Reinforcement Learning 2020: Assignment 1
Heuristic Planning

Aske Plaat
rl@liacs.leidenuniv.nl

February 28, 2020

1 Introduction

The objective of this assignment is to build intuition on a typical reinforcement learning problem, and on the heuristic planning algorithms that can be be used to address this problem. You will build a small game playing program for the game of Hex. The program will have all the basic elements of a full game playing program, such as a search function and a heuristic evaluation function.

A secondary objective is to perform program evaluation to determine which program is stronger.

This is an introductory assignment to familiarize ourselves with the world of reinforcement learning and strategy games. We use a simple game, Hex, a simple procedure, alpha-beta, and provide a framework of functions allowing you to focus on a single topic in each part of the assignment (search, eval, skill rating, and enhancements). By staying close to the provided functions, you should be able to do this assignment efficiently. If you get stuck, ask help.

Hex is a game designed by John Nash. It has simple rules and can be played on a small board, allowing easy implementation and experimentation with limited computational resources. Hex is a snake building game. Blue tries to connect from the left of the board to the right, and Red from top to bottom. Hex is actually quite fun to play, and many smartphone apps exist.

Heuristic planning is the technology that has been central to AI for many years, and has been responsible for many of the breakthroughs that shape our modern society, in Chess, route planning, search engines, routing, social networks, and many other fields.

For this assignment, in addition to your own creativity and inventiveness, you may find the following resources useful:

- Chapter 4 of Learning to Play.
- the Chess program Sunfish. Sunfish is a full Chess program written in 111 lines of Python by Thomas Ahle. The source code can be found on

github[1].

- The *HexBoard* class provided in *hex_skeleton.py*.
- TrueSkill. TrueSkill is a library that can be used to calculate the skill rating of a game playing program. Originally developed by the Microsoft Research Team, Heungsub Lee wrote a python implementation that can be found here[2].
- The Wikipedia page has useful information on Hex, the page of Ryan Hayward's group at the University of Alberta has useful information, and if you want the full story you should read Ryan Hayward and Bjarne Toft's book.

Write your program in Python. Use the Hex implementation provided. Most research and development of current deep reinforcement learning algorithms is performed in Python. Becoming familiar with Python is a learning goal of this course.

When you are stuck, your first resource should be the book for this course. If you find that confusing or it does not answer your question, please write an email to `info@learningtoplay.net`. Consult the resources of the assignment, ask questions at the classes, or the question hour.

2 No Pie rule

The first player in Hex has an advantage. For that reason, the pie rule has been invented. The pie rule, or swap rule, allows the second player to choose which color to play, to negate the first-mover-advantage.

For this assignment, we do not use the pie rule, to ease implementation, and to allow us to focus on search and evaluation algorithms.

3 Search - 3 points

Write a program for the provided Hex implementation. The program consists of

- a search function: alpha-beta
- a move generator
- a dummy evaluation function: a random number generator, or, if you want a deterministic one for debugging: one that always returns the same number. Note that for now three program will not be able to determine if it has won or lost. It is just a blind searcher.

[1] `https://github.com/thomasahle/sunfish`
[2] `https://trueskill.org/`

- a text-based user interface to allow interactive playing of a game

Test your code.

Evaluate and play against the program to check if it works.

Hints (these are hints to get you started. If you have other ideas to come to correct, efficient, well commented and beautiful code, then please do so): start with small board sizes, such as 2 or 3. Extra functions that you may want to consider implementing are getMoveList, makeMove, unMakeMove, various checks for empty to encapsulate the raw board data structure management, and of course functions to help debugging (such as the *HexBoard.print()* function). Print how many nodes are searched. Print cutoffs.

4 Eval - 2 points

Add a heuristic evaluation function to the engine. Use Dijkstra's shortest path algorithm to determine how far from the edge a snake is. This blog[3] shows how to do it.

Test your code by using test positions of which you know the outcome. Test the program by playing against it manually.

5 Experiment - 2 points

Build a script to evaluate the strength of your program. Use TrueSkill to determine the skill rating (see resources). Another rating library is also allowed. Play the following three programs against each other:

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

Determine the skill rating of each program. How many games should be played for the rating to stabilize statistically? Can you calculate this number?

How fast is your program, what board size will you use?

6 Iterative Deepening and Transposition Tables - 2 points

Perhaps the most important enhancements of alpha-beta programs are Iterative Deepening and Transposition Tables. They have three advantages:

[3]https://towardsdatascience.com/hex-creating-intelligent-adversaries-part-2-heuristics-dijkstras-algorithm-597e4dcacf93

- 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.
- They provide for improved move ordering.
- The state space to be search effectively shrinks, since states that have been searched before will not be searched again.

Review ID and TT in Learning to Play in Sections 4.4.2 and 4.4.3. Implement both enhancements, using a time bound for iterative deepening, and store the best moves in the transposition table. Listing 4.6 does not show how best move and depth should be used. In your solution, do 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.

Perform a new experiment to see how much the skill rating changes with these enhancements.

7 Report

Your submission should consist of a report in pdf of at most 10 pages with skill graphs of the experiments and a textual report on your findings, explaining the outcomes of the experiments, and the implementation process. Your code should be uploaded to blackboard and ready to run and free of syntax errors. Your report should contain:

1. One overall implementation report: general approach, issues encountered
2. For each section: Source Code, ready to interpret and run, on blackboard. Short user documentation in report
3. For each section: Supporting scripts for running, testing, and performance evaluation on blackboard. Short user documentation in report.
    - This means that we want to be able to know how to reproduce the results of your experiments
    - e.g., if you have one file per experiment: For task 6 run "skillrating_ab_idtt.py" and find the result in figure.png / data.json
    - e.g., If you like writing DRY code and everything is in one file: For task 6 run "a1.py –task skillrating –idtt enabled" and find the result in figure.png / data.json
4. For each section: Test results (approach, positions used, outcome)
5. For each section: Report of performance evaluation
6. One overall conclusion, lessons learned, observations

The report must be handed in on **28 February 2020 before 23:59**. For each full 24 hours late, one full point will be deducted.