`Functional Programming`
Project: Desert Explorer (part 2)

Assistant: Noah Van Es
`noah.van.es@vub.be`

The final grade of the course is determined by an oral exam and a project (each counts for 50% of the total grade for the course). The project consists out of two parts. A non-empty solution for both parts has to be submitted in order to get a grade. This document describes the second part of the project.

Project description

For the second part of this project, you will have to apply three improvements to the "desert explorer" game you developed previously. Specifically, these involve:

GUI making a graphical user interface for the game using the Gloss graphical library.

Concurrency adding new characters to the game. They need to operate concurrently using software transactional memory (STM).

Persistency making the game persistent by adding save/load functionalities based on parser combinators.

Practical information

Submission The deadline for this assignment is **June 10, 2025 (23:59)**. You submit all your code and other materials in single ZIP-file through the Canvas platform.

Grading Note that your project will not only be graded on functionality, but also on the quality of your implementation (programming style, performance considerations, ...).

Individual work In case you have questions or need clarification to complete this project, feel free to contact the assistent by mail (`noah.van.es@vub.be`) or make an appointment to schedule a meeting (virtual or at my office).
The project has to be executed on a strictly individual basis! Automated tools are used to scan for plagiarism between projects; any suspicion of code exchange shall immediately be reported to the faculty's dean, and both parties shall be sanctioned in accordance with the examination rules regarding plagiarism.

Project assignment

This section details the specific requirements for each improvement.

GUI

The first improvement consists in creating a graphical user interface for the game using the Gloss graphical library[1]. The rendering of the game is left to your discretion, as long as it supports all the functionalities described in this and the previous part of the project assignment. You can install Gloss using the following commands:

```
cabal update && cabal install --lib gloss
```

You should then be able to compile and run the following program:

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

To help you get started, you can have a look at some examples from the `gloss-examples` library[2]. These can be installed using Cabal and then executed as follows:

```
cabal install gloss-examples
~/.cabal/bin/gloss-draw   # might be installation-dependent
```

The `gloss-draw` example is interesting to examine how events can be handled. Similarly, `gloss-conway` and `gloss-gravity` demonstrate how matrices can be rendered and how full-screen mode can be enabled, respectively.

Concurrency

The second improvement consists in ramping up the difficulty by adding (sand) worms to the game. If the explorer occupies a tile also occupied by any part of 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 and spawn at a certain rate; both of these parameters should be given together with the other game parameters (cf. previous assignment). This time however, all parameters should be passed as options on the command line and handled appropriately.

A worm's lifespan consists out of the following phases:

- At every exploration step, a worm's head may spawn from any desert tile which contains no treasure, no explorer, and no other worm's body.
- 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 a worm's body. If there is 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.

[1] `https://hackage.haskell.org/package/gloss`
[2] `https://hackage.haskell.org/package/gloss-examples-1.13.0.4/src/`

- The third and final phase begins once the worm has entirely dug himself out of the sand. 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 worm's last segment disappeared into the sand, its 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.

Persistency

The third improvement consists in making the game persistent. To that end, you will have to implement two new functionalities:

- At every exploration step, the player should have to ability to save the current state of the game to a text file.
- Upon launching your application, the user should be able to choose between two options: starting a new game or loading one from a text file.

The format that should be followed for this text file is defined in Table 1. The current game state, as well as the game's parameters, are encoded in this format.

Grammar Rule			Comments
GAME	→	LINE LINES	The game's serialization
LINES	→	\n LINE LINES	A list of lines
		ϵ	
LINE	→	position (POSITION)	The explorer's position
		supply (NATURAL)	The explorer's water supply
		revealed (POSITION)	A revealed tile
		collected (POSITION)	Collected treasures
		emerging (POSITION POSITIONS)	An emerging worm
		disappearing (POSITION POSITIONS)	A disappearing worm
		s (NATURAL)	The explorer's line of sight
		m (NATURAL)	The explorer's water capacity
		g (NATURAL)	The initial seed
		t (NATURAL)	The treasure likelihood
		w (NATURAL)	The water likelihood
		p (NATURAL)	The portal likelihood
		l (NATURAL)	The lava likelihood
		ll (NATURAL)	The adjacent lava likelihood
		x (NATURAL)	The length of a worm
		y (NATURAL)	The spawning rate of worms
POSITION	→	[NATURAL , NATURAL]	(x, y) coordinates
POSITIONS	→	, POSITION POSITIONS	A list of positions
		ϵ	

Table 1: The serialization format. NATURAL denotes any natural number.