Nim (Monte Carlo)

Help Center

Overview

Nim is a mathematical strategy game in which two players take turns removing items from one of several heaps. For this practice activity, we will focus on a simple variant of the game (sometimes called 21) in which a single heap starts with 21 items and the players then alternate having a choice of removing up to three items from the heap. The winner is the player to remove the last item from the heap. This game has a very simple optimal strategy that we will not reveal until you have completed your implementation.

Implementation

Your task for this activity is to write a Python function `evaluate_position(num_items)` that uses a Monte Carlo simulation to compute a good move for a given number of items in the heap. We suggest that you start from this template that provides a function `play_game(start_items)` which plays a game of Nim using `evaluate_position` to compute the move for the computer and `input` to allow the player to specify a move.

The function `evaluate_position` should choose an initial move in `range(MAX_REMOVE) + 1` using the following algorithm:

- For each possible initial move, make the given move (that is decrement `num_items` the specified amount),
- Play `TRIALS` games of Nim using randomly generated moves,
- Compute the fraction of games won for that initial move,
- Finally, choose the initial move that leads to the highest fraction of random games won.

To keep the performance of your code reasonable, we suggest that you use a `while` loop that iterates until the number of items in the current game reaches zero for each trial. In particular, avoid storing the random moves for each game in a list since this will reduce the number of trials that `evaluate_position` can process in a reasonable amount of time.

We suggest that you test your function using `play_game(start_items)` where the value of `start_items` is less than 10. If your Monte Carlo simulation is implemented correctly, `evaluate_position` should reliably return the optimal move as described below when the number of item is less than 10. You are welcome to consult our solution if you are stuck or need a nudge.

The optimal strategy

After you have played several games, the optimal strategy for Nim may be apparent to you. If it is your turn to move, you should remove exactly enough items so that the number of items remaining in the heap has remainder zero when divided by four. For example, if the heap contains 10 items, you should remove 2 items to leave 8 items remaining since 8 divided by 4 has remainder zero. Then, if your opponent removes n items where $1 \leq n \leq 3$, you should remove $4 - n$ items to restore heap to a state where the number of items remaining has remainder zero when divided by four. Repeating this process ensures that you can always remove the last item(s).

If the number of items in the heap happens to have remainder zero when it is your turn, you should remove a random number of items between one and three and hope that your opponent doesn't know the optimal strategy for Nim. Then, if the remainder on your next turn happens not to be zero,

choose your next move to make the remainder zero.

Observations from the Monte Carlo algorithm

In our implementation using 10000 trials for each possible move, the Monte Carlo simulation does a poor job of determining the optimal move until the number of items lies in the range $12 - 15$. At that point, simulation begins returning the optimal answer more frequently. By the time the number of items reaches 10, the algorithm reliably returned the optimal move.

The reason for the decay in the performance of the method lies in the behavior of `evaluate_position` for small numbers of items. For $1 \leq n \leq 3$ items remaining, the method computes that the initial move of n has a 100% win rate. For 4 items remaining, the method chooses to remove 1 item since this choice results in the most random games won at that point. For 5 items remaining, the method removes 1 item since the random games starting with 4 items remaining tend to favor the player that goes second. More generally, the effect of small games being won or lost fades as the number of starting items increases leading to the decreased performance for larger numbers of starting items.

Created Wed 4 Jun 2014 3:11 PM PDT

Last Modified Sat 19 Jul 2014 11:23 AM PDT