Game Tree Searching by Min/Max Approximation

A summary by Kris Roosen as requested by Udacity for the 2nd project in the AIND program.

What follows is a brief summary of the paper written by Ronald. L. Rivest from the *Laboratory for Computer Science, MIT, Cambridge.*

In the mentioned paper an iterative method for searching min/max game trees using the idea of approximating the 'min' and 'max' operators by generalized mean-valued operators is presented. There operators are a good approximation of the min and max operators, but the difference is that they have continuous derivatives for all input values. For this purpose, a 2 player, zero sum, perfect information game is being investigated. The method is implemented so that nodes get a value of which the derivative indicates how much they impact the root node of the tree upon changing. A node's value can then be used to sort nodes from 'most impactful' to 'least impactful'.

The goal of this technique is to be able to direct searching effort towards the most impactful nodes and thus spending computation time more efficiently. Even if nodes are at a different branch or search depth, they can be preferred over other nodes. This is accomplished through an iterative process. At every iteration, the mean values are updated by selecting the best node, expanding it and scoring it. The best node can be selected by allocating weights called penalties to each tree edge and selecting the node of which the total penalty to the root is the smallest. This technique of growing the search tree one step at a time is called iterative deepening. The values of the newly explored nodes are backed up to the root repeatedly as the search depth increases. This is continued until time runs out and we return the last known best result of the search tree. Values of non-terminal nodes are calculated by a heuristic approximation function. This is a function that scores the value of a node based on static properties from the current game state.

The exploration algorithm can be implemented using the following steps repeatedly:

Step 1: If there are no expandable nodes: stop
Step 2: Pick the expandable node with the smallest penalty
Step 3: Add the successors of this node to the tree
Step 4: Compute the value and penalty of each added successor and all its ancestors back to the root.

One result that is worth mentioning, is that this technique pays attention to good backup moves, next to selecting the best move. This is not the case in minimax searching with alpha beta pruning. The proposed technique is tested in two scenarios. In the first scenario the number of calls to the move operator is counted. It is shown that the technique outperforms minimax searching with alpha beta pruning. In the second scenario, the CPU time is the limiting resource. In this case, alpha beta searching is superior. Further investigation and optimization is needed in order to beat alpha beta searching in this scenario.