

# Loki Text Adventure

SEGUNDO PROYECTO DE PROGRAMACIÓN DECLARATIVA



**Equipo de desarrollo:**  
Carlos Toledo Silva C-311  
Ariel Alfonso Triana Pérez C-311

2021

# 1 Introducción

En el siguiente documento se realizará la presentación de la aplicación **Loki Text Adventure**. Para esto se divide el resto del documento en secciones. En una primera sección se presenta la historia elaborada para la aplicación y en una posterior sección se explica cómo un usuario interactúa con la aplicación. Posteriormente se explica la implementación de la aplicación: estrategia que se siguió, descripción de los módulos implementados y cómo se aprovecharon las características propias de Haskell para dicha implementación.

## 2 Historia elaborada para la aplicación

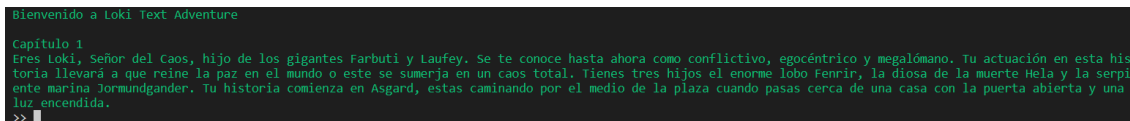
La historia que se narra en la aplicación está basada en la mitología nórdica. El usuario asume el papel de Loki: un dios embaucador de esta mitología y uno de los dioses más conocidos. El usuario, metido en el papel de Loki, tomará parte en una historia en la que pasará por diferentes aventuras y dificultades, donde las decisiones que tome y los actos que realicen tendrán una influencia directa en la historia del mundo.

Válido aclarar que aunque la historia está basada en la mitología nórdica, muchos de los pasajes que en ella se narran no ocurren exactamente igual, ni siquiera en el orden correspondiente a como están realmente recogidos en la mitología. Otros muchos ni siquiera ocurrieron y son invención de los autores. De esta forma se ha adaptado la historia de Loki y otros personajes nórdicos para lograr hacer una historia entretenida y atractiva para el usuario.

La historia está compuesta de 3 capítulos, seguidos uno a continuación del otro. Es decir que para el usuario poder llegar el capítulo  $i + 1$  deberá haber triunfado en el capítulo  $i$ . Además cada capítulo cuenta con varios finales; en algunos el jugador sale triunfante pero en otros este podría acabar perdiendo la vida. Por esto motivo el jugador deberá escoger sabiamente, si es que quiere avanzar lo más posible.

## 3 Interacción entre el jugador y la aplicación

Cuando se ejecuta la aplicación (ya compilada) lo primero que se ve es lo siguiente:



```
Bienvenido a Loki Text Adventure
Capítulo 1
Eres Loki, Señor del Caos, hijo de los gigantes Farbauti y Laufey. Se te conoce hasta ahora como conflictivo, egocéntrico y megalómano. Tu actuación en esta historia llevará a que reine la paz en el mundo o este se sumerja en un caos total. Tienes tres hijos el enorme lobo Fenrir, la diosa de la muerte Hela y la serpiente marina Jormundgander. Tu historia comienza en Asgard, estas caminando por el medio de la plaza cuando pasas cerca de una casa con la puerta abierta y una luz encendida.
>> |
```

Figure 1: Inicio de la historia

Aquí es donde comienza la historia. A medida que el personaje avance en la historia le irán saliendo textos similares a estos. Para poder avanzar el usuario deberá utilizar la información mostrada en cada uno de los textos y dar una respuesta lógica y bien escrita mediante alguna acción. Es importante señalar que las acciones se deben expresar mediante infinitivos: *entrar*, *aceptar*, *hablar*, etc y que por supuesto estos se combinan con las otras palabras necesarias para elaborar una idea coherente. También es importante recalcar que, las acciones sobre el propio jugador deben ir seguidas del pronombre personal *me*; ejemplo: *transformarme*, *quedarme*, etc.

Por ejemplo, partiendo del texto inicial de la historia (Fig 1) se puede hacer lo siguiente:

```

Bienvenido a Loki Text Adventure

Capítulo 1
Eres Loki, Señor del Caos, hijo de los gigantes Farbuti y Laufey. Se te conoce hasta ahora como conflictivo, egocéntrico y megalómano. Tu actuación en esta historia llevará a que reine la paz en el mundo o este se sumerja en un caos total. Tienes tres hijos el enorme lobo Fenrir, la diosa de la muerte Hela y la serpiente marina Jormundgander. Tu historia comienza en Asgard, estas caminando por el medio de la plaza cuando pasas cerca de una casa con la puerta abierta y una luz encendida.
>> Entrar en la casa
Al entrar reconoces que es la casa del dios Thor y su esposa Sift, la cual parece estar en su habitación.
>>

```

Figure 2: Avance de la historia basado en la decisión del usuario

Obsérvese cómo al escribir una acción lógica correctamente se ha avanzado en la historia. Si por el contrario se escribe una acción sin sentido o no escrita correctamente (ya sea por palabras mal escritas o falta de las mismas), la aplicación puede que no reconozca lo que el usuario quiere decir y si ocurre este caso entonces se alertará con un mensaje y volverá a imprimir el último texto de la historia. A continuación se muestra un ejemplo:

```

Bienvenido a Loki Text Adventure

Capítulo 1
Eres Loki, Señor del Caos, hijo de los gigantes Farbuti y Laufey. Se te conoce hasta ahora como conflictivo, egocéntrico y megalómano. Tu actuación en esta historia llevará a que reine la paz en el mundo o este se sumerja en un caos total. Tienes tres hijos el enorme lobo Fenrir, la diosa de la muerte Hela y la serpiente marina Jormundgander. Tu historia comienza en Asgard, estas caminando por el medio de la plaza cuando pasas cerca de una casa con la puerta abierta y una luz encendida.
>> Entrar a la casa
Acción inválida o irreconocible. Quizás faltan palabras o la acción no es esperada. Chequea también la ortografía de las palabras. No avance
Bienvenido a Loki Text Adventure

Capítulo 1
Eres Loki, Señor del Caos, hijo de los gigantes Farbuti y Laufey. Se te conoce hasta ahora como conflictivo, egocéntrico y megalómano. Tu actuación en esta historia llevará a que reine la paz en el mundo o este se sumerja en un caos total. Tienes tres hijos el enorme lobo Fenrir, la diosa de la muerte Hela y la serpiente marina Jormundgander. Tu historia comienza en Asgard, estas caminando por el medio de la plaza cuando pasas cerca de una casa con la puerta abierta y una luz encendida.
>>

```

Figure 3: No se reconoce la decisión del jugador

Fíjese que cuando salió el cartel por primera vez el usuario escribió mal la palabra “Entrar” y por este motivo la acción no se pudo reconocer.

También durante el juego aparecerán situaciones en las que el usuario tendrá que escoger entre varias opciones. De esta forma la aplicación posibilita que el usuario pueda transitar por diferentes pasajes en base a sus decisiones. Cuan larga pueda ser la experiencia del jugador, también dependerá de las decisiones que tome. Una “mala” decisión, por decirlo de alguna forma, podría provocar que el usuario termine el juego antes de completar los 3 capítulos. Se muestra un ejemplo donde el usuario se enfrenta a una decisión:

```

Thor se da cuenta que llevas la mayor parte de la cabellera de Sift escondida en una bolsa. Ahora tienes tres opciones : convencer a Thor de que puedes arreglar el daño que has hecho, huir de él o enfrentarlo.
>>

```

Figure 4: Momento de trifurcación de la historia, en dependencia de la decisión del jugador

En este caso el jugador podrá escoger entre convencer, hablar o enfrentar a Thor. En dependencia de lo que decida el usuario se moverá a uno u otro pasaje.

## 4 Implementación

A continuación se explica la estrategia de implementación de la aplicación, cómo se utilizaron las ventajas de Haskell como pattern matching en el desarrollo y se explica la funcionalidad de cada módulo de la aplicación.

## 4.1 Estrategia de implementación

Como toda historia, la presente está compuesta por diferentes pasajes. Por este motivo se define un tipo **Passage** para representar los diferentes pasajes. Más adelante se verá como está implementado este tipo. En todo momento el jugador se va encontrar transitando en un determinado pasaje. Para cambiar de pasaje, lo cual provoca un avance en la historia, el jugador introduce una sentencia, a la cual se realiza un análisis para determinar si se cambia a algún otro pasaje o si la historia se mantiene en el mismo. Cuando se alcanza algún pasaje “final” la ejecución de la aplicación se detiene.

## 4.2 Módulos implementados

La implementación de la aplicación, para mejor organización y comprensión, está dividida en los siguientes módulos:

- **Passages**: contiene la definición del tipo **Passage**, y todos los pasajes de la historia.
- **Synonyms**: contiene la definición del tipo **Synonyms**, y todos los sinónimos utilizados en la historia.
- **Functions**: contiene la implementación de las funciones que permiten llevar el pasaje actual, el análisis de la entrada del usuario y el cambio o no de un pasaje a otro.
- **Loki-Text-Adventure**: que apoyándose de los otros módulos permite el funcionamiento de la aplicación.

A continuación se verán al detalle estos módulos y su funcionamiento.

### 4.2.1 Módulo Passages

Como se mencionó anteriormente, en este módulo está definido el tipo **Passage** del cual veremos ahora su definición.

```
data Passage = Passage {  
    pid :: Int,  
    text :: String,  
    keywords :: [String],  
    nextPossiblePassages :: [Passage]  
}
```

El entero **pid** se utiliza para comparar pasajes de una forma eficiente. La cadena **text** es el texto correspondiente a un pasaje. Esto es lo que se imprime en pantalla cuando se llega a dicho pasaje. La lista de **Passage** **nextPossiblePassages** son los pasajes a los cuales se puede avanzar a partir del pasaje actual. La lista de cadenas de texto **keywords** son las palabras que debe teclear el usuario (o algunas que sean sinónimos de estas o que transmitan una idea similar) que permiten a partir de un pasaje específico alcanzar el pasaje con esas keywords. O sea, si el usuario se encuentra en el pasaje *i* y en la lista **nextPossiblePassages** del pasaje *i* se encuentra el pasaje *j*, para que el usuario pueda pasar del pasaje *i* al pasaje *j* deberá teclear las palabras que aparecen en la lista **keywords** del pasaje *j* (o algunas que sean sinónimos de estas o que transmitan una idea similar). De esta forma es que ocurre el cambio entre pasajes.

De la forma antes descrita, aparecen además definidos todos los pasajes de la historia. El pasaje inicial es el único que tiene su lista **keywords** vacía pues este no se alcanza desde ningún otro pasaje. Los pasajes finales, o sea los que no permitan avanzar al jugador más allá de ellos, se caracterizan porque su lista de pasajes **nextPossiblePassages** es una lista vacía.

#### 4.2.2 Módulo Synonyms

Como se había mencionado al inicio de esta sección, el módulo **Synonyms** contiene la definición de un tipo del mismo nombre que el módulo. Esta definición se presenta a continuación:

```
data Synonyms = Synonyms {  
    key  :: String ,  
    sym  :: [String ]  
} deriving (Eq,Show)
```

Con este tipo se puede definir un “diccionario de sinónimos”, donde la cadena de texto **key** representa la palabra de la cual se desean conocer los sinónimos y la lista de cadenas **sym** representa los sinónimos de esa palabra.

Utilizando este tipo se precisó un diccionario de sinónimos para la historia donde los campos **key** son las **keywords** del tipo **Passage** en cuestión, y **sym** representa el todas las palabras que pueden ser utilizadas en lugar de **key**. Todos estos sinónimos se fijan en la lista **allSynonyms**.

#### 4.2.3 Módulo Functions

En este módulo se encuentran definidas las funciones que permiten el correcto funcionamiento de la aplicación. Este módulo depende de los anteriores para su correcto funcionamiento.

La función **actualPassage** permite la creación de una variable global en la que se guardará en todo momento el pasaje en el cual se encuentra la historia.

La función **changePassage** recibe como parámetro un **Passage** y lo que hace es definir al pasaje actual como el pasaje que se le pasa de entrada.

La función **obtainPassage** se utiliza para poder obtener el pasaje actual en el que se encuentra la historia.

La función **next** se utiliza para obtener el siguiente pasaje de la historia dado el pasaje actual y la entrada del usuario. El funcionamiento es el siguiente se construye una lista que contiene los pasajes que son “descendientes directos” del pasaje actual en la historia, cuyas **keywords** están de forma exacta o a través de sinónimos en la entrada del usuario. Si esta lista no contiene elementos entonces se devuelve el pasaje actual, pues la decisión del usuario no se corresponde con las opciones disponibles. Si la lista contiene elementos esto quiere decir que con esa entrada se puede alcanzar cualquiera de los pasajes en la lista por tanto se devuelve el primer elemento. Dicha función utiliza **identifyKeywords** que realiza el procedimiento de chequear si la entrada del usuario contiene las **keywords** del pasaje o sus sinónimos. Para esto utiliza **getSym** que devuelve todos los sinónimos de cada **keywords** del pasaje, y además utiliza **find** que devuelve un booleano indicando si una cadena de texto está en una lista de cadenas de texto.

#### 4.2.4 Módulo Loki-Text-Adventure

Este es el módulo principal de la aplicación. En este módulo primero tenemos dos funciones “**isWindows**” y “**setEncoding**”. La primera la utilizamos para saber si la aplicación se está ejecutando en Windows o no. La segunda es se utiliza para cambiar en el formato de codificación de caracteres

a UTF-8 en caso de que no estemos en Windows. Esto lo hacemos porque nos paso que uno de nosotros programó en Windows 10 y el otro en Ubuntu 18.04 y si no cambiamos el formato de codificación de caracteres en Ubuntu la aplicación no reconocía los caracteres del Español como las tildes y la ñ. Sin embargo si cambiamos esta codificación en Windows ocurría entonces que dichos caracteres no se reconocían y al no cambiarlo si lo hacían. Por tanto se optó por comprobar si se estaba ejecutando la aplicación en Windows para cambiar o no la configuración de codificación de caracteres.

Entonces tenemos también la función "main", la principal de toda la aplicación. En esta función primeramente en dependencia del sistema operativo se cambia o no la configuración de codificación de caracteres. Luego se obtiene el pasaje actual mediante la función "obtainPassage", se guarda en la variable "passage" y de este se imprime su texto correspondiente. Luego, a menos que este sea un pasaje final, se espera a que el usuario propicie una entrada. En caso de ser un pasaje final se termina la ejecución. Entonces si la entrada propiciada por el usuario es "salir" se detiene la ejecución; en cualquier otro caso se pasa a obtener el siguiente pasaje mediante la función "next" y se guarda en la variable "auxPassage". Entonces comprobamos que si el pasaje actual y el hallado por la función coinciden. De ser así es porque el usuario no tecleo una entrada que permitiera el avance a un nuevo pasaje y en este caso se le escribe un mensaje de advertencia con las posibles causas del fracaso en el avance y se llama al método "main". En caso de que no coincidan, es porque hubo un avance y por tanto se actualiza el estado actual mediante la función "changePassage" utilizando como parámetro "auxPassage". Luego de esto llamamos al método "main". Obsérvese que la aplicación solo termina su ejecución si se alcanza un pasaje final o el usuario tecla la palabra "salir".