Functional Programming: Explore the Desert (Part 1)

Laurent Christophe, Wolfgang De Meuter

Programming Project: Assignment #1 (2017 - 2018)

1 Introduction

The final mark for the Functional Programming course is calculated as follows: (i) 25% for the first programming assignment (ii) 25% for the second programming assignment (iii) 50% for the oral exam. All exam parts have to be done in order to obtain a final mark. This document describes the first programming assignment. Submission is done by sending your raw code files to the teaching assistant: Laurent Christophe (`lachrist@vub.ac.be`). This first assignment is due 4th December (08:00 AM). Notice that the execution of the project is strictly individual and no plagiarism shall be tolerated! Any exchange of code will be considered plagiarism. The project will be marked according to how well you fulfill the functional requirements described below and according to how well you apply the concepts explained during the lectures and the lab sessions. If you encounter any problem or if you have a precise question, feel free to contact the assistant at `lachrist@vub.ac.be`.

2 Informal description

In this first programming assignment you will have to implement an adventurer game. In this game, the player incarnates an explorer on the lookout for treasures in some desolated place where water is scarce and lava omnipresent. His goal is to find as many treasures as possible before escaping through a portal. As he explores the map, he will have to manage his water supplies. If he runs out of water, he instantaneously dies from thirst. However, he can refill his water stock on occasional oasis. To help him in his task, he received a magic compass which indicates the distances to the closest oasis, the closest treasure and the closest portal.

3 Requirements

The goal of this first programming assignment is to create a terminal-based game about exploring the desert. Your project should be entirely written in Haskell and compiled using the Glasgow Haskell Compiler [1]. The desert is modeled as an infinite matrix of tiles randomly generated using the function `mkStdGen` from an integer seed provided by the player. The explorer starts at the position `(0,0)` and cannot walk outside the map (negative coordinate). The explorer can move vertically (up and down) and horizontally (left and right) using the usual keys: `w`, `a`, `s`, `d`. The way the map is displayed is left to your discretion. However, once a tile has been revealed by the line of sight of the explorer, it should remain so. Each move costs the explorer a measure of water and the game is lost if the player runs out of water. Make sure that the exploration of the map does not affect its generation. In other words, the map should be entirely defined by the player-provided seed. There exists four different tiles:

- Desert tile: may contain a treasure which should be automatically collected if the explorer walks into the tile.
- Water tile: if the explorer walks (swims) into it, its stock of water should automatically be refilled.
- Lava tile: if the explorer walks (jump) into it, the game is lost.

[1] `https://www.haskell.org/ghc/`

- Portal tile: if the explorer walks into it, the game is won and the number of collected treasure should be displayed.

The player should be able to parametrized the game with the integers below:

- `s`: The line of sight of the explorer as a number exploration step.
- `m`: The maximum number of measure of water the explorer can carry.
- `g`: The initial seed to randomly generate the tiles.
- `t`: The likelihood, in percentage, of a desert tile to contain a treasure.
- `w`: The likelihood, in percentage, of a water tile generation.
- `p`: The likelihood, in percentage, of a portal tile generation.
- `l`: The likelihood, in percentage, of a lava tile generation when none of the previously-generated adjacent tiles is lava.
- `ll`: The likelihood, in percentage, of a lava tile generation when at least one of the previously-generated adjacent tiles is lava.

Make sure invalid parameters are appropriately handled and do not crash the game. For instance, the application should not accept `w + p + l > 100` nor `w + p + ll > 100`. At any time, the player should have access to the information below:

- The number of treasures collected.
- The number of measures of water left.
- The distance, in exploration step, to the closest water tile accounting for lava.
- The distance, in exploration step, to the closest desert tile containing a treasure accounting for lava.
- The distance, in exploration step, to the closest portal tile accounting for lava.

Provide two versions of your distance computation function. One should be fully lazy while the other should not contain any memory leak. Demonstrate the difference in memory consumption between the two versions by tuning the game parameters so that it can be noticed by profiling the heap. Include the resulting graphs in your submission archive.

4 Part 2

In part 2, we will use parser combinators to make the game persistent, and software transactional memory to introduce concurrency. This will be described in a follow-up document.