README.md

MP2

team: banana.ai

Setup

1. Ensure you have the `pypy` runtime installed, running on Python 3.7, as well as the `pip` package manager.
2. Install python required packages, run from root folder: `pip install -r requirements.txt`

How to run

Basic Game

- run `pypy lineEmUp.py`

As an executable w/ flags

- set executable bit on `lineEmUp.py`
- run `./lineEmUp.py --help` to see command line options

Analysis

Demo slides can be found at https://jamboard.google.com/d/1OQW7jbKz-L5Dqh8K9px6Xm59RnkEuQ4P347_CUdMqCY/edit?usp=sharing

Heuristics

The 2 separate heuristics functions developed were done so that h1 would outperform h2. All functions below take advantage of a helper method which returns an array of arrays, each representing the elements in a line: horizontally, vertically, and diagonally.

A simple caching system implemented allows for end game calculations and heuristic evaluations to be sped up.

h1

The h1 function focuses on an aggressive approach to solving the problem. We favor winning chains and will award an increasing score to a player that build them. ex line: `0.xxx.0` will evaluate to -1000 for X and only +20 for O. In order to support any length of s we use consecutive tokens as an exponent, so +/- 10 ** count.

We try to avoid blocs and impose a negative score of 5 if one is encountered.

And finally we prefer moves with a lower depth as they will lead to a better result. We add the depth directly to the score as its weight is relatively small.

Our final score is then reduced back into a range of -1 to 1 in order to support the limits set out by `is_end()`

h2

The h2 function employs a simple approach which focuses on a more defensive strategy overall. While analyzing the board and computing scores, more weight is placed on boards that showed more "blocking" moves made.

Essentially the algorithm simply looks at each line (horizontally, vertically, diagonally) and runs through each element one by one, once for each direction. The only thing that is kept track of is the current and previous value. Priority is given to state changes from `x` to `0`, and vice versa.

Since this heuristic only accounts for some basic checking, and does not use the depth or maximizing information, it is easily outperformed by the h1 heuristic.

is_end

`Game.is_end()` can also be considered a heuristic that can only evaluate the endgame states. Instead of duplicating that code in our own heuristics, we chose to depend on it to look up win or tie conditions and let it be the boundary for our scores.