INFO188 Tarea 1: Mini Adventure X

Estudiantes: Patricio Lobos, Benjamín Parra, Pascal Salinas, Rodrigo Erlandsen.

Profesor: Cristóbal A. Navarro

Ayudante: Alejandro Villagrán

Introducción:

En esta tarea se nos encargó el desarrollo de un videojuego en Haskell, un lenguaje de programación orientado al paradigma funcional. Para desarrollar este videojuego tratamos de hacer el mejor uso posible del paradigma funcional y sus abstracciones, en este documento usted podrá encontrar detalladamente nuestro proceso de desarrollo y una explicación profunda del funcionamiento del programa.

Descripción del videojuego:

Mini Adventure X (MAX para abreviar) es un videojuego hecho en el lenguaje haskell usando el paradigma funcional. En esta tarea, el juego está diseñado para ser jugado desde terminal, consistiendo en explorar un mapa 2D sencillo para encontrar el tesoro marcado con “X”.

Al ejecutar el juego comienza con un menú que le da una breve introducción al jugador y le da contexto a la aventura.

Este mundo se genera usando una grilla de n x n celdas, cada celda puede ser caminable (símbolo “ “), obstáculo (símbolo “L”), o bien lava (símbolo “$”). A continuación listamos las interacciones correspondientes a estos elementos:

- No es posible atravesar obstáculos, estos tienden a formar murallas
- Si el jugador cae en la lava, muere. La lava tiende a formar piscinas

Al generar el mundo el juego inicia posicionando al personaje (carácter “@”) en la posición (0,0) para mayor comodidad del jugador, por otra parte, el tesoro (“X”) y las runas(λ) se generan en celdas caminables aleatorias.

El jugador puede moverse con las teclas: W; A; S; D; hacía arriba, izquierda, abajo y derecha respectivamente, también puede utilizar “R” que regenera el mapa manteniendo al jugador, a X y a λ sin verse afectados.

El jugador en todo momento puede presionar la tecla “H” para ver los controles.

Las runas le sumarán puntos al puntaje total y además, cuando se recolectan todas se podrán destruir los obstáculos. el juego finaliza cuando el personaje llega al tesoro.

Funcionamiento del programa:

Para poder jugar, primero uno debe compilarlo con el comando make y después ejecutarlo en consola de la siguiente forma:

./max <n> <semilla>

n siendo el tamaño de la grilla que conforma el mapa, la cual será de tamaño n*n y n debe ser mayor a 4. La semilla se usa para los generadores random.

También se consideraron condiciones de borde tales como:

- El tesoro no se podrá generar sobre otros objetos.
- Las runas no se generarán encima del cofre o el jugador

Para crear el juego se utilizaron nuevos tipos de datos, estos siendo:

1. Celda: Un conjunto de estados en los que puede estar una casilla del mapa.
2. Mapa: Estructura que almacena celdas
3. Action: Estructura para recibir las entradas.
4. Score: Estructura que instancia functor para poder controlar la puntuación del jugador.

La función main() se encarga de generar el mapa siguiendo las restricciones mencionadas anteriormente, para esto se usaron diversas funciones que serán explicadas más adelante.

El mapa es generado con la función generarMapa(), la cual crea casilla a casilla cada celda, comprobando si debe ser obstáculo, lava, runa o camino, con ayuda de la función generarFilas().

Para saber que posiciones corresponden a lava y obstáculos se generan dos listas de tuplas usando la función genChunk(), una para cada uno, que contienen las posiciones en las que estarán ubicados en el mapa. La cantidad de cada uno está dada por una fórmula matemática, para los pozos $n^2 * 0.06$ y para los obstáculos es el resultado anterior $* 3$. Para dar forma a estas posiciones se usan las funciones chunksLava(), chunksLavaSmall(), chunksObstaculos(), las cuales indican las formas que tendrán las generaciones que hace genChunk().

Una vez generado el mapa se llama a la función loop, la cual se encarga de revisar el estado del juego. En caso de movimiento vuelve a generar el mapa con las

nuevas posiciones; en caso de choque con un muro mantiene todo igual; En caso de lava llama a la función deadMessage() que genera la pantalla de muerte; En caso de conseguir una runa, suma puntaje; En caso de conseguir todas las runas, habilita la opción de destruir paredes y si llega al tesoro invoca la función que imprime la pantalla de victoria.

El puntaje final se calcula de la siguiente manera:

Puntaje = $log(n) * 2^k$; donde k es la cantidad de runas obtenidas y n es el tamaño del mapa

Al final de este documento podrá encontrar un enlace al repositorio Git-hub donde almacenamos el proyecto.

Desafíos del paradigma funcional:

Al carecer de variables y por ende, de los ciclos presentes en otros paradigmas, cosas como la generación de los distintos tipos de celda requirió pensarlo de manera bastante distinta, siendo lo que más tiempo nos tomó, sin embargo, esta línea de pensamiento nos permitió desarrollar soluciones más “elegantes” a nuestro parecer. Soluciones como la generación del mapa por capas en la cual, como se explicó anteriormente, generamos las murallas y piscinas de lava fuera del mapa para luego crear este y evitar que el programa deba rehacerlo cada vez que se modifica una celda.

En comparación, este desarrollo nos llevó a largas sesiones de discusión al contrario que otros paradigmas donde el mayor tiempo se lo lleva la implementación del código.

A pesar de esto, consideramos que el uso de este paradigma permite un código más legible y fácil de analizar, con soluciones que nos requirieron pensar fuera de nuestra zona de confort.

Conclusión:

Consideramos que el desarrollo de este videojuego fue un gran desafío que puso a prueba nuestro manejo del paradigma funcional y nos llevó a largas sesiones de pensamiento analítico, aprendimos mucho de esta experiencia y consideramos que desarrollamos un buen producto con las limitaciones presentes.

Repositorio: https://github.com/SBlader/MAX