Assignment 2 - Connect 4

Author: Tamara Alhajj

Student ID: 100948027

Overview

An informative explanation of the traditional game can be found at: https://en.wikipedia.org/wiki/Connect_Four

This is a python implementation of connect 4 using a CLI to setup and pygame for the GUI. My implementation for this game allows for the conventional game play as well as additional features. These include:

```
- The option to remove your players piece from bottom peg
- Smart AI vs Human
- Smart AI vs Dumb AI (*required*)
  - Dumb AI uses random moves
- Smart AI vs Average AI
  - Average AI uses minimax alpha-beta pruning with depth 1, thus finding
best moves at a glance
- Smart AI vs Smart AI
  - The perfect play.
  - No option for peg removal
```

Play

To play run `python3 connect4-GUI.py` from the working directory. Then simply follow the instructions given by the CLI to start the GUI with your desired features.

If you are playing against the AI, move your mouse to the desired column and press `a` to add a peg there or `r` to remove your bottom peg from the column. If this is an invalid move your command will be ignored and you are free to try again.

If you choose to watch the smart AIs play against each other, please note that peg removal will not be enabled. This is due to unreasonable runtime if both of the AIs were to check best moves and best removals.

AI

This is a two player game with minimax AI using alpha beta pruning. The state space is a 2D numpy matrix, with dimensions 6 by 7. These are the conventional dimensions of a connect 4 game. I setup a random turn start so that any player has the chance to start first.

Although playing first does give a theoretical advantage, the smart AI seems unbeatable by any player, human or otherwise. The only case where I noticed the advantage of starting first was when I tested 2 smart AIs against each other. This ultimately gave favour to the beginning player.

The heuristics I used for my minimizing/maximizing functionality were an offensive scoring, and the other, a defensive scoring. I chose to score highly for moves with consecutive AI pieces, most highly for 4 consecutive red pegs and less so for 3 or 2 pegs. Significantly, in both cases I use the static strategy of scoring the center column highly, as there are more possibilities there. The defensive mechanism plays more timidly, and will occasionally miss a winning move to block their opponent. This is because I heavily penalize moves that allow for the opponent player to have consecutive pegs lined up. The offensive heuristic in contrast would diminish the score of that move only slightly, and heavily reward move that lead to consecutive pegs. As for removing nodes, the smart AI never does so, which I assume is because it never needs to. The other AIs remove valid pegs at random, for some fun game play. For the AI it was too costly to check with removal capabilities at each level, so I have them just handle the *threat* of removal, which works quite well.

The number of nodes searched were consistently in the thousands. This is due to the fact that node expansion is based on the production system and the depth of the alpha-beta search. However, without alpha beta pruning, it searched between 80 thousand and 100000 nodes, depending on the AI mode. So the pruning indeed had a profound effect on the runtime.

Currently, the depth is bound at 5 levels. However, with a lower cut off, say 3, the number of nodes searched were in the hundreds. Moreover, when the Average AI is playing with the Smart AI, they *both* use the alpha-beta search so this can magnify the number of nodes searched to the tens of thousands. Amazingly, yet understandably, when both smart AIs play the nodes searched can still reach 64505 with the pruning.

As for the heuristics, the number of nodes searched seem to be more for defence due to tendency to block rather than win quick, however it is hard to make a direct comparison with so many game play possibilities.

Improvements

Alpha-Beta pruning would work best if the best move was considered first, otherwise it is costly since it must search all trees one by one. Notably, the killer heuristic optimizes the number of nodes pruned by checking the move that caused the beta-cut off at a similar level last time, and will be checked first. A similar heuristic to this is the history heuristic, which saves the best moves to reference using a transposition table (python dictionary) of previous moves. Essentially, the history heuristic is the killer with memorization.

An improvement I plan to implement in the future is peg removal for the smart AIs. I am curious to see whether this would lead to an endless game.

Acknowledgements

Thank you to Keith Galli for his video series of connect 4 graphics using pygame. This aided me greatly in implementing the game play with a pygame GUI.