Programming Paradigms

Lab 5. Functional programming in Haskell. N-body simulation

Outline

- Higher-order functions and lists in Haskell recap
- User-defined types in Haskell recap
- Exercise: N-body simulation

N-body simulation: prerequisites

Exercise 5.1.
Define type **`Body`** to represent state of a given body: mass, 2D position, and velocity vector. Implement function **`renderBody`** to render a given body. Define some bodies (e.g., earth, moon, or sun).

Exercise 5.2.
Define type **`System`** to represent a system (collection) of bodies for simulation. Implement function **`renderSystem`** to render an entire system. Define a sample system (e.g., earth+moon or earth+sun).

N-body simulation: moving bodies

Exercise 5.3.
Define a function **`moveBody :: Double -> Body -> Body`** that updates body's position based on its velocity and given time (in seconds).

Exercise 5.4.
Define a function **`updateSystem`** to update positions and velocities of all bodies in a given system over a given time (in seconds).

You can use the following snippet for more details:
https://code.world/haskell#P__XNId0wAKIATT95b_Idzw

N-body simulation: computing gravity effect

Exercise 5.5.
Define a function **`gravityAcc :: Body -> Body -> Vector`** that computes the acceleration that second body gets as the gravitational effect of the first body. Use the following formula:

$$a_2 = G\frac{m_1}{|\mathbf{r}|^3}\mathbf{r}$$

Exercise 5.6.
Define a function **`applyGravity :: Double -> [Body] -> Body -> Body`** that computes the combined gravity effect of a collection of bodies on a given body over given time (in seconds) and returns the new state of the body.

N-body simulation: putting things together

Exercise 5.7.
Define a function `updateBody :: Double -> [Body] -> Body -> Body` that updates both position and velocity of a body. Use this function in `updateSystem` to make the entire simulation work with gravity.