Yale University
CPSC 474/574 - Computational Intelligence for Games
Fall 2020

Assignment 4 - Monte Carlo Tree Search

Objectives

- to implement Monte Carlo Tree Search

Introduction

See the Wikipedia article on Kalah.

Assignment

Create a Python 3 module called `mcts` (so this must be in a file called `mcts.py`) that implements a function called `mcts_strategy` that takes a number of iterations and returns a function that takes a position and returns the move suggested by running MCTS for that number of iterations starting with that position.

The positions passed to the search functions will be instances of `Kalah.Position` from the `kalah` module. Python does not have a mechanism to prevent access to members, but your search will not be graded if it accesses members whose names begin with an underscore (the Python convention for indicating members not intended for use by client code). The functions you are allowed to use are

- `legal_moves` to return a list of legal moves from a position, where the moves are given as the indices of the pits to sow from, numbered counterclockwise starting with 0 for P1's leftmost pit;
- `next_player` to determine which player is to make the next move;
- `result` to return the position that results from making a move;
- `game_over` to determine if a position is terminal;

- `winner` to determine which player won at terminal positions (return values are 1, 0, or `-1` to indicate a P1 win, draw, or P2 win respectively); and
- `is_initial` to determine if a position is the starting position.

You can see the `minimax` function in the `minimax` module for examples of using these functions and read their documentation in the `kalah` module.

Additional Requirements

- You may not use global variables or class variables (or any other construct that results in a variable whose lifetime is the entire execution of the program) in any of your modules. (Global constants are allowed and encouraged where appropriate.)
- Tests will be executed with `pypy3`, so write your code for Python 3.5 and only use modules are available for `pypy3` on the Zoo (this excludes `numpy`).
- Observe the given iteration limit. Each playout is considered an iteration (so if you expand and perform a playout from all children at once in a single iteration of your loop, that still counts as multiple iterations).

Files

In `/c/cs474/hw4/Required` are three Python 3 modules:

- `kalah.py` that defines classes that implement the rules of Kalah
- `minimax.py` that defines functions that implement minimax
- `test_mcts.py` the implements the test drivers

Examples

```
[jrg94@chameleon Kalah]$ pypy3 test_mcts.py -game 100 4 0.1 1000
0.53
[jrg94@chameleon Kalah]$ pypy3 test_mcts.py -game 100 4 0.1 10000
0.705
```

Testing

Because of the stochastic nature of the tests, they must be repeated many times in order to determine your agent's performance with enough accuracy for grading. For example, that the first test in the examples above takes about 2½ minutes and the second test takes the over half an hour (this is the cost of using a search technique that doesn't require any domain knowledge).

Because the tests take so long, we leave it to students to decide how much time they need to allow the tests to run in order to be confident in the results. The public test script therefore only tests that your submission will execute correctly with the required files.

You should aim to have your MCTS agent beat the depth-4 minimax agent 50% of the time when given 1000 iterations and 70% of the time with 10000 iterations, assuming both agents make random moves 10% of the time.

Submissions

Submit just your `mcts.py` module along with any other supporting modules and your log. There is no need to submit a makefile or executable; the test scripts create an executable called `MCTS` that is used to run the test drivers using `pypy3`.

The time limit for the single public test is set to allow 1000 iterations per move. The private test cases may use more iterations and the time limit will be set proportionally higher. Faster implementations may be rewarded with more iterations.

Submissions must implement Monte Carlo Tree Search, so submissions that use a different search will not be graded even if they meet the performance requirements automatically checked by the grading scripts.