Computer and Information Sciences
University of Strathclyde

Week 7 – Practical - Big Nim (worth 7% of your overall grade)

Aim

By now, you should have a game tree for Nimsticks, and have performed some pruning to make your game tree run faster. In this week's lab, we will undertake several tasks. Firstly, further optimising your game tree to make it run a little faster. This can then be used to create a simple AI player for a game. For larger games though, you will want an AI that will not freeze, so you should use Monte Carlo rollouts to generate results. Finally, as an extra task, you can generate a version of the AI that uses Monte Carlo Tree Search to decide its next move. Please read the submission instructions carefully.

Part 1: The Game of Nim

So far, you have focused on simply creating algorithms. Now, you can develop a game. We have provided you the base code for a human vs human game, or you can create your own implementation. You should extend this game to develop an AI player that can use your pruned minimax tree to play you at Nimsticks.

- A possible game should start as follows (starting state should be randomly generated):

  Let's play nim!
  How many piles initially? 3
  Maximum number of sticks? 10
  The initial state is [6, 9, 5].
  Do you want to go first or second?

  And so on...

- This means, after the user chooses a number n of piles and max number m of sticks, the computer generates n piles, all of them containing a random number <= m of sticks. Then the user decides whether or not to start and the game kicks off.

- To save you time, we have implemented a simple human vs human game that you can modify to add an AI player to. This AI player should use your optimised algorithm to choose a state, and should be able to beat a human. You should create an **AI_player_basic(state)** function, and use this to produce a state for the AI player to move next. You will need to return a state or path as well as a value from your minimax function so that the AI player knows which option to choose.

- You should keep a copy of your completed game, and then attempt to improve it in the next stage.

- ```
  game start  ([8, 8, 8], 1)
  Next move options:
  0.    [7, 8, 8]
  1.    [6, 8, 8]
  2.    [5, 8, 8]
  Enter next move option number 0
  You moved to state [7, 8, 8]

  AI player moved to state ([6, 8, 8], 1)
  Next move options:
  0.    [5, 8, 8]
  1.    [4, 8, 8]
  2.    [3, 8, 8]
  3.    [6, 7, 8]
  4.    [6, 6, 8]
  5.    [5, 6, 8]
  ```

Part 2: Monte Carlo Rollouts

In Part 1, you would have found that your AI player was quick at solving simple games, but still became very slow when it came to larger game trees. You should therefore create a new AI player (a new copy of your code) that uses Monte Carlo rollouts from successor states after depth limited search to decide on the next move.

- It will likely be a little slower for smaller trees, but should be much quicker for very large trees. For example, for [40,40,40,40,40], it should be able to choose a move in under 10 seconds of thinking time.

- This AI player should use your optimised algorithm, combined with an evaluation function, to choose a state, and should be able to beat a human. You should create an **AI_player_rollout(state)** function and use this to produce a state for the AI player to move next.

- The lecture notes will give you mode detail on creating rollouts, but when you reach a depth in your tree (3 levels might be sensible), you should create an eval function and pass in a state, and then generate and choose random successors until you reach a terminal node. The total should then be updated using the terminal value (-1 or +1), and you should also update a count

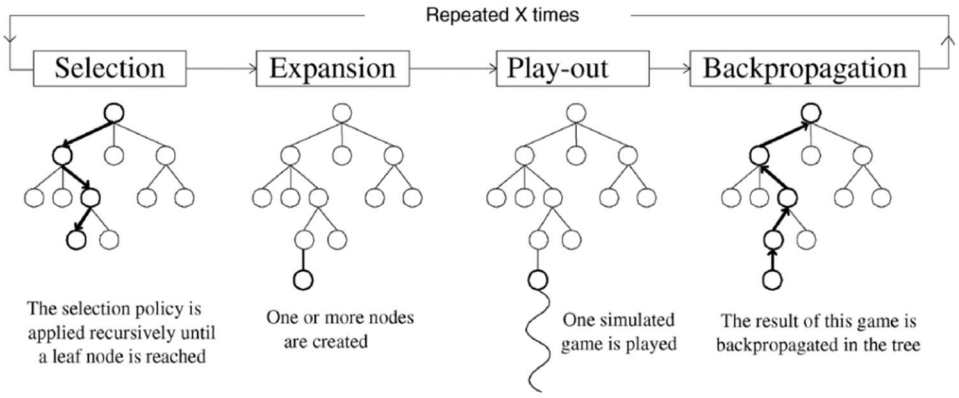
- Your eval state should then return total/count, to your minimax functions.

Part 4: Monte Carlo Tree Search

The final part is to improve your MCR AI by implementing full MCTS

- This is different from what you have calculated before, and is not just an edit of minimax. However, there are several things that will be the same, such as testing the terminal, and next states.
  ```

- As per your lectures, there are several stages, so you are advised to have at least a basic MCTS function, and then a function for each of the selection, expansion, play out, and backpropagation stages.



- It will likely be a little slower for smaller trees, but should be much quicker for very large trees. For example, for [20,20,20,20,20], it should be able to choose a move in well under 10 seconds of thinking time.

- This AI player should use full MCTS, and should be able to beat a human. Part 3 is is the most challenging aspect of this lab, and is only recommended for the most advanced students.

Submission Instructions

This will be evaluated by lab demonstrators. You should complete as much of the lab as you can, and then demonstrate and answer questions on the most advanced AI player you have completed.

- If you have implemented a basic AI player (part 1), you will get 1%

- If you have completed part 2, with the Monte Carol rollouts, you will get 2%

- If you have completed the full Monte Carlo Tree Search, you will get the full 4%.

- This is probably the most challenging lab of the semester, so do not feel disheartened if you do not get the full 7%!