are available based on the last search depth that has been completed.
2. They allow us to come up with good move orderings.
3. The state space to be search effectively shrinks, since states that have been searched before will not be searched again.

You review iterative deepening and transposition tables in Learning to Play [2] in Sections 4.4.2 and 4.4.3. Your plan is to implement both enhancements, using a time bound for iterative deepening, and storing the best moves in the transposition table. Listing 4.6 in the book does not show how best move and depth should be used. In your solution, you incorporate the search depth, otherwise shallow search results will spoil the search effort. The implementation of the TT should work correctly, but do not spend too much effort on an efficient implementation.

You perform a new experiment to see how much the ELO rating changes with these enhancements.

3 Monte-Carlo Tree Search (MCTS)

The second algorithm that you will implement, is called *Monte-Carlo tree search*, or MCTS. Its advantage is good performance in applications where it is hard to find a heuristic evaluation function. It has propelled the field reinforcement learning significantly, and artificial intelligence in general, by showing that great successes of heuristic planning (such as in Chess) can also be accomplished without the hard work of building domain specific heuristic functions. Since you are up for a challenge, you do *not* use MCTS packages. After all, everyone can call functions.

You come up with a few ideas to go about the implementation:

- You refresh your mind by reading Chapter 5 of the Learning to Play book [2], and, optionally [1]. You form an idea of default hyperparameter settings for the number of simulations N and the UCT parameter C_p
- To test your program, you feed it test positions, play against it manually, and let it play against the alpha-beta agent (which you previously implemented). You wonder how you can setup robust experiments with the non-deterministic, random play-outs.
- You compare the ELO rating of MCTS [4] and the iterative deepening agent with transposition tables (IDTT). Again, you think about how many games should be played to obtain statistically stable results.
- You experiment with different hyperparameter settings for N and C_p. Sufficient care is required in the design of these experiments, as they can easily become too computationally expensive. Choose the board size wisely. No results mean no points for this part. "The best is the enemy of the good" – Voltaire, a.k.a. "Premature optimization is the root of all evil" – Donald Knuth.

4 Submission

Of course, you nicely document everything that you do, such that your work can be used by other people too. Your final submission consists of:

- Source code with instructions (e.g., README) that allows us to **easily** rerun your experiments.
- A self-contained pdf report of *at least* 6 to 8 pages if not even more with figures etc. (more pages is allowed!) This report should contain an explanation of the techniques, your experimental design, results (ELO plots, performance statistics, other measurements,...), and overall conclusions, in which you briefly summarize the goal of your experiments, what you have done, and what you have observed/learned.

If you have any questions about this assignment, please visit our lab sessions on Friday where we can help you out. In case you cannot make it, you can post questions about the contents of the course on the Brightspace discussion forums, where other students can also read and reply to your questions. Personal questions (not about the content of the course!) can be sent to our email address: rl@liacs.leidenuniv.nl.

The deadline for this assignment is the **26th of February 2021 at 23:59 CET**. For each full 24 hours late, one full point will be deducted (e.g., if your work is graded with a 7, but you are two days too late, you get a 5).

Good luck beating that friend of yours! :-)

References

[1] *Analytics Vidhya Introduction to Monte Carlo Tree Search: The Game-Changing Algorithm behind DeepMind's AlphaGo.* `https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/`.

[2] Aske Plaat. *Learning to play—reinforcement learning and games.* 2020.

[3] *Towards data science Hex - Creating Intelligent Adversaries (part 2: Heuristics & Dijkstra's algorithm).* `https://towardsdatascience.com/hex-creating-intelligent-adversaries-part-2-heuristics-dijkstras-algorithm-597e4dcacf93`.

[4] *TrueSkill The video game rating system.* `https://trueskill.org/`.

[5] *WikipediA Hex (board game).* `https://en.wikipedia.org/wiki/Hex_(board_game)`.