HW #04 – Racket #2
CSCI-400 Spring 2013

In this assignment you are to write a Racket program that plays the game of NIM. The rules of NIM are simple. The game board consists of a number of rows with each row containing a number of sticks. Players alternate turns and, on their turn, must remove one or more sticks from a single row of their choice. The player that takes the last stick on the game board wins.

You will implement three types of players. The first player type is the "human" player and when it is their turn the program will ask them for a row and then ask them for the number of sticks to remove from that row. If the move is legal, the computer will update the board and go to the next player. If the move is illegal, the computer will inform the user of that fact, redraw the board, and ask the user for a new row and number of sticks. This will continue until the user enters a legal move. The second player type is the "random" player and when it is their turn they randomly choose any valid move. The third player type is the "smart" player and when it is their turn they will make a move that, if possible, will guarantee them a win.

The top level specification for your function is as follows:

```
(NIM board `(player player))
```

Where:	**`board`**	is a list of rows with each row being a list of sticks where a 'stick' is represented by an uppercase '**`X`**'.
	`player`	(**`human`** | **`random`** | **`smart`**)

Hence the following would be one way to play a game:

```
(define board1 `[(X) (X X X) (X X X X X) (X X X X X X X)])
(define players `(human smart))
(NIM board1  players)
```

The exact manner in which you display the game board and information is up to you. The required elements are that the board must be displayed between each move, the board must be displayed graphically (using text), whose turn it is must be indicated, and the move that was chosen must be displayed, even if the move was made by a computer player. In addition, an appropriate error message must be displayed if the user makes an illegal move and the board must be redisplayed before asking them for a new move. Finally, the identity of the winner must be displayed.

An example run of NIM is shown on the next page. Note that the function must return the number of the player that won, which explains the number on the final line. This is the return value of the expression and not something displayed by the NIM procedure directly.

HW #04 – Racket #2
CSCI-400 Spring 2013

```
WELCOME TO NIM!
--------------------------
Row 1: X
Row 2: X X X
Row 3: X X X X X
Player 1’s turn
--------------------------
Which row do you choose:.. 3
How many sticks:.......... 5
--------------------------
Row 1: X
Row 2: X X X
Row 3:
Player 2’s turn
--------------------------
Which row do you choose:.. 2
How many sticks:.......... 2
--------------------------
Row 1: X
Row 2: X
Row 3:
Player 1’s turn
--------------------------
Which row do you choose:.. 3
How many sticks:.......... 1
ILLEGAL MOVE! – TRY AGAIN
--------------------------
Row 1: X
Row 2: X
Row 3:
Player 1’s turn
--------------------------
Which row do you choose:.. 2
How many sticks:.......... 1
--------------------------
Row 1: X
Row 2:
Row 3:
Player 2’s turn
--------------------------
Which row do you choose:.. 1
How many sticks:.......... 1
--------------------------
PLAYER 2 WINS!
2
```

HW #04 – Racket #2
CSCI-400 Spring 2013

GRADING RUBRIC – 80 pts

20	**- Good Faith effort.**
5	**- Procedure interface is correct (arguments and return value).**
5	**- Displays the game board in a suitable manner between each move.**
5	**- Prompts the human player for row and number of sticks.**
5	**- Validates the move and displays an error and reprompts as needed.**
5	**- Updates the board correctly for the move made.**
5	**- Displays a message at game end indicating which player won.**
5	**- Random player makes random, but valid choices.**
5	**- Smart player always wins when possible.**
10	**- Code is modular with small, well-defined and documented functions.**
10	**- Code makes heavy use of recursion.**
-10	**- Each occurrence of any (let) or related function.**
-20	**- Each occurrence of any (set!) or related function.**
-10	**- Improper submission.**

SUBMISSION

Zip up all files into a .zip file named

`CS400_UserID_R_02.zip`

Hints

You will probably want to look over the bitwise operations available in Racket, such as (bitwise-xor) and (bitwise-and), as well as the (random) function. In addition, you may find it useful to look into the (apply) function to see if it might be useful.

To help you get into the "recursive" mindset, consider that after each move what you are left with is a game board and two players, one of whom will make the next move. From a logical perspective, this is essentially the start of a new, smaller game.