A Computational Approach to Solving Nim's Game

Ryan McKenna

1 Introduction

1.1 Background

Nim's Game, also known as Jianshizi, is a mathematical game of strategy in which players take turns removing stones from distinct piles. The winner is the person who takes the last stone from the pile. There are many variations of game play that include different number of piles as well as different rules for removing stones from piles.

1.2 Mathematics

Nim's game is a game of Perfect Information, which means that both players have access to the all the same information. Conversely, poker is a game of inperfect information, since the players do not know each others cards. Additionally, Nim's game is a deterministic game, that is, it does not depend on a random outcome.

2 Assumptions

In this paper, I will refer to a specific variation of Nim. However, the methods used in my approach can be generalized for other variations as well. The rules for this variation are as follows...

- Any number of piles
- Any number of stones per pile
- You may take from only one pile at a time
- You may take 1,2, or 3 stones at each turn

3 Solution

3.1 Approach

There are a few different approaches to coming up with a solution for this game. Due to the games simplicity and deterministic behavior, it is possible to give a direct proof of the winning strategy. For now, however, I will give a more general approach that can be easily abstracted to different variations of game play. This approach relies heavily on recursion for evaluating all the possible game configurations.

Additionally, I would like to note that since the game is deterministic and there is perfect information, it is possible to classify every game state as either a "losing configuration" or a "winning configuration". These terms have a somewhat circular defintion.

- A losing configuration is a configuration such that **for all possible plays**, your opponent will be left with a winning configuration.
- A winning configuration is a configuration in which **there exists a play** that results in a losing configuration for the opponent.

Notice that the complement of a losing configuration is a winning configuration and vice-versa. This will come in to play in the code.

3.2 Representation of the Game State

The representaiton of the game state is pretty simple. In this paper, I represent a game state as an array of ints, where each index represents a pile and each value represents the number of stones in that pile. For example {4,2,7} means there are 4 stones in the 0th pile, 2 in the 1st pile, and 7 in the 2nd pile.

3.3 Pseudocode

This recursive function will return a boolean which will be true if the configuration passed in is a winning configuration, and false otherwise.

```
boolean winConfig(piles) {
    if(all piles are empty) return false //you lose
    else {
        for each pile in piles
            for rm from 1  to 3
                if(removing rm stones from pile is not a winConfig)
                    return true
        return false
    }
}
```

Note that this approach will tell you whether or not any given configuration is a winning one or a losing one, but it will not tell you what play you have to make in order to win. The workaround for that is to create a global n dimensional array (where n is the number of piles) of winning plays and update it whenever a new winning configuration is discovered.

Also note that some special cases were left out to keep the function as basic as possible. For example, you can not take 3 stones from a pile that only has 2. Additionally, this function can be optimized by incorporating memoization, a technique used to cache results so they don't have to be recomputed.

3.4 Results

After running the function on various imputs, a pattern begins to emerge.

- For games with only a single pile, pile sizes which are multiples of 4 are losing configurations.[4,8,12,...]
- For games with two piles, if p1 mod 4 = p2 mod 4, then it is a losing configuration. [(1,1),(2,6),(9,5),(12,12)...]
- For games with three piles, if p1 mod 4 = p2 mod 4 and p3 mod 4 = 0, then it is a losing configuration. Additionally, if p1 mod 4 = 1, p2 mod 4 = 2, and p3 mod 4 = 3, then it is also a losing configuration. [(1,1,0),(3,7,8),(4,4,4),(5,6,7)...]

I leave it to the reader to prove the results[1] (or at least convince themselves why it is correct).

[1]Hint - Use Induction