STEAM Carnival
DPS STS School Dhaka
Grade 9
Team: *Naturally Nash*

Nim Game

Algorithm and Explanation: Faiyaz Siddiquee

Nim - The Rules:

1) Initially, there will be a certain number of items available in the pile.

2) In a player's turn, the player must take either *#Option 1* number of items away, or *#Option 2* number of items away from the pile.

3) If a player has no valid moves to play on his/her turn, the player loses.

Nim is an impartial combinatorial game, meaning that the initial parameters of the game makes it possible for a certain player to win *if that player plays following the optimal strategy*.

Nim - The Strategy (© Taaroop):

Python script (Recursion-based Algorithm):

```python
def play(x, y, n):
    a = min(x, y)
    b = max(x, y)

    li = []
    status = "W"
    for i in range(b):
        if i%a == 0:
            if status == "W":
                status = "L"
            else:
                status = "W"
        if i == b:
            li.append("W")
        else:
            li.append(status)

    if n < b:
        return li[n]
    else:
        if play(x, y, n-x) == "L" or play(x, y, n-y) == "L":
            return "W"
        else:
            return "L"

n = int(input("Enter the total number of items: "))
x = int(input("Enter option 1: "))
y = int(input("Enter option 2: "))

if x == y or x < 0 or y < 0 or n < 0:
    print("Invalid parameters”)

if play(x, y, n) == "L":
    print("Okay, you go first!")
else:
    print("Okay, I go first!")

    if play(x, y, n-x) == "L":
        print("I take", x, "item(s). Remaining:", n-x)
        n -= x
    else:
        print("I take", y, "item(s). Items remaining:", n-y)
        n -= y

if n-x < 0 and n-y < 0:
    win = True
    print("You have no valid moves! I win!")
else:
    win = False

while win == False:

    opp_turn = -1
    while opp_turn != x and opp_turn != y or n-opp_turn < 0:
        opp_turn = int(input("Play your turn (has to be valid): "))
    n -= opp_turn
    print("Items remaining:", n)

    if play(x, y, n-x) == "L":
        print("I take", x, "item(s). Remaining:", n-x)
        n -= x
    else:
        print("I take", y, "item(s). Items remaining:", n-y)
        n -= y

    if n-x < 0 and n-y < 0:
        win = True
        print("You have no valid moves! I win!")
    else:
        win = False
```

Explanation of the Strategy:

What makes Nim deterministic is the fact that determination of the winner is *recursive*. That is:

Given initial number of items, n, and options a & b, if n is WINNING for Player 1, then $n - a$ **or** $n - b$ must be LOSING for Player 1.

[Assuming that $n \geq \max(a, b)$]

(Note that Player 1 signifies whosoever's turn it is on that particular number of remaining items)

Using this principle, it is obvious that Player 1 can manage a win given the parameters **if and only if** he/she can put Player 2 in a losing position in the next move, and vice versa. Same goes for Player 2.

Hence, all that is left to do is to determine which initial positions (n) is winning/losing for Player 1 for $0 \leq n < \max(a, b)$. The program picks whoever goes first in such a way that it always wins!

For $0 \leq n < \max(a, b)$, the status (winning or losing) for Player 1 goes as follows ($n = 0, 1, 2, \ldots$):

$\underbrace{LL\ldots L}_{\min(a,\, b)}$ $\underbrace{WW\ldots W}_{\min(a,\, b)}$ and so on UNTIL max(a, b).

To clarify, $n = \max(a, b)$ is obviously winning for Player 1.

The validation of the final paragraph is left as an exercise to the readers (Hint: Use the recursive lemma presented earlier).

□