CS 5660 Assignment one:

Design an efficient algorithm for a more complex version of NIM. First design an algorithm for determining whether a given problem is a win when it's your turn. Then use the definition of win to decide the move to make. You will hand in not just the final program, but each step along the way that you used to design the algorithm. Your steps will be the same as the ones I used in class for the simple NIM game: Step 1) write out in pseudo code the definition for win. This will be a recursive Boolean function. Step 2) Estimate how many recursive calls will be generated as a function of the input size. Step 3) Design a cache that can store all the possible problems (all possible unique calls). Step 4) Reformulate the recursive algorithm so that it uses the cache to compute a solution once, but use it multiple times. Step 5) Write the logic to play the game. Here, you only need to compute the cache once for the largest possible game size, and then you can play any number of games. No need to do the dynamic program yet.

The problem we are solving is exactly the same as the "reverse" game that can be played at:

http://www.online-games-zone.com/pages/puzzle/nim-master.php

The problem is defined as follows

Given: 3 piles, each containing n_i stones (or pieces). To make it interesting, generate a random number for n_i between 10 and 20.

Play: Players alternate turns. When it's your turn, pick a pile that contains at least one stone, select how many stones you want to remove, between one and all the stones. Remove the stones and let your opponent move. The player left with one stone loses.

Some hints and ideas:

Algorithm:

In the simple NIM the win function had a single argument that represents the number of pieces in one pile. Here, because you have 3 piles, win() should use 3 arguments, one for each pile.

The base case of win will have sum up the total number of stones in all the piles. If the total number is 1 then we terminate and return false.

The recursive case is more complex here than the simple NIM example in class. In the class example there were only two choices, either take one stone or two from the single pile. In this problem we have multiple choices, pick the pile, then pick how many for that pile. The algorithm you write will have to consider all possibilities. A recursive call will be needed for each pile and each count of stones removed. Easiest way to do this is with loops. The loop can be terminated early if a call returns false, meaning that the opponent will lose. **See the "change problem" in Chapter 3 in the book for some insights**.

Caching:

Think about how many unique calls there are. With three piles, each with 20 stones, then there could be 21 possible values for the first pile, 21 possible values for the second pile and 21 possible values for the third pile. Since we need to keep track of how many stones there are in each pile, and we need to store all possible combinations, we will need an 21 by 21 by 21, three dimensional array.

Turn in:

Your recursive algorithm design, your approximate analysis giving how many calls the recursive algorithm could make in the worst case, and your implemented program that uses caching. For the game player, there is no need to use a UI, a text interface is fine.