CS302: Paradigms of Programming

Spring 2021

PA2: Programming with Streams

As we have studied in the class, streams are a lazy data structure that allow us to work efficiently with long or even infinite sequences. Apart from the modular elegance, they also allow us to model the real world in a time-less manner. In this assignment, we implement an entertaining and instructive two-player game that serves as a concrete illustration of the newly learnt paradigm.

1 Problem: Hobbit, Dwarf, Elf, Orc

Recall Lord of the Rings, and the power of Hobbits? In this assignment, we implement a game that we can play in Covid times even over phone calls. In each round, each player calls out one of the characters among *Hobbit*, *Dwarf*, *Elf* and *Orc*. The winner is determined by the rule "orc is stronger than hobbit", "hobbit outwits dwarf", "dwarf is heavier than elf", "elf is faster than orc", "dwarf kills orc", and "hobbit can enter where elf cannot", where in each relation the former character wins over the latter. Thus, in a particular round, if player 1 calls out an orc and player 2 calls out a dwarf, then player 2 wins that round. If both players call out the same character, then the round is a tie. The game continues in this manner for a fixed number of rounds (agreed in advance to avoid cheating!), and the player having won the highest number of rounds wins the game.

2 Strategies

Say each player can follow either of the following two strategies:

- **Strategy 1: Lazy**. Always call out whom the opponent called out in the previous round.
- **Strategy 2: (Over)Smart**. Find the favorite character of the opponent by analyzing all the characters called out by him/her so far, and call out the next character as the higher winning character (taking fewer-lettered one in case of multiple options). For example, if the favorite character of the opponent is found to be *Elf*, then this strategy, among the winning ones *Hobbit* and *Dwarf*, would call out *Dwarf* as the next character.

3 Assignment

Imagine the sequence of the calls generated by each player as two infinite streams. Apart from this, construct two players such that the strategy chosen by each player is decided before-hand (our players don't plan to learn about artificial neural networks). Notice that similar to our `primes` and `prime?` example from the class, both the strategies rely on the fact that though the stream to be examined is infinite, enough of it has been generated to decide the next call. Your program should consist of a top level function called `play` that takes four inputs: strategy for player one (as a number), strategy for player two (as another number), the first move of player one (as a number), and the number of rounds to simulate. The function `play` should return either 1 or 2 indicating the player who won the game, or 0 indicating that there was a tie overall. The codes for the first move of the first player are: Elf 1; Orc 2; Dwarf 3; and Hobbit 4. For the second player, assume that the first move is always *Hobbit*, irrespective of the strategy employed. Further, make sure the two streams are retained in global variables *Fu* (player 1) and *Pooh* (player 2) so that we should be able to print a particular element of the streams, if required.

4 Submission

- You need to submit a single file named `rollnum-pa2.scm` (where `rollnum` is your roll number in small letters) consisting of the function `play`, and the streams `Fu` and `Pooh` defined in the global environment.
- Make sure your file contains `#lang sicp` at the top, and that it doesn't depend on any other modules.
- Any other helper functions or variables that you use should be modularized and invisible to the outside world.

5 Evaluation

We will be using a testcase-based automated evaluation to check various standard as well as corner cases. You will get marks based on how many testcases does your submission pass.

6 Plagiarism Warning

The assignment has to be done individually. Any hint of plagiarism (caught easily using automated tools) will lead to serious implications.