Programming Assignment #2

Implementing Mini-Max to play a game

Introduction

Mancala is an ancient game that traces its way back to at least the 7th century. There are many variations of the game but it is normally played on a "two sided" board where each player has six pits on their side of the board with a seventh "well" or "Mancala" on the right end of the board:

(Read about it on wikipedia or try to play it online)

As normally played, mancala is a bit too big for our MiniMax to "solve" it on a personal computer. So we are going to be working with a variation of the game that will be a hybrid with Nim. It will only be played on one side of the board. We will call this game "Nimcala"

Rules

For our version of this new game we are going to assume a board with N pits.

- At the start of the game, some number of stones reside in each of the pits.
  - Thus, the opening board might look like this:
When it is their turn a player picks up ALL of the stones from a single pit. They then add one stone to each pit that comes after the pit they emptied. If there are extra stones left over these stones are removed from the game.

Thus, a player picking up the stones in pit 2 for the opening move would produce:

```
3
0 4 4 4 3
```

While a player picking up the stones in pit 4 for the opening move would produce:

```
3 3 3 3 0 4 4
```

The winner is the player who picks up the LAST stone.

Thus, in theory, the shortest game is 6 moves since player one could pick up/redistribute the stones in pit 1, then player two could pick up/redistribute the stones in pit 2, then player one could pick up the stones in pit 3, and so on, until player 2 picked up the remaining stones in pit 6 (which have no place to go but off the board). Of course, that would mean that player one played very foolishly.

If you want to experiment with the mechanics of this game you can use this simple Scratch program to play around.

**Instructions**

For this homework you should write a program (preferably Python but I will accept other languages if you talk to me first) that fits the following requirements:
Your code must provide a function called `search()`

- `search()` must accept a length N string of numbers which we will assume is the encoding of the N pits.
  - Thus, the board used above in my examples would be "333333" as illustrated above, and the two "after move" examples would be "304443" and "333044" respectively.
  - For simplicity we will assume that no pit has more than 9 stones at the start of the game, meaning your program should accept/handle all of the valid length N strings from 000000 to 999999
  - Having said that, once the MiniMax search starts it must be able to handle double digit numbers in pits.
    - Starting with 999999 and picking up the stones in pit one produces 0 10 10 10 9 9
  - It is assumed that when we invoke `search()` that the next turn is Max's turn.
  - You must devise a program which uses MiniMax to search for the "optimal" next move for the current player.
    - For our evaluation function we will use the base evaluation of:
      - 1 for Max win
      - -1 for Min win
      - There are no ties
    - Thus, Max should choose a move that will allow her to win IF such a move is available and Min should choose a move that will allow him to win IF such a move is available
  - Your search function should RETURN a tuple consisting of two items,
    - The pit # that the current player should pick from. [For consistency this should be numbered as a HUMAN would number them not as the computer would number them. This is what I did in my examples above]
    - The string of either "win" or "lose" which indicates whether we expect Max to win or lose with this move.
    - For example:
      - >>>> search("33333")
        (4, 'win')
      - >>>> search("33304")
        (1, 'lose')
      - >>>> search("00011")
        (5, 'win')
Testing your results

How do you know you are getting good results? Don't overlook those simple boards where you know the right answer:

```python
>>> search("00001")
(5, 'win')
>>> search("00010")
(4, 'lose')
>>> search("10000")
(1, 'win')
```

It is always best to start with ones like this where you can SEE the right answer. Only then do you need to move on and try the base game of "33333" (whose answer is above).

Submitting Work

Please submit your code to the appropriate dropbox on Blackboard.