Functional Programming in Erlang


futurelearn.com/courses/functional-programming-erlang/1/steps/161082

In these exercises you'll use functions and higher-order functions to define strategies and game-playing functions for the rock-paper-scissors game.

Do make use of the comments here to share and discuss your approach to these challenging exercises.

There is a supporting file, `rps.erl`, available under 'Downloads' below.

Strategies

A **strategy** is a function from the list of opponent's plays (latest at the head) to a choice of play for the next turn.

These include:

- **rock** - always make the `choice` of "rock".
- **echo** choose the opponent's last move as your next one

both of which are defined in the `rps.erl` file provided.

Add to this file strategies that:

- assume that your opponent **never repeats** herself: if you know this you can make a choice that will never lose;
- make a **random** choice each time; you may want to use the `random:uniform/1` function so that `random:uniform(N)` returns a random choice of `1`,`2`, … `N` with equal probability each time;
- **cycles** through the three choices in some order;
- apply an analysis to the previous plays and choose the **least frequent**, assuming that in the long run your opponent will play each choice equally;
- apply an analysis to the previous plays and choose the **most frequent**, assuming that in the long run your opponent is going to play that choice more often than the others.

We can also define functions that combine strategies and return strategies:

- Define a strategy that takes a list of strategies and each play chooses a random one to apply.
- Define a strategy that takes a list of strategies and each play chooses from the list the strategy which gets the best result when played against the list of plays made so far.

Strategy *vs* Strategy

Define a function that takes three arguments: two strategies and a number `N`, and which plays the strategies against each other for `N` turns. At each stage the function should output the result of the round, and it should show the result of the tournament at the end.

You could also choose to modify this so that the game is ended when one player is more than M points ahead of the other, for example.

Modify the `play` functions

Modify the `play/1` and `play/2` functions in `rps.erl` so that they show the overall result of the tournament when it is ended.
