COMSW4701: Artificial Intelligence

Assignment 3

William Falk-Wallace (wgf2104)

The submitted zip folder contains my submission for AI Assignment 3. It has a command line prompt which requests board size, winning chain length, and turn time limit. Those parameters may also be specified as command line arguments. The main class is named `GomoTest`, rather than `player-account`, to fit with Java Specifications (which do not allow hyphens in classnames) and Conventions.

How To Run:

1. make sure `runtests.sh` is executable: `chmod a+x runtests.sh`
2. run `./runtests.sh`

Alternatively,

1. run `make`
2. run `java GomoTest n m s p1 p2`

   Where

   - `n` is an integer corresponding to the board size (n x n)
   - `m` is an integer corresponding to the winning chain length
   - `s` is an integer corresponding to the turn time limit (in seconds)
   - `p1` is an integer corresponding to the player 1 mode (0 - Random; 1 - Human; 2 - α-β)
   - `p1` is an integer corresponding to the player 2 mode (0 - Random; 1 - Human; 2 - α-β)
     (ex. `make; java GomoTest 10 5 30 2 2`)

Or, for a Command Line Prompt,

1. run `make`
2. run `java GomoTest`

Notes:

The α-β Search function is coded in *GTree.java*, which calls a heuristic funtion `getUtility` in *GState.java*.

Heuristic Analysis:

My heuristic evaluation function computes the longest consecutive chain of each player's tokens (ie. longest string of x's or o's) less than the chainlength. The resulting heuristic value is the difference between the two (maximize own chainlength, minimize opponent's). This is effective first, since any chain longer than the winning chainlength is not valuable to the player. It also balances the simultaneous (zero-sum) goals of diminishing an opponents chances of winning while optimizing its own. The algorithm uses a depth limit of 4 during its execution to widen the breadth of its search given time constraints.

Attempts to further limit the set of searched actions following a given state to enable further depth of search diminished accuracy for this heuristic and did not see worthwhile gains. The current design consistently wins against a random player, both when moving first (as x) as well as second (as y).

My algorithm runs in polynomial time. A better Actions function could reduce this significantly in the same way a more directed start position for the evaluation function would: given the time cutoff, a more directed start of the search would result in significantly better performance and an ability to increase depth without losing accuracy.

Test Analysis

Playing against my player 5 times resulted in quick wins by me. My heuristic doesn't effectively measure threat-space fully; as a result, it is easy to set up multiple-win-move states without being blocked by the computer opponent. With more time and depth, my computer was able to block a few of my open-sided attempts to force a win, but still eventually lost.

Pinning my heuristic player against a random player consistently results in the heuristic agent winning. it takes a few moves longer when playing second, but has not lost yet.

When the α-β player plays against another α-β player, the result is generally a win for whichever moves first (as x), but has resulted in a few o-player wins.

The results of several games with Random *v.* α-β and α-β *v.* α-β are attached in the file `log.txt` which is produced from the tests listed in the `runtests.sh` file.