hola

Nicolás Herrera
@nicolocodev
github.com/nicolocodev
nicolocodev.wordpress.com
freelance software developer

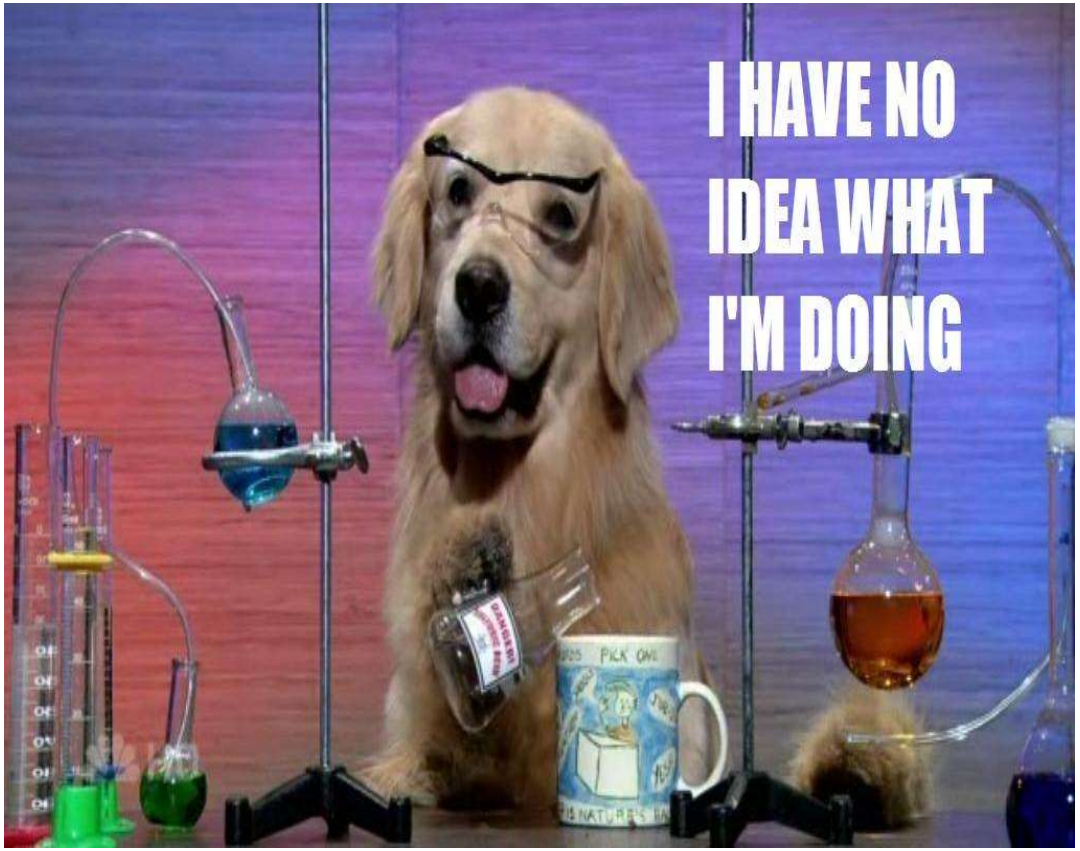
Desarrollo de juegos 2d en haskell

Empezando con haskell

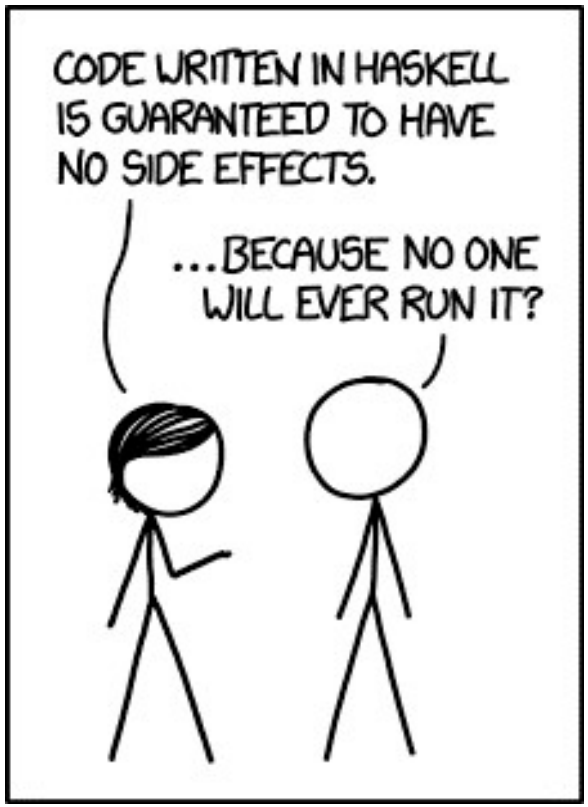
- GHCI
- Tipos de dato algebraico
- Lambda
- Funciones de orden superior
- Composición
- Aplicación Parcial
- Currying
- Monads
- …

Haciendo *algo* con haskell

- Cabal
- Stack
- IDEs
- Web? (mi caso) Yesod, Scotty, Happstack…
- Bases de datos



lolz ☹



http://imgs.xkcd.com/comics/haskell.png

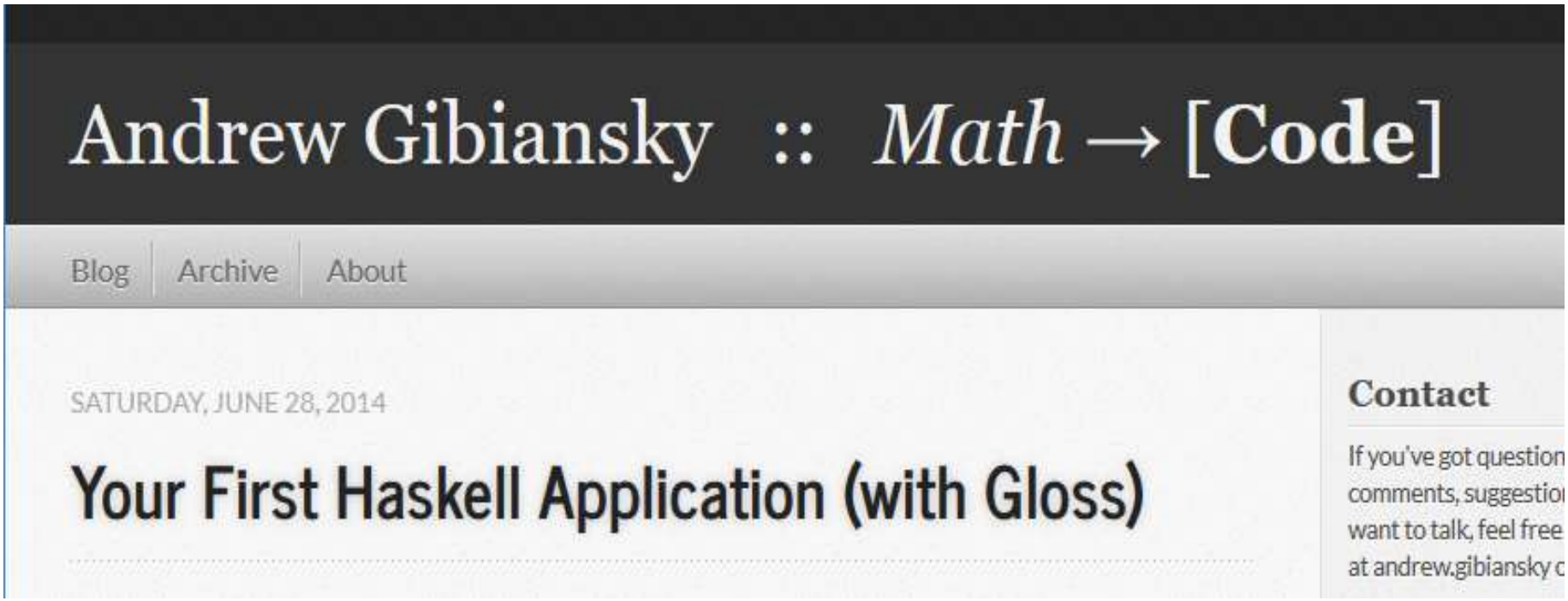
Que pereza el desarrollo web!

Agenda

Quien hace qué

- Gloss: a graphical framework for painless 2D vector graphics, animations and simulations. It has a friendly, functional interface for graphical display.
- OpenGL: graphics hardware API for rendering 2D and 3D vector graphics. Most of the common graphical libraries like Gloss, SDL2 and Qt use OpenGL under the hood.
- GLFW: a portable library which will handle the interaction with the operating system – installing the library on your OS is a prerequisite to using the Haskell bindings GLFW-b

http://andrew.gibiansky.com/blog/haskell/haskell-gloss



Demo

Window: Gloss - GLFW

Animaciones

El loop (pseudocódigo)

loop:

check input (like keyboard, mouse, ...)
change state and process as a result of input
perform any necessary output (render frame)
if no exit signal was given, loop again

El loop

```
loop window = do
        pollEvents
        renderFrame window
        threadDelay 20000
        k <- keyIsPressed window Key'Escape
        if k then return () else loop window
```

threadDelay 20000

the problem is that rendering a frame actually means firing up a chain of events: going from OpenGL to the GPU to the actual rendering of a screen buffer. This only takes a very short amount of time, but some time nevertheless. If the processor (or core) spins this loop as fast as it can, this means the commands to display frames will not have enough time to be executed, pile up and make the process crash.

No se necesitan tantos frames

A human eye will quite happily interpret a succession of images as smooth video if they are displayed at a rate of 24 images a second – which is what is known as the frame rate. Slowing the loop down with 20000 microseconds will get you about 60 frames per second - though that may have to be re-evaluated as the processing in the loop gets more computationally intensive - slower. In any case, it’s more than enough for the rendering to take place in the background.

En gloss


simulate ::
Display
-> Color
-> Int
FPS
-> model
Estado inicial
-> (model -> Picture)
render
-> (ViewPort -> Float -> model -> model)
Update
-> IO ()
-- Defined in 'Graphics.Gloss.Interface.Pure.Simulate'

El estado

State is this cumbersome, yet unavoidable thing: in a game, nothing can happen without at least keeping track of, say, the position of your character, or the state of the world, the score, health etc.

In Haskell you would usually encode the state in a type (or a number of types), and work with either a state monad, or to pass it around as an argument to relevant functions.

Demo

Animaciones

User input

Gloss: play

Demo

Manejando entradas de teclado

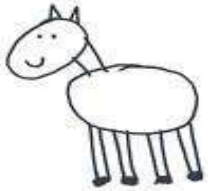
Uniendo todo

HOW TO:
DRAW A HORSE

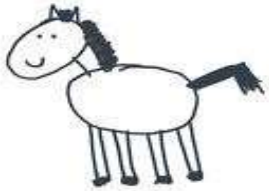
BY VAN OKTOP

(1) DRAW 2 CIRCLES

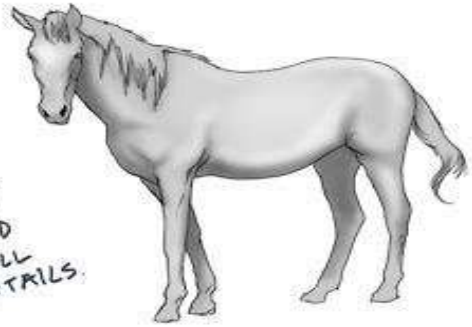
(2) DRAW THE LEGS



(3) DRAW THE FACE



(4) DRAW THE HAIR



(5) ADD SMALL DETAILS.

Estado, Eventos… FRP

Functional Reactive Programming (FRP)

Functional Reactive Programming integrates time flow with events to make state evolve, in a functional, composed way. It can be used for domains like games, robotics, user interfaces, anything where you're looking to combine external inputs with state *over time* to produce a result.

Your state is not a single variable being modified by the loop, but becomes a combination of "Signal"/"Behaviour"/"Streams" over time ("signal" in the rest of this text). You could think about a signal as the history of a component, one long, linear timeline of states. You can combine those signals, and/or make them dependent on external inputs, also to be represented as signals.

En Haskell

For practical purposes, reactive-banana is a library that is more aimed at GUI development, Yampa and Netwire, both in the category of Arrowized FRP, have both been used for games, and Yampa has also been used in the context of music (synthesizers). **I used yet another library called Elerea.**

Why use this library? Well, one reason (for me) is that the author, Patai Gergely, is a game developer, and he open sourced nice examples of his library. A second reason is that I like the apparent simplicity of the concept - **everything can be expressed with Signals and Signal Generators, both of which are monadic**.

Gracias

Nicolás Herrera
@nicolocodev
github.com/nicolocodev
nicolocodev.wordpress.com