Lab 7 – The Game of Nim

1. Currently, a game-state is represented as a single cons cell containing the number of coins in each pile. For example, the cons cell `(5 . 8)` represents the state of having 5 coins in the first pile and 8 coins in the second pile. However, it might be useful to include a symbol in the representation of game-states indicating its *data type*, like we did for move-instructions. Suppose we change the representation of a game-state to be an ordinary list containing the symbol `state`, the number of coins in the first pile, and the number of coins in the second pile (in that order). For example, the above game-state would now be represented as the list

   ```
   (state 5 8)
   ```

   This way, we can easily distinguish game-states from move-instructions, just by looking at the type symbols in their representations (*i.e.*, `state` versus `move`). Rewrite the constructor function **`make-game-state`** and the selector function **`size-of-pile`** to support this new representation for game-states. Then test your code to make sure everything still works.

2. We also added support for *strategies* to our Nim code. The `simple-strategy` function takes one coin from the first pile if it is not empty, otherwise it takes one coin from the second pile. However, this is not a very good strategy. Write a better strategy function called **`take-all-of-first-nonempty`**, which takes all coins from the first pile if it is non-empty, otherwise it takes all coins from the second pile. Remember that a strategy function should take a *game-state* as input and return a *move-instruction* as output. To play against this strategy, provide it as an argument to `play-with-turns`, as shown below:

   ```
   (play-with-turns (make-game-state 5 8) 'human take-all-of-first-nonempty)
   ```

3. Write a strategy function called **`take-one-from-random-pile`**, which randomly selects a non-empty pile and then removes one coin from it. You can use Scheme's built-in `(random` *n*`)` function as a helper, which returns a random integer in the range 0 up to, but not including, *n*.

4. Write another strategy function called **`take-random-from-random-pile`**, which chooses both the pile and the number of coins to remove randomly. However, make sure that the move-instruction returned by the strategy is valid, and does not specify taking coins from an empty pile.

5. (Exercise 6.18 from *Concrete Abstractions*) If we consider the chocolate bar version of Nim, we can describe a strategy that allows you to win whenever possible. Remember that in this version, the players alternate breaking off pieces of the bar along a horizontal or a vertical line, and the person who gets the last square of chocolate loses (so the person who makes the last possible break wins, just as the person who takes the last coin wins). If it's your turn and the chocolate bar is not square, you can always break off a piece that makes the bar into a square. If you do so, your opponent must make it into a non-square. If you always hand your opponent a square, he will get smaller and smaller squares, leading eventually to the minimal square (*i.e.*, the poisoned square). Write a function called **`chocolate-bar-strategy`** which implements this strategy in 2-pile Nim. What action should it take if presented with (the equivalent of) a square chocolate bar?

6. (Exercise 6.19 from *Concrete Abstractions*) Suppose you want to randomly intermingle two different strategies. How can this be done? The answer is with higher-order functions. Write a function called **`random-mix-of`** that takes two strategies as arguments and returns a new strategy function that randomly chooses between these two strategies on each turn. Thus, a call of the form

   ```
   (play-with-turns (make-game-state 5 8)
                    'human
                    (random-mix-of simple-strategy take-all-of-first-nonempty))
   ```

   would randomly choose at each turn between taking one coin or all the coins from the first non-empty pile.

7. Create a new copy of your program file called **three-pile-nim.scm**, and modify the code to play Nim with *three piles* of coins instead of two. You will need to make several modifications to your functions, as well as to the representation of game-states.