

UNIVERSIDAD NACIONAL DE ROSARIO

CIENCIAS DE LA COMPUTACIÓN

ANÁLISIS DE LENGUAJES DE PROGRAMACIÓN

TRABAJO FINAL

AGARSIMU

Autor

MARTÍN VILLAGRA

18 de diciembre de 2015



1. Descripción

AgarSimu es un simulador de una versión simplificada¹ del popular juego agar.io². Provee una interfaz para diseñar inteligencias artificiales³ (o más específicamente bots⁴) que responden a su entorno y la evolución del mismo en el tiempo. A su vez permite ejecutar estos agentes en conjunto dentro de un mundo (instancia del juego).

2. Instalación

Pre-requisito SDL library. Para Ubuntu:

```
$ apt-get install sdl-gfx1.2-dev
```

Una vez instalada la SDL, descargar el simulador e ingresar en la carpeta del mismo:

```
$ git clone https://github.com/mypossum/agarsimu
$ cd agarsimu
```

El simulador se puede instalar usando cabal-install. Se puede instalarlo localmente (para el usuario actual):

```
$ cabal install
```

O bien en una sandbox:

```
$ cabal sandbox init
$ cabal install
```

Dependiendo de cuantos paquetes haya que bajar, esto suele tardar bastante, especialmente si se usa una sandbox.

3. Tutorial de uso

3.1. Ejecutar la simulación

El procedimiento es crear un builder y luego pasárselo a la función *runSimulation*. Un builder crea el estado inicial de la simulación. Este estado contiene las bolas y las AI asociadas a cada una de las bolas.

¹Los jugadores no se pueden dividir o regalar masa.








²<https://en.wikipedia.org/wiki/Agar.io>, <http://agar.io>

³https://en.wikipedia.org/wiki/Artificial_intelligence

⁴https://en.wikipedia.org/wiki/Video_game_bot

A modo introductorio se muestra un uso simple del paquete en el archivo *test/HelloWorld.hs*

3.2. Controles de la simulación

- Para cerrar la simulación presione .
- Para hacer zoom utilice la rueda del mouse .
- Para mover la cámara mantenga presionado el click izquierdo  y mueva el mouse.
- Para aumentar/disminuir la velocidad presione / del teclado numérico o bien /.

3.3. Creación de AIs

```
type Vector = (Double, Double)
type Environment = (Vector, Bola, [Bola])
```

Para nuestros fines, una AI es una función que recibe la información de su entorno y devuelve la dirección en la que moverse. Vamos a definir el entorno como una tripleta (Vector, Bola, [Bola]) que representa respectivamente el tamaño del mapa, información referente a la bola que controla la AI e información de las otras bolas. Si no nos interesa el factor tiempo en nuestro bot, podemos utilizar la función *liftAI*:

```
liftAI :: (Environment -> Vector) -> AI
```

Si queremos tener una especie de memoria o estado, pero sin preocuparnos por el factor tiempo, se provee la función *liftAISF*:

```
— | Tenemos que aportar el estado inicial ahora
liftAISF :: m -> (Environment -> State m Vector) -> AI
```

Para los casos en los que si importe el tiempo, tenemos que recurrir a una definición más poderosa que se verá a continuación.

3.3.1. Wires con Netwire

Las AI son modeladas usando wires, el tipo provisto por la librería de FRP “Netwire”. Se recomienda leer la bibliografía presentada al final para un mejor entendimiento. Un Wire, que es un tipo especial de Arrow autómatas, viene dado por el tipo:

data Wire s e m a b

Un Wire representa una ‘función’ reactiva⁵. Más específicamente, es una función que toma de argumentos:

- Un valor de tipo s que contiene de alguna forma el tiempo transcurrido.
- Una valor de tipo a .

De estos argumentos el Wire:

- O bien devuelve un valor de tipo a o se ‘inhibe’ con un valor de tipo e (error).
- Crea un nuevo Wire s e m a b.

Visto esto podemos pensar a un Wire como:

```
newtype Wire s e m a b =  
  Wire {  
    stepWire :: s -> a -> m (Either e b, Wire e m a b)  
  }
```

Notar como al devolver una nueva instancia de sí mismo, el Wire puede variar su comportamiento a medida que pasa el tiempo. Si bien esta definición parece ser excesivamente complicada, efectivamente vuelve más intuitiva la semántica de los Wires.

En cuanto a la creación de Wires se debe tener a mano la referencia de Netwire⁶, puesto que esta provee numerosas funciones para crear y combinar Wires. Por nombrar algunas:

- $(a < | > b)$ se comporta siempre como a a menos que el mismo se inhiba, en cuyo caso se comporta como b .
- $(a - - > b)$ se comporta como a hasta que se inhiba por primera vez, luego pasa a comportarse siempre como b .

En nuestro caso la variable s es la que aconseja el paquete por defecto para llevar cuenta del tiempo, e será el tipo $()$ ⁷. También se embebió la mónada `Rand` por sí por alguna razón se desea agregar algún comportamiento aleatorio al bot.

Con lo ya dicho podemos mostrar finalmente el tipo de las AI:

⁵Cuyo comportamiento varía con el tiempo.

⁶ mencionada en la bibliografía

⁷La inhibición de la AI es equivalente a dejar de jugar, por lo que no conviene inhibirse en la mayoría de los casos.

```
type Time = NominalDiffTime
type RandomWire = Wire (Timed Time ()) () (Rand StdGen)
type AI = RandomWire Environment Vector
```

4. Esquema de la simulación

La simulación se hizo aprovechando los Wires. En particular los mismos permiten estructurar el sistema en “cajas negras” que se pueden combinar para crear cajas más complejas.

El esquema presentado en el anexo muestra de forma simplificada la estructura de la simulación, el mismo sólo indica el flujo reactivo del sistema (los argumentos estáticos de las funciones no se muestran) por cada paso de la simulación. Cada caja representa un Wire.

5. Organización del código

El código del simulador se encuentra en la carpeta AgarSimu. Se explica a continuación el contenido de cada módulo.

- **Input.hs** Define los wires que procesan los eventos del usuario (mouse y teclado)
- **Render.hs** Provee funciones para renderizar el juego en pantalla.
- **Bola.hs** Contiene funciones que modelan el comportamiento de una bola en el juego. Es decir como varía su posición, velocidad, control de colisiones, etc.
- **Core.hs** Combina los módulos listados arriba para ejecutar el juego.
- **Scene.hs** Define un mini EDSL para crear las escenas (el estado inicial de la simulación).
- **PublicEntities.hs** Define los tipos usados por el usuario final. Se separó del resto para proveer una interfaz más clara.
- **PreFab.hs** Se incluyen funciones útiles que podrían ser útiles para el desarrollo de las AIs.
- **Utils.hs** Funciones de utilidad para los otros módulos. Aquí se definen, por ejemplo, funciones para combinar muchos wires en uno (wires de wires).

6. Detalles de implementación

6.1. Control de velocidad

Como se puede apreciar en el diagrama, la velocidad controla el usuario afecta a la simulación tanto como al renderizado. Más específicamente cuando se aumenta la velocidad:

- Aumenta la velocidad de cada una de las bolas, dentro de la lógica del juego.
- El aumento excesivo de velocidad conduce a errores de precisión dado que la simulación en realidad es un proceso discreto. Para disminuir este efecto, se implementa otra forma de aumentar la velocidad de simulación sin perder precisión. Consiste en no renderizar todos los pasos individuales que produce la simulación.

6.2. Posición

Un detalle interesante que permiten los Wires es expresar la posición de la bola como la integral de la velocidad, su implementación puede verse en *Bola.hs*.

6.3. combine y dynMulticast

Para realizar *aiswire* y *foodGenerator* se tuvieron que crear funciones que toman un conjunto de wires y crean un wire que contiene el comportamiento de todos. Ver su implementación en **Utils.hs**, especialmente sus tipos, que son bastante intuitivos.

7. Bibliografía

- Introducción a FRP
 - Lambda Jam 2015 - Conal Elliott - The Essence and Origins of Functional Reactive Programming, <https://www.youtube.com/watch?v=j3Q32brCUAI>
 - Lambda Jam 2015 - Conal Elliott - A More Elegant Specification for FRP, https://www.youtube.com/watch?v=teRC_Lf61Gw
- Arrows
 - Haskell/Arrow tutorial, https://en.wikibooks.org/wiki/Haskell/Arrow_tutorial

- How does ArrowLoop work? Also, mfix?, <http://stackoverflow.com/questions/6976944/how-does-arrowloop-work-also-mfix>
 - How does this definition of ArrowLoop.loop work?, <http://stackoverflow.com/questions/9856342/how-does-this-definition-of-arrowloop-loop-work>
 - GHC Language Features, Arrow notation, https://downloads.haskell.org/~ghc/7.8.4/docs/html/users_guide/arrow-notation.html
- Wires
- Oliver Charles Blog, Getting Started with Netwire and SDL, <https://ocharles.org.uk/blog/posts/2013-08-01-getting-started-with-netwire-and-sdl.html>
 - Today in Code, Almost a Netwire 5 Tutorial, <http://todayincode.tumblr.com/post/96914679355/almost-a-netwire-5-tutorial>
 - Readme de Netwire, <http://hackage.haskell.org/package/netwire-5.0.1#readme>
 - Referencia de Netwire, hackage.haskell.org/package/netwire-5.0.1
- Juegos de ejemplo
- Oliver Charles Blog, Asteroids & Netwire, <https://ocharles.org.uk/blog/posts/2013-08-18-asteroids-in-netwire.html>
 - Nathan Hüskén Blog, Breakout - Improved and with netwire, <http://jshaskell.blogspot.com.ar/2012/11/breakout-improved-and-with-netwire.html>

Anexo: Esquema de la simulación

