Monte Carlo AI Documentation

by Nelson Hain

Monte Carlo Search Tree Algorithm

Monte Carlo is a tree based searching algorithm and uses state trees to make choices about a game state.

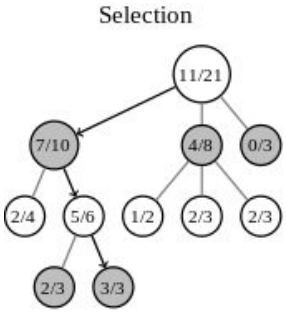
The algorithm works by searching through a graph, the nodes and edges of which have **state** and **action** properties. States are represented by nodes and contain information representing what is going on in an environment. E.g. game/simulation, how many pieces on a checkers board etc. Actions are things that could be done to a state which changes it. Such as adding a mark in tic-tac-toe or moving a chess piece.

Monte Carlo is based off the idea of random sampling and there are a few ways it can be implemented. However the basic idea is:

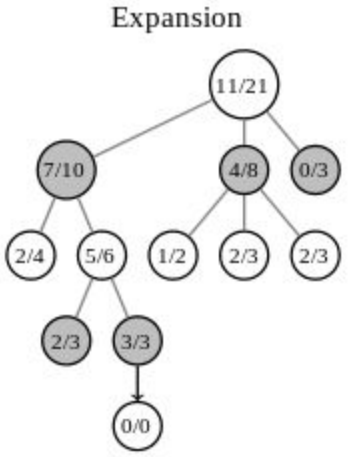
- For an AI opponent of state X, find all potential actions that can be taken.
- For each action, simulate a state, as if it had made that choice.
    1. Clone the current state.
    2. Update the clone to reflect the action taken.
    3. Randomly choose an action for the next player then update the state.
    4. Repeat steps 2-3 until the game ends.
    5. If the AI opponent won, give the state score a positive value, if lost a negative value and in the case of a draw 0.
- Simulate each action and repeat a determined number of times. Accumulate and return all values. Repeat for each original action until all original actions have a final score.
- AI then chooses action with the highest score.

The Monte Carlo Algorithm traditionally uses 4 main steps:

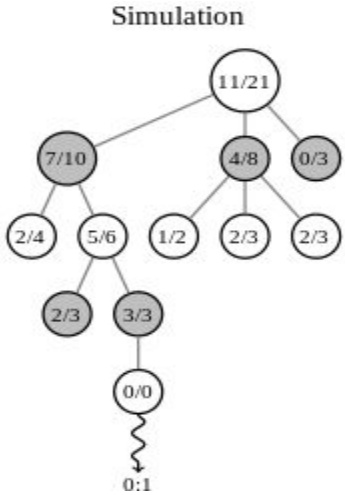
Selection - Randomly select a node in the current tree and recursively randomly select each child node until you reach a leaf node.



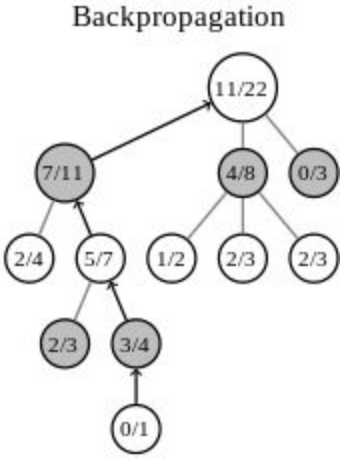
Expansion - If the leaf of the current tree doesn't end the game, then randomly create a potential child of that leaf.



Simulation - Run a simulated playout, until the end of the game is reached.



Backpropagation - Update the current path of the tree with the simulated result.



Monte Carlo Implementation

My individual implementation of the Monte Carlo Search Tree Algorithm stayed true the basic structure of the algorithm.
I saved the states of the board as their own Board object which would contain the layout of the board, as well as be able to perform operations on itself. The actions were piece-movements the player could make which corresponded with the rules of checkers.

When it was the turn of the AI player, the AI would simulate a ‘branch’ for each potential move it would make and simulate full games on each branch. If a simulation ran for longer than 1 second, the pieces would be counted and a score would be given based on whether or not the AI had more pieces on the board by that point. If they did, the score was added too. Otherwise the score was left untouched. At the end of all the simulations, the AI would make the move with the highest score.