HASKELLMÓN

Contenido

[Pokémon 2](#_Toc156232355)

[Pokémons: 2](#_Toc156232356)

[Ataques: 2](#_Toc156232357)

[Tipos: 2](#_Toc156232358)

[Código: 3](#_Toc156232359)

[ESTRUCTURA DEL PROYECTO 3](#_Toc156232360)

[PokemonData.hs 4](#_Toc156232361)

[Pokémon.hs 5](#_Toc156232362)

[Tipo.hs 6](#_Toc156232363)

[Daño.hs 7](#_Toc156232364)

[Parser.hs 8](#_Toc156232365)

[Modo de juego: 9](#_Toc156232366)

Trabajo realizado por:

* SEBASTIÁN BUZÓN CORDÓN (sebbuzcor / sebbuzcor@alum.us.es)
* SILVIA BUSTILLOS MORENO silbusmor (silbusmor / silbusmor@alum.us.es)

# Pokémon

El proyecto consistirá en simular una batalla Pokémon, la cual consiste en combates por turnos donde ambos jugadores eligen los ataques correspondientes. Los combates se conforman de varias cosas, los Pokémon, ataques y sus correspondientes tipos. Ganará el jugador que debilite al oponente primero.

Para hacer funcionar el juego, hay que correr el módulo **GameManager.hs**

### Pokémons:

Estarán conformados por un nombre, un tipo o dos y una lista de ataque que podrá aprender aleatoriamente hasta tener 4 tipos de ataques.

### Ataques:

Estarán conformados por un nombre, un tipo y la cantidad de daño que realizan al oponente. Aunque este se verá afectado por el **tipo** del ataque y por el **tipo** tanto del oponente como del Pokémon que realiza el ataque.

### Tipos:

Sin duda uno de los apartados más interesantes y donde se centra toda l jugabilidad de Pokémon, apartando así el azar y centrando el juego en pura estrategia, no obstante, el **RNG** (**R**andom **N**umber **G**enerator) sigue siendo un factor clave en el juego…

Los tipos se basan en elementos naturales como Eléctrico, Fuego, Agua, Planta, Roca, etc. La relación entre los elementos es un poco como un piedras papel y tijeras, pero quizá algo más complicado… Para poner un ejemplo es necesario implementar la famosa “**Tabla de Tipos**”, una tabla algo complicada de entender donde se almacenan todas las debilidades, fortalezas e inmunidades de todos los tipos Pokémon.

Pulsa [aquí](https://pokemew20.files.wordpress.com/2018/09/8e65aee644cb571585bf1f985d8facd3.png) para visualizar la Tabla de Tipos.

## Código:

Una vez definido el juego, vamos a desglosar el código para entender cómo crear paso a paso un Pokémon primitivo y funcional.

Primero que nada, vamos a tener que minimizar un poco las estadísticas que existen en el juego original, ya que hay muchísimas más variables en juego que solo el daño del ataque, pero en nuestro miniPokémon de Haskell vamos a tener solo en cuenta el poder del ataque, la vida de los Pokémons y los respectivos tipos. Ya que en el juego original existen muchísimos más atributos como el Ataque físico y el Ataque especial, la Velocidad, etc.

### ESTRUCTURA DEL PROYECTO

Tenemos organizado los diversos ficheros en 4 carpetas:

Captura de pantalla de computadora

Descripción generada automáticamente

**Data**: módulos necesarios para trabajar con los tipos de datos.

* **PokemonData.hs**: Módulo donde se definen todos los tipos Abstractos.
* **Pokémon.hs**: Módulo donde se tratan los datos **Pokémon** y **Habilidad**.
* **Tipo.hs:** Módulo donde se tratan los datos **Tipo**.
* **Potion.hs**: Módulo donde se tratan los datos **Pociones** (Pila).

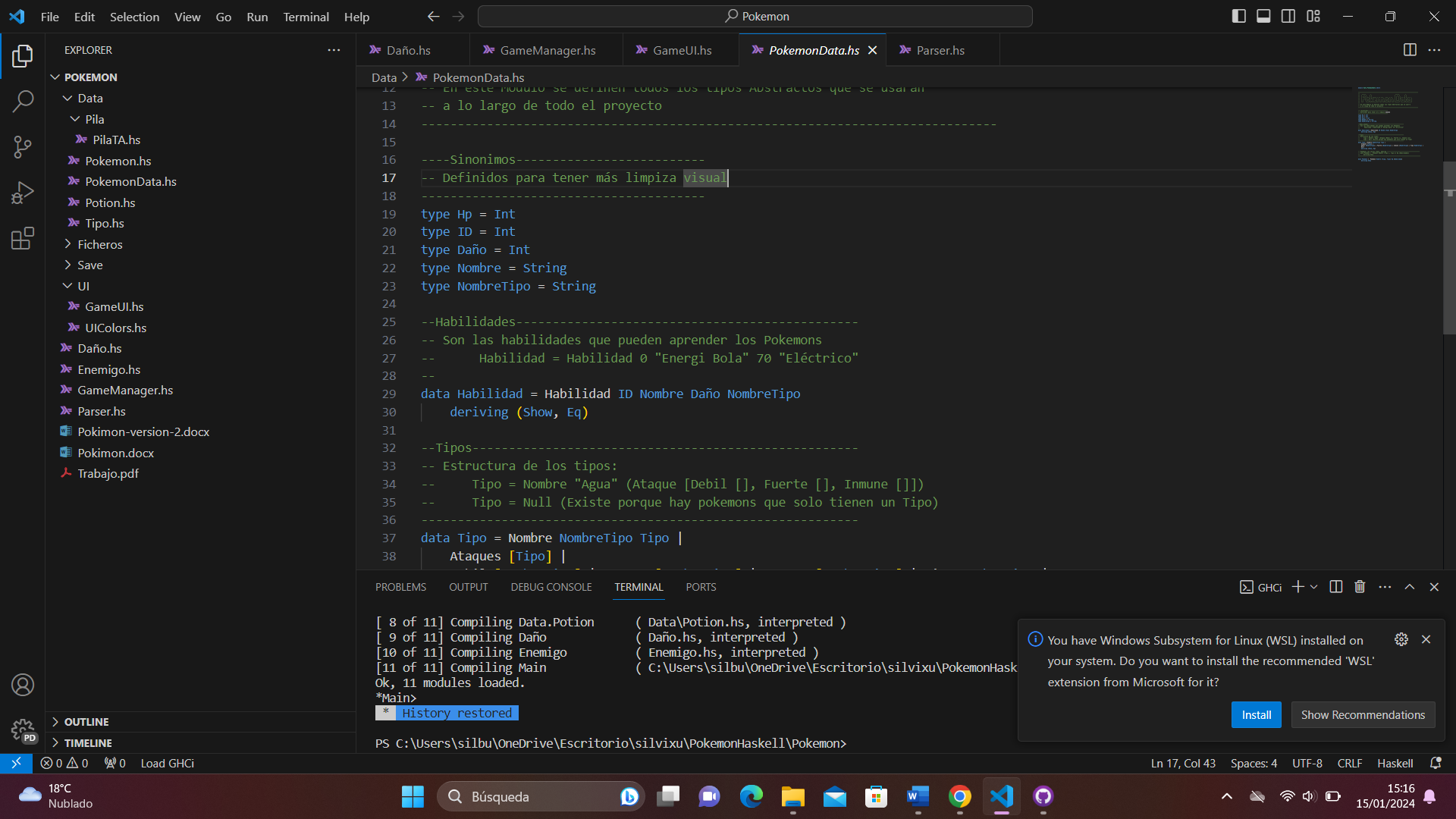
**Ficheros:** ficheros de texto e imágenes necesarias para la lectura de datos.

**Save**: ficheros donde se almacenarán las partidas guardadas anteriormente.

**UI**: módulos creados para la Interfaz de Usuario:

* **UIColors.hs**: creado para almacenar colores.
* **GameUI.hs**: creado para dar diseño y forma al juego.

Vamos a centrarnos primero en la carpeta **Data**, ya que es donde se almacenan los módulos base para la creación de las funciones.



Podemos observar que hemos usado el **Tipo de Dato Abstracto Pila**, y hemos creado 4 ficheros para trabajar las funciones necesarias.

### PokemonData.hs

El módulo **PokemonData** está creado para definir los **Tipos de Datos** necesarios para la creación de un Pokemon:

#### SINÓNIMOS

type Hp = Int

type ID = Int

type Daño = Int

type Nombre = String

type NombreTipo = String

#### TIPOS DE DATOS

--Habilidad--------------------------------------------------------------

data Habilidad = Habilidad ID Nombre Daño NombreTipo

    deriving (Show, Eq)

--Tipo-------------------------------------------------------------------

data Tipo = Nombre NombreTipo Tipo |

    Ataques [Tipo] |

    Debil [NombreTipo] | Fuerte [NombreTipo] | Inmune [NombreTipo] | Tipo NombreTipo |

    Null

    deriving (Show, Eq)

--Pokemon----------------------------------------------------------------

data Pokemon =  Pokemon Nombre (Tipo, Tipo) Hp [Habilidad]

    deriving Show

Definidos estos tipos, ya podemos crear las funciones necesarias para la batalla.

Las funciones básicas están creadas en los módulos **Pokemon.hs**, **Tipo.hs** y **Potion.hs**.

Vamos a explicar brevemente qué se trabaja en cada módulo y poner algunas funciones de ejemplo para ver la relación que tiene en la asignatura:

### Pokémon.hs

El módulo **Pokemon** está creado para el tratamiento de los tipos de datos Pokémon y Habilidad. Veamos algunas funciones de este módulo:

#### FUNCIONES POR PATRONES

setPokemonVida :: Pokemon -> Int -> Pokemon

setPokemonVida (Pokemon n t hp h) daño = Pokemon n t (hp-daño) h

getPokemonHabilidades :: Pokemon -> [Habilidad]

getPokemonHabilidades (Pokemon \_ \_ \_ xs) = xs

#### LISTAS POR COMPRENSIÓN

getPokemonHabilidadPorNombre :: Nombre -> [Habilidad] -> Habilidad

getPokemonHabilidadPorNombre n hs = head [ h | h@(Habilidad \_ nom \_ \_) <- hs, n == nom]

getHabilidadPorID :: [Habilidad] -> Int -> Habilidad

getHabilidadPorID habilidades idPok = head [h | h@(Habilidad id \_ \_ \_) <- habilidades, id == idPok]

#### FUNCIONES DE ORDEN SUPERIOR

getPokemonNombreHabilidades :: Pokemon -> [String]

getPokemonNombreHabilidades (Pokemon \_ \_ \_ xs) = foldr (\x ac -> getNombreHabilidades x : ac) [] xs

### Tipo.hs

El módulo **Tipo** está creado para el tratamiento del tipo de dato Tipo. Veamos algunas funciones de este módulo:

getNombreTipo :: Tipo -> String

getNombreTipo (Nombre x \_) = x

getNombreTipo Null = ""

getNombreTipo \_ = error "Error al extraer el nombre del tipo!"

#### USO DE GUARDAS

getTipoPorNombre :: [Tipo] -> Nombre -> Tipo

getTipoPorNombre [] n = error $ "No se ha encontrado el Tipo: " ++ n

getTipoPorNombre ((Nombre n t):tipos) nombre

    | nombre == "null" = Null

    | n == nombre = Nombre n t

    | otherwise = getTipoPorNombre tipos nombre

#### Potion.hs

El módulo **Potion** está creado para el tratamiento del tipo de dato Pocion (sinónimo de Pila Int). Veamos algunas funciones de este módulo:

type Pociones = Pila Int

usarPocion :: Pociones -> (Int, Pociones)

usarPocion p

    | esVacia p = (0, vacia)

    | otherwise = (cima p, desapila p)

parsePotion :: String -> Pociones

parsePotion s = parseo lista

    where

        lista = splitText (=='|') s

        parseo (x:xs)

            | x == "-" = vacia

            | otherwise = apila (read x) (parseo xs)

        parseo [] = vacia

Ahora, fuera de la carpeta Data, tenemos los 4 ficheros fundamentales para crear la batalla entre los Pokémon.

Captura de pantalla de computadora

Descripción generada automáticamente

### Daño.hs

El módulo **Daño** está creado para el cálculo de la eficacia de los ataques entre los Pokémon (siguiendo la [**Tabla de Tipos**](https://pokemew20.files.wordpress.com/2018/09/8e65aee644cb571585bf1f985d8facd3.png) anteriormente mencionada). Veamos algunas funciones de este módulo:

esCritico :: IO Bool

esCritico = do

    rand <- getNumRandomInterval 0 100

    return (rand < 10)

esSTAB :: Pokemon -> Habilidad -> Double

esSTAB (Pokemon \_ (tipo1, tipo2) \_ \_) habilidad

    | tipoHabilidad `elem` [getNombreTipo tipo1, getNombreTipo tipo2] = 1.5

    | otherwise = 1.0

    where tipoHabilidad :: String

          tipoHabilidad = getTipoHabilidad habilidad

#### LLAMADAS RECURSIVAS

getEficaciaAtaque :: Tipo -> Tipo -> Double

getEficaciaAtaque (Nombre n \_) (Debil xs)

    | n `elem` xs = 0.5

    | otherwise = 1

getEficaciaAtaque (Nombre n \_) (Fuerte xs)

    | n `elem` xs = 2

    | otherwise = 1

getEficaciaAtaque (Nombre n \_) (Inmune xs)

    | n `elem` xs = 0

    | otherwise = 1

getEficaciaAtaque tPokemon (Nombre \_ ataque) = getEficaciaAtaque tPokemon ataque

getEficaciaAtaque tPokemon (Ataques ts) = product [getEficaciaAtaque tPokemon t | t <- ts]

esEficaz :: Pokemon -> Tipo -> Double

esEficaz (Pokemon \_ (tipo1, tipo2) \_ \_) tipoH

    | esNull tipo1 && esNull tipo2 = error $ setColor red "El pokemon carece de tipo"

    | esNull tipo2 = getEficaciaAtaque tipo1 tipoH

    | esNull tipo1 = getEficaciaAtaque tipo2 tipoH

    | otherwise = getEficaciaAtaque tipo1 tipoH \* getEficaciaAtaque tipo2 tipoH

hacerElDaño :: (Pokemon,Pokemon) -> (Habilidad,Tipo) -> Bool -> Bool -> (String,Pokemon)

hacerElDaño (pObjetivo,pAtacante) (h,th) crit turno = (comentario,nuevoPokemonObjetivo)

        where

            comentario = generarComentario pObjetivo h (e,c) daño turno

            b = esSTAB pAtacante h

            e = esEficaz pObjetivo th

            p = obtenerPotenciaHabilidad h

            c | crit = 2

              | otherwise = 1

            daño = round (b \* e \* (((1.2\*p)/25) + 2) \* c \* 2) :: Int

            nuevoPokemonObjetivo = setPokemonVida pObjetivo daño ::Pokemon

### Parser.hs

El módulo **Parser** está creado para el parseo de los datos de los ficheros. Veamos por ejemplo cómo sería el parseado de Tipos:

parseoDebilFuerteInmune :: String -> ([String],[String],[String])

parseoDebilFuerteInmune s = (debilidades, fortalezas, inmunidades)

    where

        lista = drop 1 (splitText (=='\t') s)

        debilidades = (words.head) lista

        fortalezas = words (lista!!1)

        inmunidades = words (lista!!2)

parseoUnSoloTipo :: String -> Tipo

parseoUnSoloTipo atk = Nombre nombre attak

    where

        (atk\_d, atk\_f, atk\_i) = parseoDebilFuerteInmune atk

        attak = Ataques [Debil atk\_d, Fuerte atk\_f, Inmune atk\_i]

        nombre = head (splitText (=='\t') atk)

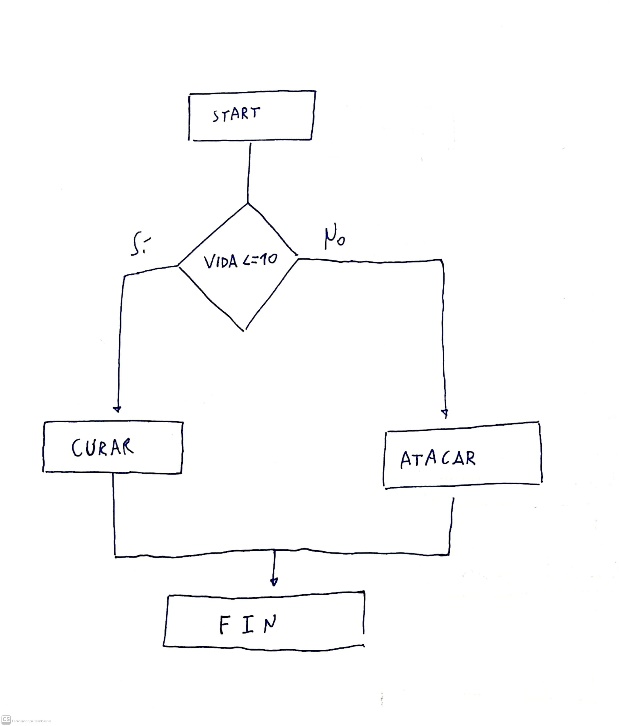
parsearTipos :: [String]  -> [Tipo]

parsearTipos = foldr(\x ac -> parseoUnSoloTipo x : ac) []

## Enemigo y Game Manager:

### Enemigo:

La “IA” del enemigo no es más que una máquina de estados muy sencillita, para ver cómo funciona más a fondo el módulo “**Enemigo.hs**”, es importante ver el código:



### Game Manager:

El Game Manager es lo que hace todo el juego funcionar y donde se fusiona todo. Verlo detalladamente en este PDF llenaría las 10 páginas por si solo, por lo que hemos recurrido a mostrar una máquina de estados sobre su funcionamiento general:

Diagrama, Esquemático

Descripción generada automáticamente

## Modo de juego:

Nuestro juego es todo a formato escrito (ojo con escribir correctamente). Al empezar tu partida nos encontramos con el menú principal, donde podemos elegir empezar la partida (**Start**), cargar una partida previamente guardada (**Load**) o salir (**Exit**).

Captura de pantalla de computadora

Descripción generada automáticamente

¡Empecemos una partida! Escribimos “**Start**”, después escribimos “**Ok**” para aceptar que queremos jugar y generar una partida aleatoria, se nos pedirá que introduzcamos una serie de dígitos para comenzar una partida aleatoria. En este caso he elegido **17-17**:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

¡Vaya! Nuestro Pokémon es Mismagius, y se va a enfrentar a Sinistea, ambos de Tipo Fantasma. Aquí podemos elegir si queremos atacar (**Attack**), usar alguna poción para recuperar puntos de vida (**Potions**) o guardar la partida para jugar más adelante (**Save**). Vamos a atacar (recuerda escribir correctamente la acción)

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Tras escribir “Attack” nos aparecen estos 4 ataques aleatorios para atacar al enemigo, (recuerda que [aquí](https://pokemew20.files.wordpress.com/2018/09/8e65aee644cb571585bf1f985d8facd3.png) puedes ver la efectividad de los ataques).

Captura de pantalla de computadora

Descripción generada automáticamente

¡Qué bien!, tenemos ataques de tipo Fantasma, que sabemos que nuestro enemigo es débil ante él. Podemos elegir escribir el ataque o volver atrás (**Back**).

Escribamos un ataque, vamos a elegir **Shadow Claw** que sabemos que será **eficaz**.

Podemos observar cómo ha bajado la vida de nuestro enemigo, pasando de 40 a 8 puntos.

Texto

Descripción generada automáticamente

Nuestro enemigo también tiene la posibilidad de usar pociones para recuperar puntos de vida.

Texto

Descripción generada automáticamente

Así que habrá que seguir atacando hasta conseguir dejarlo sin puntos de vida.

Texto

Descripción generada automáticamente

Y así finalmente, tras un par de ataques, hemos conseguido derrotar a nuestro enemigo. Aquí es donde se acaba la partida.

Pantalla de computadora con fondo negro

Descripción generada automáticamente con confianza media

Pero no se acaba el juego, ya que podemos probar cientos de combinaciones diferentes de Pokémon y ataques distintos, y probar cómo reaccionan a los ataques, si son fuertes, débiles, inmunes, etc.

Algunos combates interesantes son este que hemos visto con la **seed**: 17-17, pero que tal algo más interesante, probemos la **seed** **1444-1444** (recuerda que la **seed** ha de ser cómo máximo de 16 dígitos, 8 por cada lado del “-“). ¡Vaya! Este combate es algo más interesante. Tenemos un ataque de tipo **Veneno** contra un Pokemon de tipo **Acero,** haz la prueba ¿Qué pasa si le hacemos este ataque al Enemigo?