Mike Schmid
Lab 2 Document
3/31/2017

About My Solution
Name: Darwin
Language: Mostly C. Python is used for the interactive shell.
Opening Plies: 8
Later Plies: 8-11

Evaluation

My implementation has a 11x8x6 table containing a value for every combination of a given tile. The resulting evaluation of a board is just the sum of all the tile values. The table values are refined through autonomous play.

Techniques

- Alpha-Beta Pruning
- Iterative Deepening
- Move Order Evaluation – I use a heap and order the move by it's resulting evaluation so moves that result in captures tend to get evaluated first and moves that result in lost pieces are checked later.
- Evolutionary Computing – The results of this aren't really included in this submission since I just finished the process shortly before the due date. But weights are randomly generated and competed against each other. The losers are cut. The winners are spliced for some new sets and a couple new random sets get added in. The overall best winner competes against the previous overall winner and if that results in a win for the new best winner that set gets recorded as the current best.

Strength

This is hard for me to personally judge as I'm pretty bad at the game. My early broken version was already beating me. Now that it's searching 8 plies deep it easily stomps me. One of the current quirks early on, with my hand crafted set of weights is that most early positions evaluate the same so it has a tendency to just play the first move generated which is the king's, which in the long run doesn't seem very ideal. This issue should go away if my program managed to generate a good reliable set of weights.

My program has played itself many times so at the very least it's internally consistent and doesn't crash. Assuming my understanding of Morph is right, I don't think it's generating any illegal moves.

Program Install and Structure

Note

I have developed and tested this application on Linux. It's possible that it may compile on Windows and be usable but this is untested. I've built and tested the application on athena so it represents a good set of minimal requirements. The packaged version was built on athena so it should be possible to run without compilation. If you're testing on another machine make sure to compile it for that machine. I'm using the -march=native flag so the generated binary is using architecture specific optimizations.

Installation

- Unpack the submission
- Navigate to the Minimax folder
- Run build.sh

Basic Usage

If you're not already in the Minimax folder, navigate there. Inside the Minimax folder run this command

```
python play.py 1 baseweights.txt 5
```

This will launch the game with the human player getting the first move and the computer getting 5 seconds to make a move. The parameters are as follows:
1 – The starting player. Change to 2 if you want the computer to go first
baseweights.txt – The file containing the weight for evaluation. There is also a randomweights.txt to test with.
5 – The number of seconds the computer is allowed to use for taking a turn.

If you'd like to compete a set of weights against each other you can run something like this

```
python auto.py baseweights.txt randomweights.txt 5
```

The first weights file will be player 1 and the second will be player 2 with player 1 always going first. The 5, like before, is the number of seconds each computer gets to spend taking a turn.

If you want to test with some new random weights. The generateRandomWeights.py generates a set of random weights and outputs them to stdout. Just run it like

```
python generateRandomWeights.py > newweights.txt
```

Structure

The main portion of my implementation is in the Minimax directory. My implementation of Morph is in many parts: getmove, isgameover, playmove, printboard, validatemove. Each part is non interactive and takes in various arguments and information from stdin to produce something in stdout. This allows a game to be played by piping the output of one command into the input of the next and makes it easy to swap out individual components. It also makes it easier to automate games for the evolutionary part of my program.

To make the experience a bit more user friendly the python files play.py and auto.py string these commands together for the user for a continuous start to end experience.

All the shared c files are in the core directory so compilation can be done in the following fashion:

```
gcc -o playmove playmove.c core/*.c
```

The second portion of my implementation is in the GeneLab folder. This is for testing large amounts of both random and spliced weights sets. To launch it run

```
python evolve.py
```

This program runs endlessly and populates the genes.txt and best.txt after each round. The genes.txt contains the sets of weights at the end of the last round and best.txt contains the winners of each round provided it can also beat the most recent best as well.