Functional Programming: Explore the Desert (Part 2)

Laurent Christophe, Wolfgang De Meuter

Programming Project: Assignment #2 (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 second programming assignment. Submission is done by sending your raw code files to the teaching assistant: Laurent Christophe (`lachrist@vub.ac.be`). This second assignment is due 15th January (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 questions, feel free to contact the assistant at `lachrist@vub.ac.be`.

2 Brief Summary

For this second assignment you will have to apply three improvements over the game "Explore the Desert" which you developed for the first assignment:

1. Add new characters to the game. They need to operate concurrently using software transactional memory (STM).
2. Make the game persistent by adding save/load functionalities based on parser combinators.
3. Make a graphical user interface for the game using the Gloss graphical library.

3 Concurrency

The first improvement consists in ramping up the difficulty by adding (sand) worms to the game. If the explorer occupies a tile also occupied by a worm's body, the game is lost. A worm's body is linear and can be represented as a list of adjacent positions whose first element is the head of the worm. Worms are all of the same length which should be parameterizable in the same way as for the other parameters. Worms lifespan consists in the following phases: (i) At every exploration step, a worm's head may spawn from any desert tile which contain no treasure, no explorer, and no worm's body. The likelyhood of worm spawning should be parameterizable in the same way as for the other parameters. (ii) At every exploration step, the worm's head should move to an adjacent tile. A worm's head can only move into desert tiles without treasure and without worm's body. If there exists no legal move, the worm should start the next phase. The rest of the worm's body should follow its head and an additional segment should be added at the initial head position. The third and final phase begins once the worm has entirely dug himself out of the sand. (iii) At every exploration step, the worm's head digs deeper and deeper into the sand leading its body to disappear segment by segment into the final head position. Once the last worm's segment disappeared into the sand, the worm's lifespan is over.

In your implementation, each worm should correspond to a forked process which communicates through software transactional memory. Bonus points will be attributed if worms collaborate to establish a collective strategy for catching the explorer.

4 Persistency

The second improvement consists in making the game persistent. To that end, you will have to implement two new functionalities: (i) At every exploration step, the player should have to opportunity to save the current state of the game to a text file. (ii) Upon launching your application, the user should be offered two options: start a new game or load one from a text file. The format of the text file is defined in Table 1. It uses the parameters given by the user to encode the entire (infinite) map and is otherwise pretty much self-explanatory.

5 Graphical User Interface

The third improvement consists in creating a graphical user interface for the game using the Gloss[1] graphical library. The rendering of the game is left to your appreciation but it should enable all the functionalities described in the first and second assignments. If you have a recent version of GHC (8.2.x), Gloss should be as easy to install as:

```
cabal update
cabal install gloss
```

You should then be able to compile and execute the hello world program below:

```
import Graphics.Gloss
main = display (InWindow "Nice Window" (200, 200) (10, 10)) white (Circle 80)
```

To help you get started, you can draw some inspiration from the `gloss-examples`[2]. These examples can be installed and executed as follow:

```
cabal install gloss-examples
~/Library/Haskell/bin/gloss-draw # OSX-specific
```

I've noted the following examples to be of interest:

- Rendering a matrix: `gloss-conway`.
- Handeling events: `gloss-draw`.
- Fullscreen: `gloss-gravity`.

[1]http://code.ouroborus.net/gloss/icebox/gloss-head.old-darcs/gloss-examples/picture/
[2]http://code.ouroborus.net/gloss/icebox/gloss-head.old-darcs/gloss-examples/picture/

Grammar Rule			Comments
`GAME`	$\rightarrow$	`LINE LINES`	Game’s serialization
`LINE`	$\rightarrow$	`position ( POSITION )`	The explorer’s position
		`supply ( NATURAL )`	The explorer’s water supply
		`revealed ( POSITION )`	Revealed tile
		`collected ( POSITION )`	Collected treasure
		`emerging ( POSITION POSITIONS )`	Emerging worm
		`disappearing ( POSITION POSITIONS )`	Disappearing worm
		`s ( NATURAL )`	Explorer’s line of sight
		`m ( NATURAL )`	Explorer’s water capacity
		`g ( NATURAL )`	Initial seed
		`t ( NATURAL )`	Treasure likelihood
		`w ( NATURAL )`	Water likelihood
		`p ( NATURAL )`	Portal likelihood
		`l ( NATURAL )`	Lava likelihood
		`ll ( NATURAL )`	Adjacent lava likelihood
		`x ( NATURAL )`	Worms length
		`y ( NATURAL )`	Worms spawning rate
`LINES`	$\rightarrow$	`\n LINE LINES`	List of lines
		ϵ	
`POSITION`	$\rightarrow$	`[ NATURAL , NATURAL ]`	X, Y coordinates
`POSITIONS`	$\rightarrow$	`, POSITION POSITIONS`	List of positions
		ϵ	

Table 1: Serialization format. `NATURAL` denotes any natural numbers.