**UNIVERSIDAD SIMÓN BOLÍVAR**

**DEPARTAMENTO DE COMPUTACIÓN Y TECNOLOGÍA DE LA INFORMACIÓN**

**CI2693 – LABORATORIO DE LENGUAJES DE PROGRAMACIÓN I**

**Trimestre:** Abril - Julio 2022

**Profesor:** Fernando Torre

**Informe del Proyecto #1**

Wordle para dos en Haskell

**Estudiantes:**

José González 15-10627

Ana Santos. 17-10602

Sartenejas, Julio del 2022

**Introducción**

Uno de los objetivos del Curso CI-3661 Laboratorio de Lenguajes de Programación es el estudio de lenguajes funcionales, como Haskell. Este lenguaje funcional fue lanzado en 1990 con el objetivo de instituir las bases de los lenguajes de programación funcional a partir de lógica combinatoria. De esta manera, se desea desarrollar para ampliar el conocimiento y capacidades de los estudiantes de Ingeniería de la Computación el juego ***“Wordle para dos”*** en Haskell.

***“Wordle para dos”***es una versión del juego *“Toros y Vacas”* creada por el Ingeniero Josh Wardle en octubre 2021. El juego necesita de dos modalidades llamadas: *“mente maestra”* y “*descifrador”*, el primero se encarga de crear el código para que el CPU lo adivine y el segundo se encarga de adivinar el código administrado por el CPU. La versión del juego en Haskell diseñada como Proyecto para el Curso CI-3661 cumplirá con las especificaciones presentadas en el instructivo inicial.

Para la compresión de la implementación del esta versión en lenguaje funcional, se mostrará tanto las decisiones y herramientas de diseño, junto a las dificultades enfrentadas en la implementación como el análisis y recomendación final para las mejoras de la construcción del programa para el desarrollo del juego.

1. **Decisiones de diseño:**
   1. Diseño general de la solución:

Para el diseño del programa “Wordle para dos” en Haskell, se decidió usar la función principal para interactuar con el usuario, leer el archivo con la lista de palabras, guardar las mismas en una lista y escoger de forma aleatoria la palabra a adivinar. Finalmente realizar las llamadas a las funciones que dirigen las distintas modalidades del juego, “mente maestra” y “descifrador”.

Posteriormente, se realizó el diseño de la función *initMenteMaestra*, la cual se encarga de inicializar este modo, se llama recursivamente así misma durante 6 turnos, o mientras el usuario no adivine la palabra. La función solicita al usuario que ingrese una palabra para evaluarla y también indica si el usuario perdió y cuál era la palabra o si ganó la partida. Para ello, llama a las funciones *mayuscula,* *contains* y *revisar.*

Las funciones *mayuscula* y *contains* son un homólogo a cualquier función en otros lenguajes que realicen Uppercase e indique la contención de un carácter en un String definida para Haskell, respectivamente. Por otro lado, en la función *revisar* toma la palabra insertada por el usuario, verificando carácter por carácter si es un Toro, luego se llama a la función *revisionV.* Esta última verifica si el String evaluado anteriormente contiene Vacas, y en el caso de que el carácter no correspondo ni a un Toro ni a una Vaca se coloca un guion ( - ). Finalmente, este procedimiento se realiza repetitivamente por los turnos restantes.

Para el modo descifrador se definió la función *initDecifrador*, encargada de imprimir una palabra del archivo de manera aleatoria para que el usuario la evalúe para buscar, a partir de ello, la adivinanza más certera. Luego, a partir de la evaluación de Toros, Vacas y guiones, se construye un árbol de 4 niveles con 10 ramas en cada nodo, donde la raíz es la palabra dada por el programa y los niveles se construyen por la definición dada de minimax en el proyecto para encontrar la palabra correcta, esto se hace recursivamente durante 6 turnos, de lo contrario el programa pierde. Esta función realiza llamadas a las funciones *revisarToros, revisarVacas, splitAllWords, filterByToros, filterByVacas y obtenerPuntajesTot.*

La función *revisarToros* se encarga de retornar un tuplas de Toros con el caracter y la posición donde se encuentran la misma en la palabra evaluada, a partir del String de evaluación. La función *revisarVacas* realiza lo respectivo para las Vacas en la palabra. La función *splitAllWords* se encarga de separar todas las palabras del archivo en listas de caracteres para poderlas comparar y armar el árbol que se necesita.

Por otro lado, la función *filterByToros* se encarga de tomar únicamente las palabras que posean Toros en la misma posición que marca el String de evaluación, (información que se tiene reservada en las tuplas anteriormente mencionadas). La función *filterByVacas*, realiza lo respectivo para la Vacas sólo que se consideran todas las palabras que incluyan el caracter menos, en la posición que se encontraba.

Finalmente, la función *obtenerPuntajesTot* es utilizada para asignar el puntaje a las lista de caracteres, generando una lista de tuplas con los caracteres y sus respectivos puntajes; para ello, llama a las funciones *obtenerPuntajeW* y *asignarPuntaje.* En la primera, se asigna el puntaje a cada palabra y retorna una tupla de String y Floar, y la segunda, dado un caracter se retorna un puntaje tipo Float.

* 1. Módulos de Haskell utilizados como librerías:

1. Data.List.:
2. Data.List(sort)
3. Data.List.Split
4. Data.Char
5. System.Enviroment
6. System.Random
7. **Dificultades en la implementación:**

Durante la implementación del programa en Haskell, se encontraron las siguientes dificultades:

* Compresión de Haskell como lenguaje funcional, debido a que los programadores de este proyecto se encuentran mayormente familiarizados con lenguajes imperativos, desarrollados en materias anteriores.
* Manejo de frecuente de recursiones para la implementación de funciones que representan ciclos convencionales como *while-loops* y *for-loops* en lenguajes imperativos.
* Compresión de Haskell como un lenguaje fuertemente tipado para la escritura de los tipos de una función y su implicación en el aprendizaje de los programadores para su correcta aplicación.
* Construcción de un Árbol de 10 ramas y 4 niveles, debido a la necesidad de una construcción distinta para los nodos de cada nivel. Por tanto, representa un reto la creación de una recursión de las funciones para lograr el objetivo, entendiendo que la primera dificultada en lo fuertemente tipado del lenguaje.

1. **Optimizaciones:**

**Conclusiones y Recomendaciones**

**Referencias**