Cacería Monstruosa

La empresa Cuadrado Enix se encuentra al borde de la quiebra y nos contrató para crear un juego de fantasía con un fuerte énfasis en su historia. Este va a ser un juego de rol y, como esperaríamos, los personajes van a poder interactuar de diferentes formas con el mundo. El objetivo principal es que los personajes deben ir a cazar monstruos a diferentes mapas, para subir de nivel y progresar en el juego.

Los personajes

De un personaje nos interesa saber:

- Cuál es su nivel, lo que indica cuán bueno es. Se calcula en función de cuánta es su experiencia, como la experiencia al cuadrado sobre la experiencia más uno, redondeado hacia arriba.
- Su experiencia, que es un valor inicial arbitrario, se puede ir modificando a medida que transcurre el juego.
- Su capacidad para la caza, que se calcula como la fuerza básica que tiene el personaje modificado por el elemento que el personaje tiene.

Algunos elementos que existen en el mundo y que pueden ser llevados por un personaje, son:

```
espadaOxidada = (1.2*)
katanaFilosa  = (10+).(0.9*)
sableLambdico cm = ((1+cm/100)*)
redParadigmatica = sqrt
baculoDuplicador x= x* 2
espadaMaldita = espadaOxidada.sableLambdico 89
```

Modelar algunos personajes de ejemplo

Alquimistas

En este tipo de juegos suele haber alquimistas que (generalmente) mejoran la capacidad de caza de los personajes, alterando los elementos que se utilizan. Hay muchos tipos, pero algunos de los más usuales son:

- Los **aprendices**, que fusionan el elemento del cazador con un químico secreto que duplica su efecto.
- Los **maestros alquimistas**, que hacen lo mismo que los aprendices, pero además aplican al elemento un 10% extra de capacidad, tantas veces como años de oficio tenga en su profesión
- Los **estafadores**, que reemplazan el elemento del personaje por un elemento que no hace nada.

- Inventar un nuevo tipo de alquimista que, similar a los anteriores, haga algo con el elemento del personaje. Debe utilizarse alguna expresión lambda, de una forma que no sea trivial.

Lo que se necesita saber es, de una lista de alquimistas:

1. Todos aquellos que hacen que la capacidad del cazador sea superior a un valor dado.
2. Si todos le convienen al personaje

Monstruos

De cada monstruo se conoce su especie, su resistencia y una lista de habilidades, donde para cada una hay una descripción y un tipo, que puede ser magia, física, naturaleza, entre otras.

1. Nos interesa saber si un monstruo es agresivo. Eso sucede cuando tiene más habilidades ofensivas (tipo mágicas o físicas) que de soporte (las restantes) y además su resistencia es mayor a 0. Los monstruos de las especies animal y chocobo nunca son agresivos.

A la Caza!!

1. Averiguar si un personaje le gana a un monstruo. Le gana si su capacidad de caza es mayor que la resistencia del monstruo.
2. Se quiere conocer el estado del personaje luego de explorar un mapa y pelear con todos los monstruos agresivos que haya. Cada vez que le gana a un monstruo su experiencia aumenta en 100 puntos, si pierde disminuye en 50 y su objeto pierde un 10% de su poder, pero en todos los casos va por el próximo enemigo del mapa.
3. Ver si hay algún monstruo en el mapa al que, pese a haber recurrido a un alquimista amigo, el cazador no pueda vencer.

Para pensar...

1. Explicar dónde y para qué se utilizaron los conceptos de aplicación parcial y composición.
2. Analizar qué cosas podrían suceder si la lista de monstruos de un mapa fuera infinita. Justificar conceptualmente.

Importante: Mostrar ejemplos de uso de las principales funciones

Posible Solucion

```
import Text.Show.Functions

data Personaje = UnPersonaje
  {nombre :: String,
  fuerza :: Float,
  experiencia :: Float,
  elemento :: Elemento} deriving Show

type Elemento = Float -> Float

marco = UnPersonaje "marco" 15 10 katanaFilosa
paul  = UnPersonaje "paul" 17 6 baculoDuplicador
lucas = UnPersonaje "lucas" 7 69 espadaMaldita

espadaOxidada = (1.2*)
katanaFilosa  = (10+).(0.9*)
sableLambdico cm = ((1+cm/100)*)
redParadigmatica = sqrt
baculoDuplicador x= x * 2
espadaMaldita = espadaOxidada.sableLambdico 89

nivel (UnPersonaje _ _ experiencia _)  = ceiling ( experiencia * experiencia / (experiencia + 1))
--nivel personaje  = ceiling ( xp * xp / (xp + 1))
--  where xp = experiencia personaje

capacidad (UnPersonaje _ _ experiencia elemento) = elemento experiencia
--capacidad personaje = (elemento personaje) (experiencia personaje)

type Alquimista = Personaje -> EstiloAlquimista-> Personaje
type EstiloAlquimista = Elemento -> Elemento

alquimista estilo  personaje = personaje { elemento = estilo (elemento personaje) }

estiloAprendiz elemento = (2*).elemento
--interpretando que "duplicar" consiste en aplicar dos veces el elemento
--estiloAprendiz elemento = elemento.elemento

estiloMaestro anios elemento = coeficienteAntiguedad anios (estiloAprendiz elemento)

coeficienteAntiguedad 0 elemento = elemento
coeficienteAntiguedad anios elemento = coeficienteAntiguedad (anios-1) ((1.1*).elemento)
--coeficienteAntiguedad anios elemento = (1.1*).coeficienteAntiguedad (anios-1) elemento

--coeficienteAntiguedad 0 = id
--coeficienteAntiguedad anios = (*1.1).coeficienteAntiguedad (anios-1)

estiloEstafador = id
--estiloEstafador x = x
```

```
--estiloEstafador = (1*)

estiloInventado = (\x -> x * 3)

--alterarElemento:: Elemento -> Personaje -> Personaje
alterarElemento nuevoElemento personaje = personaje { elemento = nuevoElemento.elemento personaje }

{-
--Variante sin usar estilo, sino directamente alquimistas que modifican personajes

type Alquimista = Personaje -> Personaje

aprendiz = alterarElemento (2*)
--aprendiz personaje = alterarElemento (2*) personaje

maestro anios = alterarElemento (coeficienteAntiguedad anios).aprendiz

estafador personaje = personaje { elemento = id }

inventado personaje = personaje { elemento = (\x -> x * 3) }

-}

capacidadSuperiorA valor personaje alquimista = ((>valor).capacidad.alquimista) personaje
--capacidadSuperiorA valor personaje alquimista = capacidad (alquimista personaje) >valor

capacidadesSuperioresA valor personaje alquimistas = filter (capacidadSuperiorA valor personaje) alquimistas
convienenTodos alquimistas personaje = all (capacidadSuperiorA (capacidad personaje) personaje) alquimistas


data Monstruo = UnMonstruo
  {especie :: String,
  resistencia :: Float,
  habilidades :: [Habilidad]} deriving Show

--habilidades(descripcion,tipo)
type Habilidad = (String, String)
tipo = snd
descripcion = fst

esAgresivo (UnMonstruo especie resistencia habilidades) = mayoriaOfensivas habilidades && especieAgresiva especie && resistencia > 0
-- esAgresivo monstruo = mayoriaOfensivas (habilidades monstruo) && especieAgresiva (especie monstruo) && resistencia monstruo > 0

mayoriaOfensivas habilidades =  length (filter (esAtaque.tipo) habilidades) > div (length habilidades) 2
```

```
esAtaque "magia" = True
esAtaque "fisica" = True
esAtaque _ = False
--esAtaque tipo  = tipo == "magia" || tipo == "magia"

especieAgresiva especie = elem especie ["animal", "chocobo"]

gana personaje monstruo = capacidad personaje > resistencia monstruo

cambiarExperiencia variacion personaje = personaje {experiencia = experiencia personaje + variacion}

pelear personaje monstruo
   | gana personaje monstruo = cambiarExperiencia 100 personaje
   | otherwise =  alterarElemento (0.9*) (cambiarExperiencia (-50) personaje)

pelearConTodos personaje monstruos = foldl pelear personaje monstruos

hayInvencible personaje alquimista monstruos= all (gana (alquimista personaje)) monstruos
```