Connect 4 Game Project

Intelligent Systems Programming 2014

Jakob Melnyk, jmel@itu.dk
Jacob Claudius Grooss, jcgr@itu.dk

March 13, 2014

Our implementation of MiniMax

The MiniMax algorithm is an adversarial search algorithm. The search is performed as a depth-first search. When a terminal state is reached, the utility of the state is calculated and propagates up the search tree. The different branches of the tree are picked according to which player is making the move in the tree. The algorithm assumes that MAX (the player making the decision) will try to maximize his/her utility while MIN (the adversary of MAX) will attempt to minimize the utility of MAX.

In our implementation of MiniMax we have modified the algorithm to cut-off at a certain depth and then evaluate the board state at that depth based on a heuristic called *n-out-of-4*.

Cut-off

As mentioned in the previous section, we use a cut-off test to reduce the amount of time taken for the AI to decide on a move. A higher cut-off depth allows for more correct evaluation of the moves being considered. The expense of the higher cut-off depth is the time it takes for the algorithm to finish evaluation of all reachable board states. Because of the high branching factor (7), the cut-off depth needs to be low in order for the AI to decide within a "reasonable" time frame.

Alpha-beta-pruning could have improved the algorithm by not considering branches that would never influence the final decision. In the best case scenario, implementing alpha-beta-pruning could let us roughly double the cut-off depth[1] while taking the same amount of time to decide on a move.

We did not implement iterative deepening, which would have allowed us to be flexible in terms of cut-off depth. The branching factor is lower further into the game, so an alternative to iterative deepening could be to increase the cut-off depth depending on the number of either open columns or empty coin slots.

Evaluation function

To make the AI play intelligently, it needed a way to evaluate the different moves and compare them to each other. This evaluation is an estimate of how good a move is, based on subsequent moves.

Our evaluation of the board state is determined by the number of n-out-of-4 a player has in a row. When the function encounters a coin belonging to a player, it will check how long the row is. It will first check for 4-in-a-row, then check for *n-1* and keep going until it reaches 1-in-a-row. The first type of row that is encountered takes priority. Thus, a 3-in-a-row is not also a 2-in-a-row and a 1-in-a-row.

The function will count horizontal rows, vertical rows and diagonal rows and add their values together. Depending on how many n there are in a row, the row is worth different values. This

[1] Russell, S. and Norvig, P., "Artificial Intelligence. A Modern Approach", Third Edition (New International Edition), Pearson, ISBN 13 978-1-292-02420-2, Aug, 2013., p. 172

value grows exponentially (with the exception of 4-in-a-row, the winning state), as the closer you are to winning, the better the row is:

- **4-in-a-row** - This is a winning move, and as such has a high value (the maximum value an integer can hold minus one, actually). This way, a winning move is never overridden by a non-winning move.
- **3-in-a-row** - 3-in-a-row is worth 9 points.
- **2-in-a-row** - 2-in-a-row is worth 3 points.
- **1-in-a-row** - 1-in-a-row is worth 1 point.

In the case the evaluation function finds a 4-in-a-row, it will stop all subsequent counting and immediately return the value of the 4-in-a-row. If the game is won, there is no point in figuring out what the value of the board is, the game is over.

Conclusion

We can conclude that the AI we have implemented plays somewhat intelligently (see Known Bugs) and with a decent speed (not taking more than a few seconds to calculate its move). It chooses its move based on the board state and tries to minimize the possibility of its opponent winning. When it plays against itself, it exhibits the same behaviour, making the game drag on for a while.

Known bugs

We have encountered one major bug with the AI. It plays smart, but it does not aim to win the game. Rather, it aims to prevent its opponent from winning, even if it has access to one (or more) winning moves.

Improvements

As mentioned in section we could have improved the algorithm by implementing alpha-beta-pruning and iterative deepening to give the algorithm a more correct evaluation of the possible moves. If move ordering is implemented, alpha-beta-pruning can be highly effective and in combination with iterative deepening, the algorithm would be able to evaluate far more moves which in turn would improve the AI's decision making.