

Informe App3

El Bosque de las Runas Mágicas

|  |  |
| --- | --- |
| Integrantes | Matías Astraín  Diego Calvo |
| Asignatura | Lenguajes y Paradigmas de Programación |
| Sección | 1 |
| Profesor | María Loreto Arriagada |
| Fecha de entrega | 10-06-2025 |

# Introducción

En este trabajo se realizó un código usando el lenguaje Haskell, que consiste en simular un mago que quiere encontrar el camino óptimo por un camino encantado, el cual está lleno de runas las cuales pueden modificar la energía del mago de forma positiva o negativa.

# Objetivos

1. Comprender el paradigma de Programación Funcional
2. Practicar herramientas de apoyo a la programación: GIT (fork, pull requests, commits balanceados), debuggers, etc.

# Desarrollo

Para la solución del problema, como grupo decidimos utilizar tres archivos. Uno se llama *Bosque.hs,* el cual se encarga de verificar que se cumplan todas las restricciones que hay en el bosque.

Luego, tenemos el archivo *Caminos.hs*, el cual se encarga de definir lo que hace cada movimiento que podría hacer el mago, y modifica su energía dependiendo de la celda donde se vaya moviendo.

Por último, tenemos el archivo *main.hs*, encargado de generar la matriz que representa el bosque de runas según las dimensiones que ingresa el usuario, quien también puede elegir la energía inicial que tendrá el mago. Después de esto, el archivo ejecutará el juego.

A continuación se especificarán las utilidades que tiene cada función que se encuentran dentro de los 3 archivos ya mencionados.

### Bosque.hs

#### obtenerValor

* Obtiene el valor de una celda dada la posición.

#### dentroMatriz

* Verifica si una posición está dentro de los límites válidos de la matriz.

#### esTrampa

* Determina si una celda es una trampa, es decir, si la celda disminuye la energía del mago.

#### esDiagonal

* Verifica si el último movimiento fue en diagonal abajo-derecha.

#### destino

* Retorna la posición final de la matriz (abajo a la derecha).

### Caminos.hs

#### movimientosPosibles

* Tiene los movimientos válidos que podría hacer el mago (abajo, derecha, diagonal, arriba, izquierda).

#### sumarPosiciones

* Suma dos posiciones. Sirve para moverse en la matriz.

#### caminoConMaximaEnergia

* Busca el camino con mayor energía restante

#### buscarUsandoMemoizaciónYPoda

* Búsqueda recursiva de todos los caminos posibles desde la posición actual.
* A pesar de que la energía inicial es constante, las técnicas de memoización y poda anticipada mejoran el resultado final. Esto se debe a que permiten al algoritmo reutilizar resultados parciales eficientes y descartar caminos subóptimos, permitiendo alcanzar rutas más beneficiosas energéticamente que estaban disponibles desde el inicio pero no eran exploradas completamente.

#### procesarMovimiento

* Aplica un movimiento posible desde la posición actual, calcula el nuevo estado (camino, energía, tablas de memoización y poda), y acumula los resultados de todos los movimientos posibles.

#### seleccionarCaminoConMasEnergia

* Selecciona el camino que deja más energía al final.

### main.hs

#### ejecutarJuego

* Ejecuta el cálculo del camino con matriz y energía dados.

#### pedirEnergia

* Pide al usuario la energía inicial.

#### pedirDimensiones

* Pide las dimensiones (fila y columna) de la matriz al usuario.

#### pedirMatrizManual

* Permite ingresar la matriz fila por fila desde la consola.

#### matrizAleatoria

* Genera una matriz aleatoria con valores entre -5 y 5.

#### guardarMatriz

* Guarda una matriz en un archivo de texto.

#### formatearMatriz

* Convierte una matriz a texto plano (para guardar en un archivo).

# Conclusión

Al principio, como grupo, se hizo complicado el hecho de acostumbrarse a realizar código en Haskell, ya que es un lenguaje con una forma de escribir muy distinto a lo que hemos hecho ante. Pero, a medida que fuimos avanzando en la solución de la tarea, nos pudimos adaptar al nuevo lenguaje.

Unos de los desafíos que tuvimos fue que, durante un periodo, nuestro código se demoraba demasiado en ejecutar el juego (varios minutos), pero, luego de realizar varias correcciones, pudimos lograr que el código se ejecutara en menos de un segundo.

Otro desafío que tuvimos es que, una vez que el código realizaba correctamente lo que tenía que lograr, nos aparecían varios warnings. Afortunadamente pudimos corregir esos warnings con la ayuda de la inteligencia artificial.