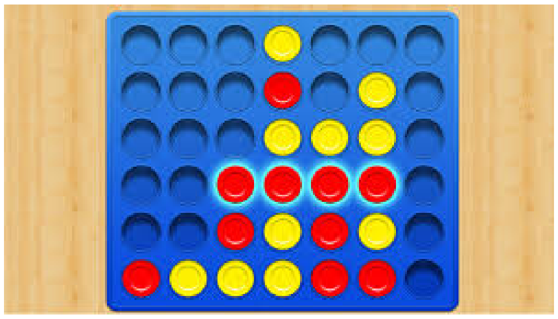
IT-University of Copenhagen

Intelligent systems programming

Connect 4

Tróndur Høgnason (thgn@itu.dk)
Kristian Mohr (kvin@itu.dk)



March 17, 2017

1 Connect 4

1.1 Minimax

MidOrFeed uses a minimax algorithm to calculate the optimal move. A minimax algorithm calculates the best move based on the assumption that the adversary has takes his best move previous to our previous move. In plainer terms, the algorithm seeks to maximize his score based the knowledge that the adversary will try to minimize our gain (maximize his gain).

1.2 Cutoff function

When we are using a minimax algorithm for Connect Four, it search through all the possible game states reachable. In Connect Four the searchspace is so large that it is unfeasible to examine all possible move and their consequences until only a few moves are left. Therefore it is necessary to restrict the search depth with a cut-off function. *MidOrFeed's* cut-off function is implemented using only tree depth. At the start of the game there is a hardcoded maximal search depth, and the minimax algorithm will return with best possible move so far, if it has reached the maximal tree depth. However, we have observed that increasing the search depth after a few moves doesn't give too bad of a performance, so after turn 12 the maximal search depth is incremented by one.

1.3 α-β pruning

Even with a cut-off function the searchspace is still quite large. To further reduce the searchspace *MidOrFeed's* minimax algorithm implements α-β pruning. α-β pruning. α-β pruning allows *MidOrFeed* to end the exploration of a branch early. Let us consider a n example, where the minimax algorithm is in the maximum phase in a state, where branch A will return a score of 4 and branch B leads to another branch, which has two option which give 3 and 5. In this example our α is 4. If we choose to follow branch B, the minimax algorithm will be in the minimum phase. So when the algorithm examine option 3, minimum will always pick that option. So we do not need to examine B's other options, they are pruned, since branch A will always be better (if both players play optimally).

To optimize the α-β pruning *MidOrFeed* explores the middle column first and then out towards the edges. In most cases this speeds up the pruning, since the best moves, especially at the start of the game, are usually found close to or in the middle column. And when the best moves are found earlier, there is a bigger chance that some branches can be pruned. This makes the searchspace smaller and in turn lets us use a higher maximal search depth. However if there are better moves further from the center column this approach will slow down the pruning.

1.4 Heuristics

The heuristic used scores a state based on the amount of possible lines of four tokens are started and possible to finish, compared to the opponent. An open line of length three is worth more than any line with two tokens filled which again is worth more than one. A line is worth the number of tokens squared. This means that a line of two tokens is worth four while a line of three is worth nine. If a line is blocked it is worth zero. A score for the opponent is calculated using the same method, and the score of his lines are subtracted from our score to calculate the final heuristic score.

The lines are starting from each field on the row towards the right, the two diagonals in the right direction and up the column. The fields in the three times three square in the upper right corner are not counted, since no lines can start from there. This means that fields towards the middle of the board will be used in multiple lines, which means that *MidOrFeed* will value positions with open lines in the middle higher, as they will give a greater combined score.

This gives an heuristic where more possible lines of four for you increases the score while possible lines for the opponent decreases the score. But lines towards the middle and lines with more places filled are worth more.

1.5 Precoded moves

As a result of the chosen heuristics and the rather naively implemented cut-off function *MidOrFeed* occasionally makes very bad moves at the start of the game, when the searchspace is very large. To prevent the bad moves at the beginning, we compared the starting moves to the actual perfect moves using a connect 4 solver.[1]

We identified the worst moves, and hardcoded *MidOrFeed's* actions in certain early game scenarios. This method is used at most until turn 7.

1.6 Further improvements

While testing *MidOrFeed*, we have seen that the heuristics we use are not optimal. An improved heuristic should probable include scores of threats. Having two tokens in a row then one gap and then another token for example should probably give a much larger heuristic score than just having two in a row. Currently the score difference between these is only 1. Even further one could explore if certain positions were more valuable to the starting player or the player going second.

The cut-off function is also implemented rather naively. A better approach would be to consider if a gamestate was suitable for cut-off rather than just having a fixed tree depth. We are however at the moment unsure how to evaluate how suitable a gamestate is for cut-off.

[1]`http://connect4.gamesolver.org`